Doctoral Program in Pure and Applied Mathematics (35th cycle)

Cryptographic Innovations in Blockchain

Algorithms for blockchain technology

By

Andrea Gangemi

******

Supervisor:
Prof. Danilo Bazzanella

PhD chairman:
Prof. Anna Maria Fino

DISMA "G.L. Lagrange" - Politecnico di Torino
Dip. di Mat. "G. Peano" - Università di Torino

MAT/02 - ALGEBRA

2019-2023

Abstract

Blockchain is a disruptive technology that has gained prominence in recent years, after Satoshi Nakamoto published the well-known paper that gave birth to Bitcoin. One of the reasons blockchains are attractive is the use of cryptography, which is utilised in particular to ensure scalability, privacy and security. Despite their success, blockchains are still relatively young, which is why new protocols are constantly being born with the goal of improving the points listed above. In addition, since blockchains use elliptic curves cryptography for key generation, another open problem is the introduction of post-quantum algorithms that do not compromise the scalability of the blockchain itself. This work focuses on four different proposals aimed at improving or analyzing some of the currently used algorithms. First, TRIFORS, a ring signature based on a modern cryptographic assumption that is considered post-quantum, namely the equivalence between alternating trilinear forms, is described. This signature is heavier than that produced by algorithms based on pre-quantum assumptions, but it is still competitive with the state-of-the-art of other post-quantum proposals. Second, a protocol that can be seen as a generalization of Bitcoin's Proof-of-Work is described. To insert a block, the network does not have to find a single nonce, but must find a few. This simple modification allows for a more equitable distribution of rewards and at the same time has the effect of regularizing the time of block insertion. Next, an idea for a new dispute resolution protocol that can be built on the Ethereum blockchain is presented. In this case, privacy is ensured by design through the use of the zero-knowledge protocols Semaphore and MACI (Minimal Anti-Collusion Infrastructure), which provide, among other things, resistance to Sybil-type and collusion attacks. These two protocols are based on zk-SNARKs, a family of succinct zero-knowledge cryptographic algorithms that has gained much prominence recently for ensuring scalability and privacy in decentralised contexts. The idea is also among the first in the literature to introduce social governance rather than one based on economic incentives, through the use of soulbound tokens. Finally, the security of some addresses generated on the secp256k1 elliptic curve, used in particular by Bitcoin and Ethereum, is analyzed. In particular, this paper shows that the weak keys found in a previous work are most likely due to a faulty implementation of the wallet and not to an inherent weakness in the cryptographic protocols used.

Contents

List of Figures

List of Tables

Part I
What do we already know?

Cryptography is a constantly evolving field. In recent years, major steps forward have been made in terms of both theory and practical applications. One example is the increasingly widespread use of blockchain technology. This first part summarises the theoretical foundations on which the innovative part of this thesis is based.

Chapter 1

Introduction

Cryptography is the study of techniques for secure communication in the presence of adversarial behaviour. It can be divided into two macroareas: *symmetric cryptography* [24] and *asymmetric cryptography* [48].
The first category was the only one known and used up to the 1970s, and it uses the same secret key to encrypt and decrypt a message. Asymmetric (or public key) cryptography instead uses two different keys: a *public key* to encrypt a message, and a *secret key* to decrypt it. It was born in 1976, when Diffie and Hellman wrote about the possibility to use mathematics to link these two keys. In particular, it must be computationally intractable to derive the secret key knowing only the public key, while the generation of the latter starting from the former is straightforward.
There are many different assumptions that can be used to build secure schemes: one well-known example is the *Discrete Logarithm Problem* assumption [96]. Given a finite cyclic group G, generated by an element g, and any other element of the group a, we know that for sure the relation $g^x = a$ for some $x \in \mathbb{Z}$ holds. However, knowing the elements g and a, the computation of x is not computationally efficient if the cardinality of G is big enough. In this context, x represents the secret key, while $a = g^x$ is instead the public key.
More recently, *Elliptic Curve Cryptography* (ECC) [82, 84, 127] became predominant since it provides the same level of security of classic DLP systems, using much smaller keys. ECC security is based on a generalization of the DLP, known as *Elliptic Curve Discrete Logarithm Problem* [128]. Given an elliptic curve E defined on a finite field $\mathbb{F}_q$, and given two points of the curve G and P, it is unfeasible to find the value d such that $P = dG$, where the notation dG denotes the sum of the point G

with itself $d-1$ times.
One of the most important cryptographic primitives is the *digital signature* [48], which provide *non-repudiation*, meaning that the signer of a document that effectively used his private key cannot claim that he did not sign that document. Digital signatures are a crucial cryptographic primitive for various applications; one example is given by *blockchains* [103], which are decentralized, distributed and public digital ledgers that are used to record transactions. Blockchain keys are generated exploiting elliptic curve cryptography and they are saved into a *wallet*. Every user is represented by an *address*, which is simply the hash of the respective public key. In this context, a digital signature certifies that a transaction has occurred correctly.
In recent years, more advanced signature protocols such as *multisignatures* or *ring signatures* are among the most popular research topics, and blockchain is precisely one of their main applications. Another essential aspect is that these signatures should be based on assumptions that are assumed to be resistant to quantum computing. In fact, over the past few years, NIST [37] has been trying to standardise digital signature algorithms based on post-quantum assumptions. During 2022, some protocols were indeed standardised, but the search for algorithms that are even more competitive but still secure continues.
Another family of algorithms with a fundamental role in the functioning of a blockchain are *consensus protocols* [119]: the most famous example is surely *Proof-of-Work* [103]. In this case, millions of nonce are generated in such a way as to find one that, when entered as input to a hash function along with other fixed information, returns a digest that is less than a certain predetermined target. Those who can find this value receive a reward in cryptocurrency, which is thus an incentive for participation in this protocol. A downside of this type of algorithm is that only those who find the nonce get the reward, while the rest of the network gets nothing. This dynamic penalises small users, as they might invest thousands of dollars in equipment, without actually ever being able to win the game.
Even though the security of a blockchain lies in cryptography, and thus in mathematics, conflicting situations can occur: an example can be a cryptocurrency that can be purchased on an exchange but turns out to be a scam. Some decentralised applications that deal with these issues exist [75], but at the moment the techniques used are not optimal for the ecosystem they refer to: for instance, they do not guarantee the privacy of the users involved, and they force parties to accept a decision made by other users.

Finally, for a blockchain to be considered secure, it is essential that the wallets generate the keys, and hence the addresses, in a truly pseudo-random manner. In fact, if one is not careful enough, it is possible to trace the aforementioned key from the very address, exploiting the structure of the group of points of an elliptic curve defined over a finite field. Tracing the secret key means being able to spend the cryptocurrency contained in its address on behalf of someone else.

The first part of the thesis is devoted to a summary of the mathematical and cryptographic tools used in the second part of the work. In more detail, Chapter 2 describes some cryptographic algorithms such as commitment schemes, digital signatures and sigma protocols, while Chapter 3 recaps how a blockchain works and how cryptography comes into play within this technology. Finally, in Chapter 4 some additional tools that will be crucial in the second part of this work are described, such as tensors or quadratic voting.
The remaining chapters are devoted to the original results and are therefore the real focus of this thesis. They are presented from the most theoretical to the most applied results. In particular, Chapter 5 [43] describes a ring signature based on the *alternating trilinear form equivalence* [131], a modern cryptographic assumption believed to be post-quantum. Ring signatures are to date used by blockchains such as Monero [2] and could therefore be a cryptographic primitive used within other blockchains in the future. Chapter 6 [10] describes an easily implemented alternative to the Proof-of-Work protocol used by Bitcoin. Among its advantages, this protocol would allow mining rewards to be distributed more equitably between all parties involved. Chapter 7 [59] describes an idea for a dispute resolution protocol for the Ethereum ecosystem [33] that also provides some level of privacy to the parties involved. Finally, Chapter 8 [47] is a two-way generalization of an attack already studied in 2020 by Sala, Sogiorno, and Taufer [118]. Their techniques allowed them to recover the secret key of some Bitcoin addresses. The first generalization concerns the format used to encode the addresses, while the second one extends the attack to other blockchains that use the same elliptic curve.

Chapter 2

Cryptography

This chapter introduces all the cryptographic primitives that will be used in the remainder of this work. Some of the definitions given are not standard, but have been adapted according to the properties and theorems that will be proved in the innovative part of this thesis. More in detail, Section 2.1 describes elliptic curve cryptography, which is the standard in all blockchain implementations, while sections 2.2, 2.3 and 2.4 introduce hash functions, digital and ring signatures, respectively. Some classic protocols typically used in a blockchain context, like ECDSA and LSAG, are also presented. Section 2.5 introduces cryptographic commitments and reports two well-known examples, namely Pedersen commitments and Merkle trees, while Section 2.6 describes sigma protocols, a family of three steps interactive protocols. Finally, Section 2.7 briefly talks about the Fiat-Shamir transform, a tool that is used to convert an interactive protocol into a non-interactive one, and Section 2.8 focuses on zero-knowledge proof algorithms, in particular on zk-SNARKs, that also have interesting properties that make them suitable for blockchain applications.

2.1 Elliptic Curve Cryptography

Elliptic Curve Cryptography (ECC) was independently suggested in 1985 by Neal Koblitz and Victor S. Miller [84, 102]. Its adoption started some years later: the biggest advantage it has over the classic cryptography on finite fields is the possibility to use shorter keys, while maintaining the same security level.

The most used curves are the ones written in the *short Weierstrass form*, over a finite field.

Definition 1 (Short Weierstrass Form Curve)**.** *An* elliptic curve in short Weierstrass form *E, defined over a finite field $\mathbb{F}_q$ with $\mathsf{char}(\mathbb{F}_q) \neq \{2,3\}$, is the set of points (x,y) in the Cartesian plane that satisfies the equation*

$$y^2 = x^3 + ax + b, \tag{2.1}$$

where $a, b \in \mathbb{F}_q$ must fulfill the condition $4a^3 + 27b^2 \neq 0$.

Denote with $E(\mathbb{F}_q)$ the set of points of the curve E on the finite field $\mathbb{F}_q$, that is

$$E(\mathbb{F}_q) = \{(x,y) \in E \mid x, y \in \mathbb{F}_q\} \cup \{\mathscr{O}\},$$

where $\mathscr{O}$ is the so-called *point at infinity* of the curve.
It is well known that the elements of $E(\mathbb{F}_q)$ satisfy a group law with respect to a certain operation $+$. Given the points $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$, the possible cases are the following:

- If $P \neq Q$, then the point $R = P + Q$ has coordinates

$$x_R = \lambda^2 - x_P - x_Q, \tag{2.2}$$
$$y_R = \lambda(x_P - x_R) - y_P, \tag{2.3}$$

  where $\lambda = \frac{y_Q - y_P}{x_Q - x_P}$ is the angular coefficient of the line passing through P and Q.

- If $P = Q$, then the point $R = P + Q = 2P$ has coordinates

$$x_R = \lambda^2 - 2x_P,$$
$$y_R = \lambda(x_P - x_R) - y_P,$$

  with $\lambda = \frac{3x_P^2 + a}{2y_P}$.

- If there is not a third intersection with the curve, that is the line passing between P and Q is parallel to the y-axis, then $P + Q = \mathscr{O}$, the point at infinity, which also represents the identity element of the group.

The sum of two points can be computed efficiently with the *Double-and-Add* algorithm [67].
Another category of curves used for cryptography applications are known as *Twisted Edward Curves* [15].

Definition 2 (Twisted Edwards Curve). *A* twisted Edwards curve *E, defined over a finite field $\mathbb{F}_q$ with* $\mathsf{char}(\mathbb{F}_q) \neq 2$*, is the set of points (x,y) in the Cartesian plane that satisfies the equation*

$$ax^2 + y^2 = 1 + dx^2y^2, \tag{2.4}$$

where a, $d \in \mathbb{F}_q$.

The elements of $E(\mathbb{F}_q)$ satisfy a group law with respect to a certain operation $+$. Given the points $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$, then the point $R = P + Q$ has coordinates

$$x_R = \frac{x_P y_Q + x_Q y_P}{1 + dx_P x_Q y_P y_Q},$$
$$y_R = \frac{y_P y_Q - ax_P x_Q}{1 - dx_P x_Q y_P y_Q}.$$

The advantages of this form consist in not having to distinguish between 3 different cases depending on the points P and Q and in a more efficient point sum operation, thus enabling faster cryptographic algorithms.
For more details, the interested reader can look at [102, 127, 82].

2.2 Hash Functions

Definition 3 (Hash function). *Let Σ be an alphabet and let Σ^* be the set of all words (of arbitrary length) obtained from Σ. A* hash function *is a function $h : \Sigma^* \to \Sigma^n$, for some fixed n. A couple of elements $a, b \in \Sigma$ such that $h(a) = h(b)$ is called* collision.

Most applications use $\Sigma = \{0, 1\}$ and $n = 160, 256$ or 512. The value $h(x)$ is called hash or digest of x; it follows from its definition that $h(x)$ cannot be injective.

Hash functions must satisfy the following properties to be suitable for cryptography:

- *Avalanche effect*: changing a single bit in the input must change most of the bits of the output;
- *One-way*: given the value of the digest $h(x)$, it must be computationally unfeasible to compute the input x;
- *Collision resistant*: the research of any collision (a, b) is computationally unfeasible;
- *second preimage resistant*: Fix $a \in \Sigma^*$. Then, the computation of a element $b \in \Sigma^*$, $b \neq a$, such that $h(a) = h(b)$ is computationally unfeasible.

Clearly, collision resistance implies second preimage resistance. Finding any two inputs which hash to the same value should require work equivalent to about $2^{\frac{n}{2}}$ hash computations. In fact, a well-known attack exploits the birthday paradox [135] to generate collisions.

Definition 4 (Cryptographic hash function)**.** *A* cryptographic hash function *is a hash function that enjoys the avalanche effect, one-way and collision resistant properties.*

There are many hash functions that are used in cryptographic applications. Some examples are *RIPEMD160* [49], *SHA256* [111] and *Keccak256* [16]. .

2.2.1 The Random Oracle Model

The *random oracle model* (ROM) assumes the existence of a publicly-accessible oracle that everyone can access, which works basically as a black box. This oracle is assumed to implement a completely random function. Their instantiation in practical applications is given by a cryptographic hash function, since truly random oracles cannot exist.
The ROM is useful to do security proofs. The idea is the following: if an adversary $\mathscr{A}$ is able to break a scheme that was assumed secure under the ROM, that is a scheme whose security has been analysed in a framework where hash functions are treated as random oracles, then any cryptographic hash function used in practice for this application would not be enough. Hence, a proof done under this model gives some trust, since it is assumed that cryptographic hash functions are secure if implemented correctly.

Clearly, in the reality there is no way to achieve a truly random function, and there are examples of algorithms that are secure under the ROM, but are then insecure when cryptographic hash functions are used; however, this proof is most of the times enough to obtain some confidence on the security of the scheme.

2.3 Digital Signatures

Definition 5 (Digital Signature)**.** *A* digital signature *scheme is a quadruple of probabilistic polynomial time (PPT) algorithms* Setup, KGen, Sign *and* Verify *such that:*

- Setup*: it takes as input the security parameter* λ *and it returns the public parameters* pp *used by the scheme;*
- KGen*: it takes as inputs the public parameters* pp *and some random coins* rr*. It returns the secret/public key pair* $(\mathsf{sk}, \mathsf{pk})$*;*
- Sign*: it is a probabilistic algorithm. It takes as inputs the secret key of the signer* sk *and the message* msg*. The output of the algorithm is the signature* σ*;*
- Verify*: it is a deterministic algorithm. It takes as inputs the public key* pk*, the message* msg *and the signature* σ*. It returns 1 (accept) if the signature is valid and 0 (refuse) otherwise.*

Definition 6 (Correctness)**.** *A digital signature is* correct *if for every security parameter* $\lambda \in \mathbb{N}$*, for every public key* pk*, and for every message* msg*,*

$$\mathbf{P}\left[\mathsf{Verify}(\mathsf{pk}, \mathsf{msg}, \sigma) = 1 \;\middle|\; \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda), \\ (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KGen}(\mathsf{pp}, \mathsf{rr}), \\ \sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, \mathsf{msg}). \end{array}\right] = 1.$$

Informally, this means that the verification algorithm of a signature generated correctly will always output 1.

2.3.1 An example: ECDSA

One digital signature that is highly prevalent in the blockchain landscape is the *Elliptic Curve Digital Signature Algorithm* (ECDSA). Its algorithms are the following:

- Setup: given a security parameter λ, this algorithm returns the public parameters $\mathsf{pp} \leftarrow (E, \mathbb{F}_q, G, n, h)$, where E is the elliptic curve defined over the finite field $\mathbb{F}_q$, G is a base point of the elliptic curve of prime order n, and h is a hash function;

- KGen: given the public parameters pp, this algorithm returns the secret/public key pair of the user $(\mathsf{sk}, \mathsf{pk}) \leftarrow (d, P)$, where $d \leftarrow\$\, \mathbb{F}_n^*$ and $P = dG$ is a elliptic curve point;

- Sign: the sender computes the digest of the message $h \leftarrow h(\mathsf{msg})$, then chooses $k \leftarrow\$\, \mathbb{F}_n^*$ and computes the point $R = kG$. Afterwards, the sender takes the x-coordinate of the point r and computes the value $s \leftarrow (h + r \cdot \mathsf{sk})k^{-1} \mod n$. The output of the Sign algorithm is $\sigma \leftarrow (r, s)$;

- Verify: the verifier, knowing the message msg, the signature σ, and the public key of the sender pk, computes the digest $h \leftarrow h(\mathsf{msg})$ and the value $w \leftarrow s^{-1} \mod n$, and then calculates the point $Q \leftarrow whG + wr \cdot \mathsf{pk}$. Finally, the verifier checks if the x-coordinate of the point Q is equal to r. If this happens, the signature is valid.

2.4 Ring signatures

Ring signatures were introduced in 2001 by Rivest, Shamir and Tauman [117]. They are a simplified variant of *group signature schemes* [35] and are useful when the involved members do not want to cooperate. A key element in these schemes is the *ring*: a set of R public keys which belong to different users. The signature is then produced by a single user, exploiting all the R public keys of the ring.
In a ring signature, after the key generation phase, where each user will receive a secret/public key pair $(\mathsf{sk}, \mathsf{pk})$, the signer I will produce the signature σ starting from a message msg, the ring containing the R public keys, and his secret key sk_I. The verifier will then be able to check the correctness, knowing only the message msg,

the ring and the signature σ. Moreover, a ring signature can also be *linkable*. In this case, an additional value τ will be produced during the sign phase. This value will not change if it is computed starting from the same secret key, so, if the same user produces two different signatures, they will be linked.

This section defines more formally what a ring signature is and some additional properties that they usually satisfy.

Definition 7 (Ring Signature). *A* ring signature *consists of four probabilistic polynomial-time (PPT) algorithms* Setup, KGen, Sign *and* Verify *such that:*

- Setup*: it takes as input the security parameter of length* λ *and it returns the public parameters* pp *used by the scheme;*
- KGen*: it takes as inputs the public parameters* pp *and some random coins* rr*. It returns the secret/public key pair* $(\mathsf{sk}, \mathsf{pk})$*;*
- Sign*: it takes as inputs a secret key* sk_I*, where* $I \in \{1, \dots, R\}$ *is the index of the signer in the ring, the message* msg *and the ring* $\Omega = \{\mathsf{pk}_1, \dots, \mathsf{pk}_R\}$*. The public key obtained starting from* sk_I *must belong to the ring, that is* $\mathsf{pk}_I \in \Omega$*. The output of the algorithm is the signature* σ*;*
- Verify*: it takes as inputs the ring* $\Omega = \{\mathsf{pk}_1, \dots, \mathsf{pk}_R\}$*, the message* msg *and the signature* σ*. It returns 1 (accept) if the signature is valid and 0 (refuse) otherwise.*

Ring signatures must satisfy three core properties: *correctness*, *anonymity* and *unforgeability*.

Definition 8 (Correctness). *A ring signature is* correct *if, for every security parameter* $\lambda \in \mathbb{N}$*, for every ring* Ω *composed of* $R = \mathsf{poly}(\lambda)$ *public keys, for every index* $I \in \{1, \dots, R\}$*, and for every message* msg*, then*

$$\mathbf{P}\left[\mathsf{Verify}(\Omega, \mathsf{msg}, \sigma) = 1 \;\middle|\; \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda), \\ (\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{KGen}(\mathsf{pp}, \mathsf{rr_i})\ \forall i \in \{1, \dots, R\}, \\ \Omega := \{\mathsf{pk}_1, \dots, \mathsf{pk}_R\}, \\ \sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_I, \mathsf{msg}, \Omega). \end{array}\right] = 1.$$

Informally, this means that the verification algorithm of a ring signature generated correctly will always output 1.

Definition 9 (Anonymity)**.** *A ring signature is* anonymous *if, for every security parameter $\lambda \in \mathbb{N}$, and for every ring composed of R public keys, any PPT adversary $\mathscr{A}$ has at most a negligible advantage when playing the following game against a challenger:*

(A) The challenger first runs the algorithm Setup *that outputs* pp *and then the algorithm* KGen*, together with random coins* $\mathsf{rr_i}$*, to obtain R secret/public key pairs* $(\mathsf{sk}_i, \mathsf{pk}_i)$*,* $i \in \{1, \ldots, R\}$*. He samples a bit* $b \leftarrow\$ \{0,1\}$*.*

(B) The challenger gives pp *and the list of random coins* $(\mathsf{rr}_1, \ldots, \mathsf{rr}_R)$ *to* $\mathscr{A}$*.*

(C) $\mathscr{A}$ *sends to the challenger a challenge* $(\Omega, \mathsf{msg}, i_0, i_1)$*. The ring* Ω *must contain the public keys* pk_{i_0} *and* pk_{i_1}*. The challenger computes the signature* $\sigma^* \leftarrow \mathsf{Sign}(\mathsf{sk}_{i_b}, \mathsf{msg}, \Omega)$ *and sends it to* $\mathscr{A}$*.*

(D) $\mathscr{A}$ *outputs a bit* b^* *and wins if* $b = b^*$*.*

This property means that it should not be possible to guess the secret key that was used to produce a signature, even if the adversary knows all the secret keys that were used to generate the public keys in the ring.

Definition 10 (Unforgeability)**.** *A ring signature is* unforgeable *if, for every security parameter $\lambda \in \mathbb{N}$, and for every ring composed of R public keys, any PPT adversary $\mathscr{A}$ has at most a negligible advantage when playing the following game against a challenger:*

(A) The challenger first runs the algorithm Setup *that outputs* pp *and then the algorithm* KGen*, together with random coins* $\mathsf{rr_i}$*, to obtain R secret/public key pairs* $(\mathsf{sk}_i, \mathsf{pk}_i)$*,* $i \in \{1, \ldots, R\}$*. He calls* $V = \{\mathsf{pk}_1, \ldots, \mathsf{pk}_R\}$ *the set with the public keys. He finally initialises two empty sets S and C.*

(B) The challenger gives pp *and the set V to* $\mathscr{A}$*.*

(C) $\mathscr{A}$ *can create signing and corruption queries a polynomial number of times:*

- $(\mathsf{AdvSign}, i, \mathsf{msg}, \Omega)$*: the challenger checks if* $\mathsf{pk}_i \in \Omega \subseteq V$*. If that is true, he computes* $\sigma \leftarrow \mathsf{Sign}(\mathsf{pk}_i, \mathsf{msg}, \Omega)$*. The challenger gives* σ *to* $\mathscr{A}$ *and adds* $(i, \mathsf{msg}, \Omega)$ *to S.*

- $(\mathsf{AdvCorrupt}, i)$*: the challenger adds* pk_i *to* C *and returns* $\mathsf{rr_i}$ *to* $\mathscr{A}$.

(D) $\mathscr{A}$ *outputs* $(\Omega^*, \mathsf{msg}^*, \sigma^*)$. *If* $\Omega^* \subset V \setminus C$, $(\cdot, \mathsf{msg}^*, \Omega^*) \notin S$ *and* $\mathsf{Verify}(\Omega^*, \mathsf{msg}^*, \sigma^*) = 1$, *then the adversary wins.*

Finally, this property means that it should be impossible to forge a valid signature without knowing one secret key corresponding to one of the public keys in the ring.

Moreover, a ring signature can have the additional property of being *linkable*, that is everyone can check if two signatures were produced by the same signer (i.e., by the same secret key).

Definition 11 (Linkable Ring Signature). *A* linkable *ring signature is a scheme that consists of the four PPT algorithms previously described for a classic ring signature scheme, plus the following PPT algorithm:*

- Link*: the inputs are two different signatures* σ_0 *and* σ_1. *The algorithm outputs 1 if the two signatures were produced starting from the same secret key and 0 otherwise.*

A linkable ring signature must satisfy the following additional properties: *linkability*, *linkable anonymity* and *non-frameability*. Notice that the correctness property must be slightly modified: if the same user generates two signatures correctly, both the Verify and the Link algorithms will always output 1.

Definition 12 (Linkability). *A linkable ring signature is* linkable *if, for every security parameter* $\lambda \in \mathbb{N}$, *and for every ring composed of* R *public keys, any PPT adversary* $\mathscr{A}$ *has at most a negligible advantage when playing the following game against a challenger:*

(A) The challenger runs the algorithm Setup *that outputs* pp *and gives it to* $\mathscr{A}$.

(B) $\mathscr{A}$ *runs the algorithm* KGen *and outputs the set* $V = \{\mathsf{pk}_1, \dots, \mathsf{pk}_R\}$ *and the set of tuples* $\{(\sigma_1, \mathsf{msg}_1, \Omega_1), \dots, (\sigma_{R+1}, \mathsf{msg}_{R+1}, \Omega_{R+1})\}$.

(C) $\mathscr{A}$ *wins if these three conditions hold:*

- $\forall i \in \{1, \dots, R+1\}$, *then* $\Omega_i \subseteq V$.

- *$\forall i \in \{1, \ldots, R+1\}$, then the algorithm* $\mathsf{Verify}(\Omega_i, \mathsf{msg}_i, \sigma_i)$ *outputs 1.*
- *$\forall i, j \in \{1, \ldots, R+1\}$ such that $i \neq j$, then* $\mathsf{Link}(\sigma_i, \sigma_j) = 0$.

The linkability property guarantees that, if an adversary produces more than k signatures with a set of k public keys, then the Link algorithm will output 1 for at least one pair of signatures.

Definition 13 (Linkable Anonymity)**.** *A linkable ring signature is* linkable anonymous *if, for every security parameter $\lambda \in \mathbb{N}$, and for every ring composed of R public keys, any PPT adversary $\mathscr{A}$ has at most a negligible advantage when playing the following game against a challenger:*

(A) The challenger first runs the algorithm Setup *that outputs* pp *and then the algorithm* KGen*, together with random coins* $\mathsf{rr_i}$*, to obtain R secret/public key pairs* $(\mathsf{sk}_i, \mathsf{pk}_i)$*, $i \in \{1, \ldots, R\}$. He calls $V = \{\mathsf{pk}_1, \ldots, \mathsf{pk}_R\}$ the set with the public keys. He also samples a bit $b \leftarrow\$ \{0, 1\}$.*

(B) The challenger gives to the adversary pp *and the set V.*

(C) The adversary chooses and outputs two public keys $(\mathsf{pk}_0^*, \mathsf{pk}_1^*) \in V$. Denote with $(\mathsf{sk}_0^*, \mathsf{sk}_1^*)$ the respective secret keys.

(D) The challenger gives to $\mathscr{A}$ all the random coins $\mathsf{rr_i}$ *related to the public keys* $\mathsf{pk}_i \in V \setminus \{\mathsf{pk}_0^*, \mathsf{pk}_1^*\}$.

(E) $\mathscr{A}$ queries for signatures, giving as inputs to the challenger a public key $\mathsf{pk} \in \{\mathsf{pk}_0^*, \mathsf{pk}_1^*\}$*, a message* msg *and a ring Ω that contains* pk_0^* *and* pk_1^**:*

- *If* $\mathsf{pk} = \mathsf{pk}_0^*$*, the challenger outputs* $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_b^*, \mathsf{msg}, \Omega)$.
- *If* $\mathsf{pk} = \mathsf{pk}_1^*$*, the challenger outputs* $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_{1-b}^*, \mathsf{msg}, \Omega)$.

(F) $\mathscr{A}$ outputs a bit b^*, and he wins the game if $b = b^*$.

This property says that an adversary cannot guess which secret key was used to produce signatures. Differently from the anonymity property, in this case the adversary does not have access to all the secret keys, otherwise he could use the linkability to understand who was the signer.

Definition 14 (Non-Frameability). *A linkable ring signature is* non-frameable *if, for every security parameter $\lambda \in \mathbb{N}$, and for every ring composed of R public keys, any PPT adversary $\mathscr{A}$ has at most a negligible advantage when playing the following game against a challenger:*

(A) The challenger first runs the algorithm Setup *that outputs* pp *and then the algorithm* KGen, *together with random coins* $\mathsf{rr_i}$, *to obtain R secret/public key pairs* $(\mathsf{sk}_i, \mathsf{pk}_i)$, $i \in \{1, \dots, R\}$. *He calls* $V = \{\mathsf{pk}_1, \dots, \mathsf{pk}_R\}$ *the set with the public keys. He finally initialises two empty sets S and C.*

(B) The challenger gives pp *and the set V to the adversary* $\mathscr{A}$.

(C) $\mathscr{A}$ *can create signing and corruption queries a polynomial number of times:*

- $(\mathsf{AdvSign}, i, \mathsf{msg}, \Omega)$*: the challenger checks if* $\mathsf{pk}_i \in \Omega \subseteq V$. *If that is true, he computes* $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_i, \mathsf{msg}, \Omega)$. *The challenger gives* σ *to* $\mathscr{A}$ *and adds* $(i, \mathsf{msg}, \Omega)$ *to S.*
- $(\mathsf{AdvCorrupt}, i)$*: the challenger adds* pk_i *to C and returns* $\mathsf{rr_i}$ *to* $\mathscr{A}$.

(D) $\mathscr{A}$ *outputs* $(\Omega^*, \mathsf{msg}^*, \sigma^*)$*; he wins if these two conditions hold:*

- $\mathsf{Verify}(\Omega^*, \mathsf{msg}^*, \sigma^*) = 1$ *and* $(\cdot, \mathsf{msg}^*, \Omega^*) \notin S$*;*
- $\mathsf{Link}(\sigma^*, \sigma) = 1$ *for some signature* σ *given by the challenger starting from a query of the form* $(i, \mathsf{msg}, \Omega) \in S$ *with* $\mathsf{pk}_i \in V \setminus C$.

Finally, this property guarantees that it should not be possible for an adversary to create a valid signature that is linked to a signature produced by an honest party.

Several ring signatures were proposed in these years. [117] described two different protocols based on the RSA and Rabin assumption, while Liu, Wei and Wong [91] introduced in 2004 a new group signature algorithm known as *Linkable Spontaneous Anonymous Group* (LSAG), based on the Discrete Logarithm Problem. More digital signatures based on this assumption have been introduced in recent years and have found application for example within the Monero blockchain (See Chapter 3 and [106, 61]). Nowadays, ring signatures schemes have an application in cryptocurrencies and e-voting [112].

2.4.1 An example: LSAG

As an example, the LSAG ring signature is illustrated in the following:

- Setup: given a security parameter λ, this algorithm returns the public parameters $\mathsf{pp} \leftarrow (E, \mathbb{F}_q, G, n, h, H, R)$, where E is the elliptic curve defined over the finite field $\mathbb{F}_q$, G is a generator of the points of the elliptic curve of prime order n, h is a hash function that returns integers, while H is a hash function that returns elliptic curve points. R is instead the cardinality of the ring;
- KGen: given the public parameters pp, this algorithm returns the secret/public key pair of each user i involved in the ring: $(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow (k_i, K_i)$, where $k_i \leftarrow\$\, \mathbb{F}_n^*$ and $K_i = k_i^s G$ is an elliptic curve point;
- Sign: let msg be the message to be signed, $\Omega = \{\mathsf{pk}_1, \dots, \mathsf{pk}_R\}$ the ring, and sk_π the private key corresponding to the public key actually used (denote them $\mathsf{pk}_\pi \in \Omega$). The sender computes the *key image* (which is used for the linkability) $\widetilde{K} = \mathsf{sk}_\pi H(\mathsf{sk}_\pi)$ and then generates $\alpha \leftarrow\$\, \mathbb{F}_q$ and $r_i \leftarrow\$\, \mathbb{F}_q$, with $i \in \{1, \dots, R\}$, $i \neq \pi$.
  Then, the signer computes $c_{\pi+1} = h(\mathsf{msg} \,||\, \alpha G \,||\, \alpha H(\mathsf{pk}_\pi))$; at this point, for each $i = \pi+1, \dots, \pi-1$, he computes
  $$c_{i+1} = h(\mathsf{msg} \,||\, r_i G + c_i \mathsf{pk}_i \,||\, r_i H(\mathsf{pk}_i) + c_i \widetilde{K}).$$
  He sets $r_\pi \leftarrow \alpha - c_\pi \mathsf{sk}_\pi \mod n$, and finally obtains the signature $\sigma \leftarrow (c_1, r_1, \dots, r_n, \widetilde{K})$.
- Verify: the verifier, knowing the message msg, the signature σ, and the ring $\Omega = \{\mathsf{pk}_1, \dots, \mathsf{pk}_R\}$, checks if $n\widetilde{K} = 0$. Afterwards, for each $i = 1, \dots, R$, he computes
  $$c'_{i+1} = h(\mathsf{msg} \,||\, r_i G + c_i \mathsf{pk}_i \,||\, r_i H(\mathsf{pk}_i) + c_i \widetilde{K}).$$
  The signature is accepted if $c'_1 = c_1$.

Security proofs for this signature can be found in [91].

2.5 Commitments

A *commitment scheme* is a cryptographic primitive that allows one to commit to a chosen value while keeping it hidden to others, with the ability to reveal the committed value later.

Definition 15 (Commitment scheme)**.** *A* commitment scheme Π_{Com} *on a message space* $\mathscr{M}$ *is a triple of* PPT *algorithms* $(\mathsf{PGen}, \mathsf{Commit}, \mathsf{Open})$ *such that:*

- $\mathsf{PGen}(1^\lambda)$ *takes as input a security parameter* λ *and returns the public parameters* pp*;*
- $\mathsf{Commit}(\mathsf{pp}, m)$ *takes as input the public parameters* pp *together with a message m in* $\mathscr{M}$*, and returns the commitment* $\mathtt{com}$ *and the* opening material *r;*
- $\mathsf{Open}(\mathsf{pp}, m, \mathtt{com}, r)$ *takes as input the public parameters* pp*, the message m, the commitment* $\mathtt{com}$*, and the opening material r; it returns* accept *if* $\mathtt{com}$ *is the commitment of m or reject otherwise.*

A commitment scheme is secure if it satisfies the properties of *hiding* and *binding*.

Definition 16 (Hiding)**.** *Let* $\Pi_{\mathsf{Com}} = (\mathsf{PGen}, \mathsf{Commit}, \mathsf{Open})$ *be a commitment scheme and let* $\mathrm{Hiding}(\Pi_{\mathsf{Com}})$ *be the hiding game represented in Figure 2.1. A commitment scheme* Π_{Com} *is* computationally hiding *if, for all* PPT *adversaries* $\mathscr{A}$*, there is a negligible function* $\mathtt{negl}(\lambda)$*, with* λ *being the security parameter, such that*

$$\mathbf{P}[\mathscr{A} \textit{ wins } \mathrm{Hiding}(\Pi_{\mathsf{Com}})] \leq \frac{1}{2} + \mathtt{negl}(\lambda).$$

If, for every pair m_0, m_1*, the commitments* $\mathtt{com}_0$ *and* $\mathtt{com}_1$ *have the same distribution, where* $(\mathtt{com}_i, r_i) = \mathsf{Commit}(m_i)$ *for* $i = 0, 1$*, then the commitment is said to be* perfectly hiding.

Definition 17 (Binding)**.** *A commitment scheme* $\Pi_{\mathsf{Com}} = (\mathsf{PGen}, \mathsf{Commit}, \mathsf{Open})$ *is* computationally binding *if, for all* PPT *adversaries* $\mathscr{A}$*, there is a negligible function* $\mathtt{negl}(\lambda)$*, with* λ *being the security parameter, such that*

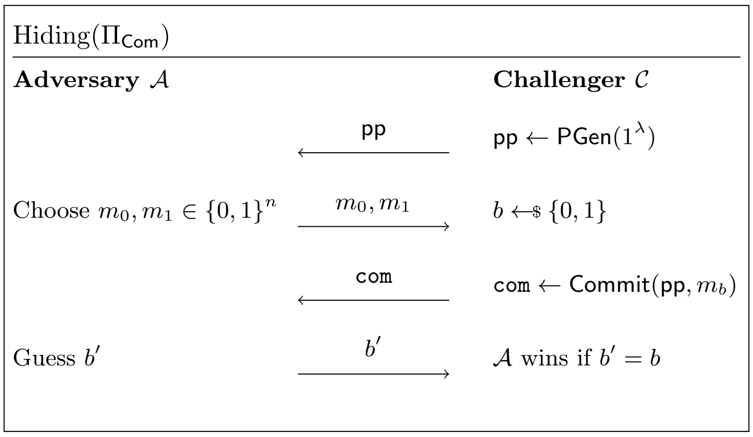

Hiding(Π_{Com})
Adversary $\mathcal{A}$
Challenger $\mathcal{C}$
pp
$\mathsf{pp} \leftarrow \mathsf{PGen}(1^\lambda)$
Choose $m_0, m_1 \in \{0,1\}^n$
m_0, m_1
$b \leftarrow\$ \{0,1\}$
$\mathtt{com}$
$\mathtt{com} \leftarrow \mathsf{Commit}(\mathsf{pp}, m_b)$
Guess b'
b'
$\mathcal{A}$ wins if $b' = b$

Fig. 2.1 Hiding game for commitment schemes.

$$\mathbf{P}\left[(\mathtt{com}, m_0, r_0, m_1, r_1) \leftarrow \mathscr{A}(\mathsf{pp}) \;\middle|\; \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{PGen}(1^\lambda), \\ m_0 \neq m_1, \\ \mathsf{Open}(m_0, \mathtt{com}, r_0) = \text{accept}, \\ \mathsf{Open}(m_1, \mathtt{com}, r_1) = \text{accept} \end{array} \right]$$

is negligible in the security parameter λ. If $\mathtt{negl}(\lambda) = 0$*, then the commitment scheme is said to be* perfectly binding.

Commitment schemes are used in many cryptographic applications. The following two subsections report two classical commitment schemes that are also used in the blockchain environment. The interested reader can check [24] to obtain more information about how commitments work.

2.5.1 Pedersen commitment

The Pedersen commitment [110] is based on the difficulty of the Discrete Logarithm Problem. It has an application on the Monero blockchain [2], to hide the amount of cryptocurrency exchanged in a transaction. Its algorithms are the following:

- PGen: it takes as input the security parameter of length λ and it returns the public parameters pp used by scheme, that is the elliptic curve E defined over a finite field $\mathbb{F}_q$, the base point G, and another point H;

- Commit: to commit a message m (which is encoded as a value $a \leftarrow \mathbb{F}_q$), the sender generates $b \leftarrow\$ \mathbb{F}_q$ and then computes the curve point $C = aG + bH$. The pair (C, b) is the output of the algorithm.
- Open: on input m, C and b, the algorithm outputs accept if the commitment was computed correctly, reject otherwise.

To check that the properties of hiding and binding hold, observe that the relation $H = \gamma G$ is true, but no one knows γ.

- Perfect hiding: the verifier cannot guess a since there exist infinite pairs (a', b') such that $aG + bH = a'G + b'H$.
- Computational binding: the sender cannot know two different values a, a' that commit to the same value C. In fact, he does not know the value γ, so it is computationally hard for him to have two valid values (this is equivalent to resolving the discrete logarithm problem on elliptic curves).

2.5.2 Merkle trees

A Merkle tree [98] is a well known data structure used for cryptography applications. It is a binary tree, where each leaf contains the hashes $\{a_1, \ldots, a_M\}$ of some data that must be hidden, and every other node which is not a leaf is given by the hash of the concatenation of the values of its two children.
In this case, the root of the tree represents the commitment C. Suppose the tree has depth d: to efficiently check that a_i is a leaf of the tree, the prover must send to the verifier, as the opening string o, one information for each level of the tree. The verifier will then compute d different hashes, and check if the final value he obtains equals the committed root C.
A *complete balanced* Merkle tree is a tree where each node has exactly two children, excluding the leaves, whose number is equal to $M = 2^d$ for some positive integer d. Chapter 5 will consider a slight modification of the construction just described, so that the prover, in addition to proving that an element a_i is in the list, does not reveal its position within the tree. This structure is known as *index-hiding Merkle tree*.

It can be easily proved that, if the used hash function is collision resistant, then the obtained commitment is both perfectly hiding and computationally binding.

Merkle trees are used by blockchains to store in the header of a block the root of the hash of all transactions: see Chapter 3, Section 3.1.1, for further details.

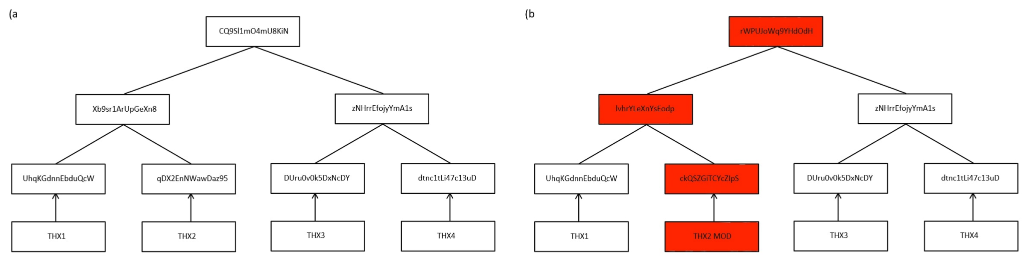


Fig. 2.2 a): A merkle tree. The commitment c is the root of the tree. b): To check that the transaction 1 is in the tree, the prover must reveal the red values, which represent the opening string o.

2.6 Sigma Protocols

Let NP be the set of all the decision problems L which can be efficiently verified.

Definition 18 (NP relation)**.** *For each $L \in$ NP, define the relation*

$$\mathscr{R}(L) = \{(x,w) \mid x \in L \textit{ and } w \textit{ is a witness for } x\}.$$

A relation $\mathscr{R}$ is a NP-relation *if there exists a problem L in* NP *such that $\mathscr{R} = \mathscr{R}(L)$.*

Definition 19 (Sigma protocol)**.** *Given an* NP*-relation $\mathscr{R}$, a* sigma protocol *is a three-move interactive protocol between two* PPT *machines, a prover $\mathscr{P} = (\mathscr{P}_{\mathsf{com}}, \mathscr{P}_{\mathsf{resp}})$ and a verifier $\mathscr{V}$.*

It is assumed that the prover uses some fixed randomness for its algorithms $(\mathscr{P}_{\mathsf{com}}, \mathscr{P}_{\mathsf{resp}})$, and that they share their internal states. The output of $\mathscr{V}$ is assumed to be in $\{0,1\}$. More formally, given a pair $(x,w) \in \mathscr{R}$ where x is the instance and w is the witness for x, the protocol follows the flow in Figure 2.3. The *transcript* of the protocol is defined as the triple $(\mathsf{com}, \mathsf{ch}, \mathsf{resp})$. The challenge ch is sampled from the space S_{ch}.

A sigma protocol must satisfy two well-known properties, *correctness* and *soundness*. Here, it is reported a more specific definition of soundness, known an *special 2-soundness*, that will be used later in Chapter 5.

$$\begin{array}{lcl}
\underline{\textbf{Prover}\,(x,w)} & & \underline{\textbf{Verifier}\,(x)} \\
\mathsf{com} \leftarrow \mathscr{P}_{\mathsf{com}}(x,w) & \xrightarrow{\qquad \mathsf{com} \qquad} & \\
 & \xleftarrow{\qquad \mathsf{ch} \qquad} & \mathsf{ch} \leftarrow\!\!\$\, S_{\mathsf{ch}} \\
\mathsf{resp} \leftarrow \mathscr{P}_{\mathsf{resp}}(x,w,\mathsf{com},\mathsf{ch}) & \xrightarrow{\qquad \mathsf{resp} \qquad} & \textbf{return}\ \mathscr{V}(x,\mathsf{com},\mathsf{ch},\mathsf{resp})
\end{array}$$

Fig. 2.3 Generic Sigma Protocol

Definition 20 (Completeness)**.** *A sigma protocol is* correct *if, for all* $(x,w) \in \mathscr{R}$*, then*

$$\mathbf{P}\left[\mathscr{V}(x,\mathsf{com},\mathsf{ch},\mathsf{resp}) = 1 \;\middle|\; \begin{array}{c} \mathsf{com} \leftarrow \mathscr{P}_{\mathsf{com}}(x), \\ \mathsf{ch} \leftarrow\!\!\$\, S_{\mathsf{ch}}, \\ \mathsf{resp} \leftarrow \mathscr{P}_{\mathsf{resp}}(x,w,\mathsf{com},\mathsf{ch}) \end{array}\right] = 1.$$

Definition 21 (Special 2-soundness)**.** *A sigma protocol has* special 2-soundness *if there exists a polynomial-time algorithm* $\mathscr{E}$ *called extractor such that, given two accepting transcripts*
$(\mathsf{com},\mathsf{ch}_1,\mathsf{resp}_1)$ *and* $(\mathsf{com},\mathsf{ch}_2,\mathsf{resp}_2)$ *with* $\mathsf{ch}_1 \neq \mathsf{ch}_2$*, then the probability*

$$\mathbf{P}\left[(x,w) \in \mathscr{R} : w \leftarrow \mathscr{E}(x,(\mathsf{com},\mathsf{ch}_1,\mathsf{resp}_1),(\mathsf{com},\mathsf{ch}_2,\mathsf{resp}_2))\right]$$

is overwhelming.

A sigma protocol can also satisfy different properties. In particular, the protocol described in Chapter 5 must satisfy *high min-entropy* and *commitment reproducibility*. Their definitions are reported below.

Definition 22 (High min-entropy)**.** *A sigma protocol has* high min-entropy *if, for any* $(x,w) \in \mathscr{R}$*, and any adversary* $\mathscr{A}$*, the probability*

$$\mathbf{P}\left[\mathsf{com}_1 = \mathsf{com}_2 \;\middle|\; \begin{array}{c} \mathsf{com}_1 \leftarrow \mathscr{P}_{\mathsf{com}}(\mathsf{seed},x,w), \\ \mathsf{com}_2 \leftarrow \mathscr{A}(x,w) \end{array}\right]$$

is negligible in the security parameter λ*.*

Definition 23 (Commitment reproducibility)**.** *A sigma protocol is* commitment reproducible *if there exists a PPT algorithm* RecCom *such that, for any pair* (x,w) *in* $\mathscr{R}$*,*

$$\mathbf{P}\left[\mathsf{RecCom}(x,\mathsf{ch},\mathsf{resp})=\mathsf{com}\;\middle|\;\begin{array}{c}\mathsf{com}\leftarrow\mathscr{P}_{\mathsf{com}}(\mathsf{seed},x),\\ \mathsf{ch}\leftarrow\$\,S_{\mathsf{ch}},\\ \mathsf{resp}\leftarrow\mathscr{P}_{\mathsf{resp}}(\mathsf{seed},x,w,\mathsf{com},\mathsf{ch}),\\ \mathscr{V}(\mathsf{com},\mathsf{ch},\mathsf{resp})=1\end{array}\right]$$

is overwhelming in the security parameter λ*.*

This property allows to send only the challenge ch and the response resp, reducing the size of the transcript. The verifier can reconstruct the commitment com using the algorithm RecCom.

2.7 The Fiat-Shamir transform

The Fiat-Shamir transform is a technique used to convert a sigma protocol into a digital signature scheme. It was introduced by Amos Fiat and Adi Shamir in 1986 [58]. The basic idea behind the Fiat-Shamir transform is to substitute the verifier challenge generation with a hash function. This allows the prover to generate a signature that can be verified by anyone who knows the public parameters of the protocol. In particular, the signer acts as a prover running the identification protocol by itself, therefore an interactive protocol is converted in a non interactive one using Fiat–Shamir. Keep in mind that the transform can be used only if the protocol is *public coin*, that is when the random generated by the verifier is also known by the prover in the interactive version of the protocol.
The signer firstly generates a random commitment com, using $\mathscr{P}_{\mathsf{com}}(x,w)$. Then, he generates by himself the challenge by applying a hash function H to the message M and com. This is the main difference with an identification protocol, where the challenge is generated by the verifier instead. Finally, he computes a response $\mathscr{P}_{\mathsf{resp}}(x,w,\mathsf{com},\mathsf{ch})$. The transcript $(\mathsf{com},\mathsf{ch},\mathsf{resp})$ is sent to the verifier.
In the verify phase, the verifier applies a hash function to (M,com), obtaining h, and accepts if $h=\mathsf{ch}$ and $\mathscr{V}(x,\mathsf{com},\mathsf{ch},\mathsf{resp})$ returns accept.

$$\begin{array}{lcl}
\underline{\textbf{Prover}\,(M,x,w)} & & \underline{\textbf{Verifier}\,(M,x)} \\
\mathsf{com} \leftarrow \mathscr{P}_{\mathsf{com}}(x,w) & \xrightarrow{\mathsf{com}} & \\
\mathsf{ch} \leftarrow H(M,\mathsf{com}) & \xrightarrow{\mathsf{ch}} & \\
\mathsf{resp} \leftarrow \mathscr{P}_{\mathsf{resp}}(x,w,\mathsf{com},\mathsf{ch}) & \xrightarrow{\mathsf{resp}} & h \leftarrow H(M,\mathsf{com}) \\
 & & \textbf{return } \mathscr{V}(x,\mathsf{com},\mathsf{ch},\mathsf{resp})
\end{array}$$

Fig. 2.4 The Fiat-Shamir transform

2.8 Zero-knowledge Proofs

The definition of a zero-knowledge proof captures the notion that the verifier should learn nothing from the prover other than the validity of the statement being proven. In other words, any information the verifier learns by interacting with the honest prover could be learned by the verifier on its own. This is formalised in the definition using a PPT algorithm, called *simulator*, which, given only as input the statement x to be proved, produces a transcript that is indistinguishable from the transcript produced when the verifier interacts with an honest prover.
A sigma protocol can also satisfy the zero-knowledge property. The following reports the definition of *Special zero-knowledge*, because this is the property that the signature in chapter 5 must satisfy.

Definition 24 (Special zero-knowledge)**.** *A sigma protocol has* special zero-knowledge *if there exists a PPT algorithm $\mathscr{S}$, the* simulator*, such that, for any $(x,w) \in \mathscr{R}$, $\mathsf{ch} \in S_{\mathsf{ch}}$, and any adversary $\mathscr{A}$ making at most a polynomial number of queries, if $\mathscr{P}$ denotes the pair of algorithms $(\mathscr{P}_{\mathsf{com}}, \mathscr{P}_{\mathsf{resp}})$, then*

$$|\mathbf{P}[\mathscr{A}(\mathscr{P}(x,w,\mathsf{ch})) = 1] - \mathbf{P}[\mathscr{A}(\mathscr{S}(x,\mathsf{ch})) = 1]|$$

is negligible in the security parameter λ.

2.8.1 zk-SNARKs

Zk-SNARK stands for *zero knowledge Succinct Non Interactive ARguments of Knowledge* and it is a class of zero-knowledge algorithms that is particularly suitable for blockchain applications. In fact, succinctness means that the produced proof is small, and the algorithm is also fast to verify. In real applications, these proofs are produced off-chain, since they take quite some time, then they are sent to a on-chain smart contract (see Chapter 3 for more details) and, if they are correct, they are saved into a block. SNARKs are calculated on an arithmetic circuit because these circuits can be optimised to generate the proof faster. The formal definition is the following:

Definition 25. *A Succinct Non-interactive ARgument of Knowledge (SNARK) on a circuit C is a triple of algorithms* $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ *such that:*

- $\mathsf{Setup}(C)$: *the setup algorithm takes as input the circuit and outputs some public parameter* pp *for the prover and* vp *for the verifier;*
- $\mathsf{Prove}(\mathsf{pp}, x, w)$*:, the prove algorithm takes as input the public parameters of the prover* pp*, the public statement x, and the private witness w, and outputs a proof* π *that has to be "short". Usually, short means* $\mathsf{len}(\pi) = O_\lambda(log|C|)$*, where* $|C|$ *denotes the number of gates of the circuit, while* λ *is the security parameter;*
- $\mathsf{Verify}(\mathsf{vp}, x, \pi)$*: the verify algorithm takes as input the public parameters of the verifier* vp*, the public statement x, and the proof* π*, and outputs 1 if the proof is correct, 0 otherwise. This algorithm has to be "fast", which means* $\mathsf{time}(\mathsf{Verify}) = O_\lambda(|x|, log|C|)$*, where* $|C|$ *indicates the number of gates of the circuit and* $|x|$ *denotes the length of the public statement, while* λ *is the security parameter.*

This family of algorithms must satisfy, as usual, the properties of completeness, soundness and zero-knowledge. While the completeness property is the usual one, some slight modifications of the last two are reported, because they are usually the required properties.

- *Completeness*: for each x, w such that $C(x, w) = 0$,

$$\mathbb{P}[\mathsf{Verify}(\mathsf{vp}, x, \mathsf{Prove}(\mathsf{pp}, x, w)) = \text{accept}] = 1;$$

- *Adaptively Knowledge Soundness*: for every PPT adversary $\mathscr{A}$ such that

$$(\mathsf{pp},\mathsf{vp}) \leftarrow \mathsf{Setup}(C), \qquad \pi \leftarrow \mathscr{A}(\mathsf{pp},x),$$

$$\mathbb{P}[\mathsf{Verify}(\mathsf{vp},x,\pi) = \text{accept}] > \frac{1}{10^6},$$

  (so, it is non-negligible), there is a PPT extractor $\mathscr{E}$ that uses $\mathscr{A}$ such that

$$w \leftarrow \mathscr{E}(C,x), \qquad \mathbb{P}[\mathsf{Verify}(\mathsf{vp},x,\pi) = \text{accept}] > \frac{1}{10^6} - \varepsilon,$$

  where ε is a negligible value.

- *Statistical Zero-knowledge*: there exists a PPT algorithm $\mathscr{S}$, called simulator, such that the following two distributions are statistically close:

$$D_0 = \{\pi_0 \leftarrow \mathsf{Prove}(\mathsf{pp},x,w) \mid (\mathsf{pp},\mathsf{vp}) \leftarrow \mathsf{Setup}(C)\},$$

$$D_1 = \{\pi_1 \leftarrow \mathscr{S}(\mathsf{pp},\mathsf{vp},x) \mid (\mathsf{pp},\mathsf{vp}) \leftarrow \mathsf{Setup}(C)\}.$$

Most blockchain applications use the Groth16 protocol [66], because the verification time is constant and the proof size is the smallest between the family of SNARKs. However, it requires a trusted setup per circuit, meaning that the setup phase must be repeated every time a new circuit has to be compiled. There are also other possibilities, for example an algorithm that is showing promise and that it is already being used by blockchains such as ZCash [100] is Halo [25], which does not need a trusted setup and also allows for recursive proofs.

Chapter 3

Blockchain Basics

This chapter is devoted to the cryptographic aspects involved in blockchain technology. In particular, Section 3.1 explains in general the elements that constitute a blockchain, namely the structure of a block, some consensus protocols like Proof of Work, and what forks are. Then, sections 3.2, 3.3 and 3.4 describe more in detail three of the most important blockchains, namely Bitcoin, Ethereum and Monero. Regarding Ethereum, smart contracts and tokens are analysed in more detail because they will be relevant later. Finally, Section 3.5 shows how blockchain addresses are generated and encoded.

3.1 What is a blockchain?

To better understand what a blockchain is, this section introduces what the *Distributed Ledger Technology*, or DLT for short, is.

Definition 26 (Distributed Ledger)**.** *A* distributed ledger *is the consensus of replicated, shared and synchronized digital data that is distributed across many sites, countries, or institutions.*

Since a central authority is not required, DLT does not have a single point of failure. This is a key point: the register is public and shared between multiple users, called *nodes*, that have the opportunity to hold and update the data. Since data are also public, every user can verify the transactions that are saved into the register. DLT has some benefits compared to a traditional database server:

- it is more resistant to cyber attacks, since an attacker has to violate multiple nodes instead of just the central server. Moreover, data cannot be lost due to a malfunctioning of a node, since all other nodes would still be up and running;
- transparency of shared data means that they are immutable. In fact, if a user tries to modify them maliciously, he would be immediately noticed by the rest of the network that still holds the original data.

Definition 27 (Blockchain)**.** *A* blockchain *is a particular type of DLT, where data are organised in a growing list of ordering blocks.*

Blockchains were theorised long time ago, but they became reality at the end of 2008, when a mysterious figure (or group of people), known with the pseudonym of Satoshi Nakamoto, published the *Bitcoin white paper* [103]. His idea was the creation of an electronic payment system based on cryptographic protocols instead of trust, that allows two parties to transact with each other without the need of a third party. *Transactions* are digitally signed and then they are included into a *block*. Each transaction will cost a fee to the users that sent it, in order to have that transaction recorded on the blockchain. The higher the fee, the faster that transaction will be inserted into a block. Blocks can only be added in the chain and once they are inserted they cannot be removed or modified. The first block of the chain is known as *genesis block*, and to add every other block the network uses both cryptography and *consensus protocols*. Users who cooperate to support the blockchain are usually referred as *nodes*.

Blockchains can be *permissionless, hybrid or permissioned*. In a permissionless or public blockchain, everyone can become a user without the need to be authorized by a central authority, and can validate a block using the specific consensus protocol of that blockchain. Examples of permissionless blockchains are *Bitcoin* [103] and *Ethereum* [33]. Permissioned or private blockchains, instead, can only be accessed by specific users that have a permission. These users can only do specific actions that are granted to them by the central administration that runs the blockchain. Finally, hybrid blockchains try to use the best features of both permissioned and permissionless blockchains: they are private, but they still guarantee integrity, security and transparency.

3.1.1 Blocks

A block is the fundamental component of a blockchain. Its structure depends on the specific protocol implemented, but there are some properties that are valid in general. Each block is identified by a digest, called *hash block*. A block is split into two parts: the *header* and the *body*. The digest of the block n is saved into the header of the block $n+1$. This chain of hash is the one that gives the name to the entire data structure. The header also contains additional basic information about the block. The most important ones are:

- the *timestamp*, a parameter that indicates the data and the time in which a block is generated;
- the *block height*, that indicates the position of a block with respect to the genesis one, which has height 0 by default;
- the *Merkle root*, which is the root of the Merkle tree [98] obtained by using transaction hashes as leaves.

Instead, the body contains the list of transactions.

3.1.2 Consensus Protocols

Since there is no central authority to manage data entry, it is necessary to build a mechanism able to create a general agreement between the nodes of the network about the current and the future state of the chain. A consensus protocol prevents incorrect data entry, maintains the integrity of the transaction history and manages the block addition. Each node verifies the validity of the proposed transactions, which will be recorded on the blockchain if they are approved.
Consensus protocols must be resistant against *Sybil attacks*, that is the creation of several pseudo-identities to achieve majority. There are many different protocols of this kind, but the two most used are *Proof-of-Work* (PoW), based on computational power used to resolve difficult mathematical problems, and *Proof-of-Stake* (PoS), based on the amount of cryptocurrency that a user is disposed to place at stake. The nodes involved in the PoW protocol are called *miners*, while the ones involved in PoS are referred as *validators*.

Proof of Work.

Proof of Work (PoW) is a form of cryptographic proof in which one user demonstrates to the rest of the network that a large amount of computational work has been done. An essential condition is that the verification must be done with little computational effort.
The basic idea was proposed by Cynthia Dwork and Moni Naor in 1992 [52] to deter denial-of-service attacks and email spam, and consists of requiring the user of a service to perform a certain amount of computation, which is time-consuming to perform but quick to verify, before being able to access the service.
The term Proof of Work was formalised a few years later, in a 1999 paper written by Markus Jakobsson and Ari Juels [72], but the most successful version of proof of work is due to Adam Back, who proposed in 2002 the protocol called *Hashcash* in a technical report entitled *A Denial of Service Counter-Measure* [5].
Hashcash is very simple, powerful and flexible. The idea is to take any text, add a small piece of random text, the so-called *nonce*, and require the hash of the composed text to be below a certain target. If it is not below the target, one must try again with a new nonce, until a hash below the target is obtained. Since it is impossible to predict the output of the hash function as the input text changes, the protocol works like randomly extracting a number in a certain range and requiring it to fall below a certain value, which makes it easy to check the probability of success. Verifying that a large amount of calculation has actually been done is straightforward: just compute a single hash, using the nonce that is shown by the person who performed the PoW and verify that it is below the set target.

The main advantages of PoW are its security and the fact that it has been used since the beginning. However, this protocol presents two great flaws: a huge waste of energy and the danger of centralisation. In fact, miners who do not have a sufficient computational power gather into a single *mining pool* to combine all their assets to increase their probability of gaining the reward for mining a block. Proof of Work is becoming increasingly difficult to solve, so that fewer and fewer users have the necessary resources to participate.
Another potential issue is the 51% *attack*: if an attacker holds at least 51% of the total mining power of the network, it would be able to generate a fork (see Section 3.1.3) with a chain longer than the one created by the honest nodes, which hold less

than the 50% of the total power. Thus, the malicious chain would be longer than the honest one and would therefore be seen as the main chain.

Many blockchains like Bitcoin currently use PoW as their consensus mechanism. For a complete overview of PoW-based consensus protocols, see [97] or [115].

Proof of Stake.

Proof of Stake (PoS) is the second most utilised consensus protocol. While in PoW miners exploit energy to mine blocks, in PoS validators commit to a stake of cryptocurrency to insert a new block. Validators have two tasks: to propose and to attest blocks on a blockchain. In this context, staking cryptocurrency means freezing it, hoping to earn more by proposing a winning block.
There are several mechanisms that can be used to choose a validator for the next block:

- *randomness*: the probability of being chosen increases with the amount of currency staked;
- *seniority*: the relation between the coins owned and the time for which they were owned. Seniority is annulled once the validator adds a block;
- *velocity*: the rate of use of the currency.

Once chosen, the validator proposes a block that must be attested by a set of randomly chosen validators. If a sufficient number of members of this set of users attests to that block, it is added to the chain. Only after the block is attested and appended, the validator and the attesters earn their reward. Validators are punished if they do not complete the task they are assigned. For example, if a validator is offline when selected, it is penalized, since in order to grant an adequate level of scalability of the blockchain the majority of nodes must always be active.
For this reason, in a PoS based system the penalties for misbehaving are higher than with the PoW protocol: users that act maliciously are *slashed*, meaning that they are deprived of a part of what they staked.

In PoS based blockchains, every user that owns a computer and internet connection can be a validator. However, they have to stake a certain amount of currency, and for some people the amount could be too high. There are other ways one can get rewards from adding a block even without staking the whole sum: *staking pools*, that are essentially the same as mining pools. In this case, if one node of the pool is chosen to be a validator and its block is attested and added to the chain, all the users that staked a certain amount of currency in it get a reward proportional to how much they have staked.

One of the main problems of PoS is the *nothing-at-stake problem*: when a fork is generated (see Section 3.1.3) , a validator could try to work on both branches because he has stakes on both. In this way, this validator could try to trick the network and, for example, spend the same currency simultaneously in two different transactions.

3.1.3 Forks

Depending on the context, the term *fork* indicates different situations that can happen on a blockchain. In fact, *regular forks*, *soft forks* and *hard forks* can usually be distinguished .
A regular fork happens when two valid blocks are generated almost simultaneously: this phenomenon can happen because the network is peer-to-peer. In this case, each miner or validator will try to continue the chain that has as its last block the first received between the two involved in the fork. After a certain period of time, a new block will be attached to one of the two chains, thus becoming part of the main chain. The block on the other chain will be discarded, just as the transactions within it that will no longer be part of the blockchain's history. Discarded blocks are called *orphans*. For this reason, it is advisable to wait for some blocks to be inserted before considering a transaction as accepted.

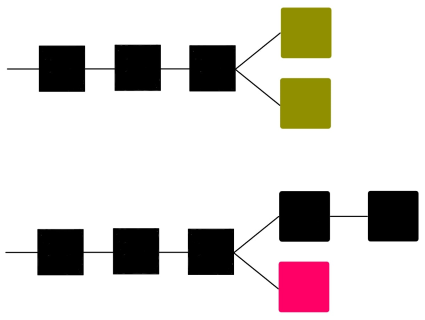


Fig. 3.1 Top: an example of a regular fork. Bottom: the fork is resolved in favor of the upper branch. The orphan block is shown in pink.

A soft fork indicates an update to the software that runs the blockchain and it is typically an optimization of the current protocol. A node may decide to not download the update and to continue to interact with the blockchain, without any issues.
In contrast, a hard fork is an update of the protocol that all the nodes are forced to install in order to keep interacting with the blockchain. In this case, two different scenarios can develop. In the first, all nodes adhere to the new protocol and the blockchain continues its growth undisturbed. In the second, the community splits in two over the proposal, as some of the nodes decide to break away from the main blockchain, creating a new one. The two chains share the same blockchain history until the hard fork occurs, and from then on they grow independently.

The next three sections describe more in detail three blockchains: Bitcoin [103], Ethereum [33] and Monero [2].

3.2 Bitcoin

Bitcoin has been the first active blockchain: its first block was inserted on January 3, 2009. It uses cryptography to secure and verify transactions as well as to control the creation of new units of a particular cryptocurrency, the *bitcoin* (BTC).

Bitcoin's cryptography is based on elliptic curves. The secret/public key pairs are generated on the short Weierstrass form curve

$$E : y^2 = x^3 + 7,$$

on the finite field $\mathbb{F}_q$ with $q = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$. This curve is known as *secp256k1*.
The number of points of the curve *secp256k1* over $\mathbb{F}_q$ is the prime number

$$\begin{aligned} n =& 1157920892373161954235709850086879078528375642790749043 \\ & 826051631415181614943337, \end{aligned}$$

so that the additive group $\mathscr{E} = E(\mathbb{F}_q)$ is isomorphic to $(\mathbb{F}_n, +)$. A generator of this group is the point $P = (P_x, P_y)$, where

$$\begin{aligned} P_x =& 55066263022277343669578718895168534326250603453777594 \\ & 175500187360389116729240, \\ P_y =& 32670510020758816978083085130507043184471273380659243 \\ & 275938904335757337482424. \end{aligned}$$

Bitcoin addresses are generated starting from a public key, thanks to the use of the hash functions *SHA256* [111] and *RIPEMD160* [49]. More details about the generation of a blockchain address are given in Section 3.5. Bitcoin supports the ECDSA signature, introduced in Section 2.3.1, since its inception. Since November 2021, after the *Taproot* [132] soft fork update, Bitcoin also supports the *Schnorr signature* [122] and its multisignature variants *MuSig* [95] and *MuSig2* [105].

The consensus protocol used by Bitcoin is the Proof of Work. In fact, to add a new block to the blockchain, miners must solve a complex mathematical problem. They have to find a special value, called *nonce*, such that the hash of the block header is smaller than a fixed value, called *target*. On average, a new block is inserted every 10 minutes. The miner who first solves the problem is rewarded with a certain number of bitcoin. This incentivizes miners to participate in the network and to secure the blockchain. Bitcoin has a finite coin supply, equal to around 21 million of coins. Every four years, through a process called *halving*, the reward for entering a new block is halved. The first block had a reward of 50 BTC and now, in 2023, the reward is 6.25 BTC. It is cointained in a special transaction, called *coinbase*.
Every 2016 blocks, about two weeks, the protocol checks how long it took to enter these blocks. If the time required was longer than the average time, which is set to 10 minutes, a parameter called *difficulty* grows and the target value becomes smaller. The opposite happens if the time required was less than the average. Nonce, target and difficulty are saved in the header of the block they refer to.
Miners also have a secondary way of getting profit, through the fees of the transactions that were inserted into the block.

The scripting language used by Bitcoin is very simple, in fact it is not *Turing complete*: in short, *for* loops cannot be used when writing a transaction or a script. The fact that only deterministic loops can be written is crucial, because the time of each individual transaction step can be estimated and, most importantly, the network knows in advance how much time it needs to check whether a transaction is correct or not. In this way, the network cannot be clogged by a malicious script that never finishes executing.

3.3 Ethereum

Ethereum is a decentralised, open-source blockchain platform that enables the creation of *smart contracts* (self-executing contracts with the terms of the agreement written directly into lines of code) and *decentralised applications* (DApps, applications that run on a decentralised peer-to-peer network, built on top of blockchain technology which make use of smart contracts to perform their functions). It was proposed in 2013 by Vitalik Buterin [33].
Like Bitcoin, Ethereum uses blocks to record transactions on its blockchain. The data in a block, in this case, includes also the current state of the *Ethereum Virtual Machine* (EVM). The EVM state refers to the current state of all the smart contracts and addresses on the Ethereum network at a specific point in time. It is updated every time a new block is added to the blockchain. This allows all nodes to have a consistent view of the network's current state, and ensures that all nodes can execute the same code and reach the same outcome. The cryptocurrency used on the Ethereum network is called *ether*.

Differently from Bitcoin, Ethereum adopts a *Turing-complete* scripting language to facilitate the writing of smart contracts. A consequence of this choice is the risk of suffering attacks due to the creation of endless loops that can congestion the entire blockchain. The elegant solution proposed by Buterin is the use of *gas*, that acts as a defence mechanism that prevents the execution of smart contracts that never halt. Every transaction has a fee, that must be paid in gas. Gas is obtained by converting ether. Every single step that this transaction must execute has its cost: for this reason, the attacker would have to possess an unlimited amount of ether to sustain the attack.

Ethereum generates the secret/public key pairs for its users using the same curve adopted by Bitcoin, secp256k1. However, the address generation is different, as in this case the hash function *Keccak256* [16] is utilized (more details can be found in Section 3.5). Ethereum uses two distinct methods to generate addresses, depending on whether it is a user's address or a smart contract address. They are usually called *externally owned accounts* and *contract accounts*. The digital signature used is ECDSA (see Section 2.3.1), but thanks to smart contracts other signatures with different properties can be used, if necessary.

The consensus protocol used by Ethereum is the Proof of Stake, namely the Gasper protocol [32]. The validator who inserts the next block is rewarded with a certain amount of ether.

3.3.1 Smart contracts.

A *smart contract* is a computer code that runs automatically on the top of a blockchain when certain conditions are met. This concept was introduced in the early 1990s by Nick Szabo [130]. Smart contracts allow to make agreements between two untrustworthy parties without the presence of a trusted third part. In fact, a smart contract eliminates the counterparty risk that occurs when a party tries to not comply with the conditions. The code is stored on multiple nodes of the blockchain ensuring immutability and transparency. Ethereum is the most important platform for developing smart contracts and the most used programming language is *Solidity* [129]. The activation of one of these takes place through the receipt of a transaction. The cost of a smart contract's activation depends on the number of computational steps.
Some of the most common uses are:

- automatic payment due to the achievement of specific results or due to triggering events;
- impose penalties, like in Proof of Stake protocols;
- creation of fungible *tokens*, handled by the ERC-20 standard;
- foundation and management of *Decentralised Autonomous Organisations* or *DAPPs*.

3.3.2 Tokens

A *token* is a digital representation of a value or a right. A high level classification divides them into three different classes: *utility tokens, security tokens* and *payment tokens*. Cryptocurrencies, such as BTC, are also tokens, precisely payment tokens.

Utilities and security tokens are instead generated by smart contracts, so they can be designed with even more complex features.
An utility token guarantees the right to access a service offered by the platform that generated it, or to confer a voting right to its holder. Another, very popular, application is the creation and trading of *Non Fungible Tokens* or NFTs. Ethereum enables the creation of them through the ERC-721 standard. Each NFT is unique and its value is defined exclusively by the request.
Security tokens guarantee the participation in the profit of the platform that issued them, through dividends.

3.3.3 Decentralised Autonomous Organisations

A *Decentralised Autonomous Organisation* or DAO is an organisation where activities and executive power are managed entirely by smart contracts. It is decentralised in terms of both the infrastructure and the governance.
A DAO relies on a blockchain that records transactions and the digital properties of its assets. This technology is chosen for its high difficulty of modifying data and its decentralisation features.
Decisions within the DAO are made collectively through a voting system. Each member has the opportunity to propose actions aimed at the management or development of the organisation. The right to vote on proposals is conferred to holders of DAO tokens: the greater the amount of token a user possesses, the higher is its voting influence.
A DAO is defined as autonomous since the executive power is entirely entrusted to smart contracts. The combination of these organisations can lead to the definition of a structure completely independent of the human factor. When a DAO generates profit, it is divided among its members in the form of cryptocurrency.

3.4 Monero

Monero is an open-source, privacy-focused cryptocurrency that was launched in April 2014. Unlike many other cryptocurrencies, Monero uses a public ledger to record transactions, but it is designed to provide a high level of anonymity for its users. Monero's blocks are structured differently than Bitcoin's, with a larger maximum block size and a dynamic block size that adjusts based on network conditions. Monero uses a proof-of-work consensus protocol, designed to be resistant to Application Specific Integrated Circuit mining, which is a type of hardware that is specifically designed to mine a particular cryptocurrency. This ensures that Monero remains decentralised, as individuals can mine the cryptocurrency using regular computer hardware.

One of the key features of Monero is its use of cryptography to protect the privacy of its users. It employs a different curve with respect to Bitcoin and Ethereum: in fact, the key pairs are generated on the twisted Edwards curve known as *edwards25519* , whose equation is

$$E : -x^2 + y^2 = 1 - \frac{121665}{121666} x^2 y^2.$$

The curve lies over the finite field $\mathbb{F}_q$ with $q = 2^{255} - 19$.
The secret key is given by the couple (k^v, k^s), with $k^s \leftarrow h(k^v)$ (h is a hash function); instead, the public key is the couple of points (K^v, K^s).
Monero uses several different cryptographic techniques to hide the identities of its users, including *stealth addresses* and *ring signatures*. A stealth address is a one-time, public address that is generated for each transaction starting from a public key of a user, while ring signatures are used to provide even greater privacy for its users. In the context of Monero, a ring signature is used to mix a user's transaction with a group of other transactions, making it difficult to determine which transaction belongs to the user.

Ring Confidential Transactions (RingCT) [106] and *Bulletproofs* [27] are further enhancements to Monero's privacy features. RingCT, which was introduced in January 2017, allows for the hiding of transaction amounts. This is achieved by using a combination of ring signatures and stealth addresses. The sender generates a one-time stealth address for the transaction, and the transaction amount is encoded using *Pedersen commitments* [110]. Bulletproofs is a non-interactive zero-knowledge proof protocol, which uses a specific type of zero-knowledge proof called a range proof; it can be used to prove that a value is within a certain range without revealing the exact value. Monero uses them since 2018.

3.5 Address generation

This section shows how addresses are generated, for some of the most important blockchains, starting from the choice of the private key. All these blockchains use elliptic curve cryptography, and, at least for the ones described, the security parameter level is 256 bits.

3.5.1 Private and public Key encoding

The private key is an arbitrary 256-bit integer k. Using k, it is possible to compute the point

$$K = kG = (K_x, K_y)$$

over the elliptic curve, from which the public key is derived. There are two different ways to represent the public keys:

i) $PK_1 = 0\text{x}04 \,||\, K_x \,||\, K_y$,

ii) $PK_2 = \begin{cases} 0\text{x}02 \,||\, K_x & \text{if } K_y \text{ is even,} \\ 0\text{x}03 \,||\, K_x & \text{if } K_y \text{ is odd,} \end{cases}$

where $||$ denotes the string concatenation. In the first case, the public key, also known as *uncompressed public key*, consists of 65 bytes (32 bytes for K_x and K_y, plus the byte 0x04), while in the second case the public key is called *compressed public key* and consists of 33 bytes. The two encoding are equivalent: the second one uses less memory space, but the second coordinated must be computed every time it is necessary to do so. The ways of obtaining the addresses for each cryptocurrency are discussed in the following paragraphs.

3.5.2 Bitcoin addresses

There are three manners to generate Bitcoin addresses starting from the point $K = (K_x, K_y)$.

Two hash functions appear in Bitcoin address generation, which are SHA-256 [111] and RIPEMD-160 [49]. The first step is the computation, for $i \in \{1,2\}$, of

$$W_i = \text{0x00}\,||\,\text{RIPEMD-160}(\text{SHA-256}(PK_i)),$$
$$checksum_i = (\text{SHA-256}(\text{SHA-256}(W_i)))[1..4],$$

where [1..4] denotes the first 4 bytes of that string. The first byte 0x00 is a prefix called *version byte*. Then, the first two ways to generate Bitcoin addresses are, for $i \in \{1,2\}$,

$$\text{Base58}(W_i\,||\,checksum_i),$$

where Base58 [9] is an encoding scheme.
The addresses of the third kind have been introduced in 2017 and they are called *segwit* addresses [123]. These addresses are computed only starting from compressed public keys. First of all, it is computed

$$W = \text{RIPEMD-160}(\text{SHA-256}(PK_2)),$$

then it is encoded using Bech32 [12] and it is concatenated to the prefix *bc1* in order to obtain the final address format

$$bc1\,||\,\text{Bech32}(W).$$

3.5.3 Ethereum addresses

Ethereum addresses for externally owned accounts are generated starting from the uncompressed public key, that is

$$PK = \text{0x04}||K_x||K_y.$$

However, a different hash function is used: KECCAK-256 [16]. The address is computed as

$$\text{KECCAK-256}(PK)[1..20],$$

where [1..20] denotes the first 20 bytes of that string.
After this computation, the addresses are encoded following the rules described in the EIP-55 document [53]. In short, the capitalisation of certain alphabetic characters in

the address is changed to obtain a checksum that can be used to protect the integrity of the address from typing or reading errors.

3.5.4 Dogecoin addresses

Dogecoin address generation is similar to the first two methods described for Bitcoin, thus it uses both uncompressed and compressed public keys. It only changes the *version byte* 0x00 of Bitcoin into 0x1E. In this way, the final Base58 encoding gives a D as the first letter for each address. Dogecoin does not generate addresses following the *segwit* standard.

3.5.5 Litecoin addresses

Litecoin can generate addresses using all the three methods described for Bitcoin. In the first two cases, the prefix is the *version byte* 0x30 instead of 0x00. In this way, all the addresses start with the letter *L*. In the segwit case, the Bech32 encoding of the address is concatenated with the prefix $ltc1$, that is

$$ltc1\,||\,\text{Bech32}(\text{RIPEMD-160}(\text{SHA-256}(PK_2))).$$

3.5.6 Dash addresses

The address generation is similar to Bitcoin, but in this case the *version byte* is changed into 0x4c, so that all the addresses start with the letter X. Dash does not generate addresses following the segwit standard.

3.5.7 Zcash addresses

The address generation is similar to Bitcoin, but in this case the *version byte* is changed into [0x1c, 0xb8]. Zcash addresses can either start with the letter t, if they are transparent, or z, if they are shielded. Zcash does not generate addresses following the segwit standard.

3.5.8 Bitcoin Cash addresses

Bitcoin Cash (BCH) is the result of a Bitcoin hard fork that happened in 2017, when some of Bitcoin nodes did not share the segwit soft fork. BCH addresses are encoded two times:

- first, an address with the same encoding of Bitcoin is obtained,
- finally, the address is encoded again, with an encoding scheme called *CashAddr* [11], which is used by Bitcoin Cash only. This encoding is similar to Bech32.

BCH addresses that are encoded twice start with the string *bitcoincash:q*, or just with the letter *q*. It comes natural that Bitcoin Cash does not generate addresses following the segwit standard.

Chapter 4

Additional tools

This chapter introduces some additional mathematical concepts, along with some specific protocols and applications of the blockchain that will be used in later chapters of this work. In more detail, Section 4.1 describes tensors, alternate trilinear forms, and cryptographic assumptions, that will be used in Chapter 5. This section also contains some information about the most efficient attacks known against these cryptographic problems. Next, Section 4.2.1 introduces Semaphore and MACI (Minimal Anti-Collusion Infrastructure), two protocols designed to be generic privacy layers for Ethereum DApps, while Section 4.3 briefly explains quadratic voting. Finally, Section 4.4 summarises the properties of soulbound tokens, which will be used, along with the two protocols mentioned above and the quadratic voting mechanism, in Chapter 7.

4.1 Tensors and Alternating Trilinear Forms

Notation. For a prime power q, $\mathbb{F}_q$ is the finite field with q elements, and $\mathbb{F}_q^n$ is the n-dimensional vector space over $\mathbb{F}_q$. $\mathbb{F}_q^{n\times m}$ denotes the linear space of $n \times m$ matrices with coefficients in $\mathbb{F}_q$. Let $\mathrm{GL}_n(q)$ be the group of invertible $n \times n$ matrices with coefficients in $\mathbb{F}_q$.

This section follows Section 2.1 of D'Alconzo [42].
Given a positive integer d, a d-tensor over $\mathbb{F}_q$ is an element of the tensor space $\mathscr{T} = \otimes_{i=1}^{d} \mathbb{F}_q^{n_i}$. If a base $\{e_1^{(i)}, \dots, e_{n_i}^{(i)}\}$ for every linear space $\mathbb{F}_q^{n_i}$ is fixed, a d-tensor

T can be represented with respect to its coefficients $T(i_1,\dots,i_d)$, that is

$$T = \sum_{i_1,\dots,i_d} T(i_1,\dots,i_d) e^{(1)}_{i_1} \otimes \dots \otimes e^{(d)}_{i_d}.$$

T has size $n_1 \times \dots \times n_d$.
Then, a group action $\star$ between the Cartesian group of invertible matrices $G = \mathrm{GL}_{n_1}(q) \times \dots \times \mathrm{GL}_{n_d}(q)$ and the tensor space $\mathscr{T}$ can be defined in the following way:

$$\star : G \times \mathscr{T} \to \mathscr{T}$$

$$\left((A_1,\dots,A_d), \sum_{i_1,\dots,i_d} T(i_1,\dots,i_d) e^{(1)}_{i_1} \otimes \dots \otimes e^{(d)}_{i_d}\right)$$

$$\sum_{i_1,\dots,i_d} T(i_1,\dots,i_d) A_1 e^{(1)}_{i_1} \otimes \dots \otimes A_d e^{(d)}_{i_d}.$$

The *tensor isomorphism problem* has recently been of interest, given its properties and links with computational complexity theory. The following definition recaps the problem.

Definition 28. *The d-Tensor Isomorphism (d-TI) problem takes as input two d-tensors $T_1, T_2 \in \mathscr{T}$, and it outputs YES if there exists an element $g \in G$ such that $T_2 = g \star T_1$, NO otherwise.*
The search version is the problem of finding such an element g.

It has been proved by Grochow and Qiao [63] that the d-TI problem can be polynomially reduced to 3-TI. For this reason, cryptographic applications focus on 3-tensors. TI is the class that contains decision problems that can be polynomially reduced to d-TI for a certain d. Furthermore, a problem is TI-complete if it is in TI and if the d-Tensor Isomorphism problem, for any d, reduces to this problem [64].

Now, alternating trilinear forms are defined.

Definition 29 (Alternating trilinear forms)**.** *Given positive integers k, n, m and a prime power q, a map*

$$\phi : \underbrace{(\mathbb{F}_q)^n \times \dots \times (\mathbb{F}_q)^n}_{k \text{ times}} \to (\mathbb{F}_q)^m$$

can be

1. alternating*: if ϕ is equal to the zero vector whenever two of its arguments are equal;*

2. k-linear*: if ϕ is linear in each of its k arguments.*

If $m = 1$, i.e. the codomain of ϕ is the field $\mathbb{F}_q$, ϕ is said to be a form.

An alternating trilinear form *is a map*

$$\phi : \left(\mathbb{F}_q\right)^n \times \left(\mathbb{F}_q\right)^n \times \left(\mathbb{F}_q\right)^n \to \mathbb{F}_q$$

that is alternating and trilinear. The set of alternating trilinear forms over $\left(\mathbb{F}_q\right)^n$ is denoted with $\mathrm{ATF}(q,n)$*.*

Observe that an alternating trilinear form ϕ corresponds to a 3-way array $(a_{i,j,k})_{i,j,k} \in \mathbb{F}_q^n \otimes \mathbb{F}_q^n \otimes \mathbb{F}_q^n$, where $a_{i,j,k} = \phi(e_i, e_j, e_k)$ [65].
It is known that $\mathrm{ATF}(q,n)$ is a linear space over $\mathbb{F}_q$ of dimension $\binom{n}{3}$. This implies that any alternating trilinear form can be represented and stored with $\binom{n}{3}\lceil\log_2 q\rceil$ bits.

Starting from $\mathrm{GL}_n(q)$, a group action over $\mathrm{ATF}(q,n)$ can be defined, similar to the one described before over $\mathscr{T}$.

Definition 30. *The group action $(\mathrm{GL}_n(q), \mathrm{ATF}(q,n), \star)$ is defined by*

$$\begin{aligned} \star : \mathrm{GL}_n(q) \times \mathrm{ATF}(q,n) &\to \mathrm{ATF}(q,n) \\ (A, \phi) &\mapsto \phi \circ A^t. \end{aligned} \tag{4.1}$$

In other words, the alternating trilinear form $(A \star \phi)(x,y,z)$ is the map

$$\phi(A^t x, A^t y, A^t z).$$

The group action above defines an equivalence, that is ϕ and $A \star \phi$ are said to be equivalent. Given two alternating trilinear forms, the problem of deciding if there is an equivalence, and the problem of finding a matrix that sends one into the other can be defined. This fact is formalised in the following definition.

Definition 31. *The* Alternating Trilinear Form Equivalence *(*ATFE*) problem is given by:*

- Input*: ϕ, ψ in* $\mathrm{ATF}(q,n)$.
- Output*: YES if there exists A in* $\mathrm{GL}_n(q)$ *such that* $\phi = A \star \psi$*, and NO otherwise.*

The search Alternating Trilinear Form Equivalence *(*sATFE*) problem is given by*

- Input*: ϕ, ψ in* $\mathrm{ATF}(q,n)$ *such that they are equivalent.*
- Output*: A in* $\mathrm{GL}_n(q)$ *such that* $\phi = A \star \psi$*.*

The assumption that the sATFE problem is intractable comes from the fact that its decisional counterpart ATFE is TI-complete. The ATFE problem is polynomially equivalent to d-Tensor Isomorphism for $d \geq 3$, hence by definition it is TI-complete. This implies that ATFE is as hard as all TI-complete problems from [64]. At the moment, no polynomial-time algorithm solving any TI-complete problem is known.

4.1.1 Known attacks to the sATFE problem

There are several algorithms that can be used to analyse sATFE. When designing a cryptographic protocol, the choice of parameters must keep track of these known attacks in order to obtain a plausibly secure algorithm. The following list enumerates the possible attacks:

- *Brute force*: it consists in trying all the matrices A, in order to verify if there is one that satisfies the relation $\phi = A \star \psi$. The complexity of this attack is around $0(q^{n^2})$;
- *Average-case algorithms*: these kind of attacks were analysed in [65] and their complexity is around $q^{O(n)}$, which is still not practical;
- *heuristic algorithms*: these algorithms are based on multivariate cryptography and computational group theory (Gröbner basis), and they have complexity $O(q^{2/3n} \cdot n^{2\omega} \cdot \log_2(q))$ for some constant ω. These attacks are more practical and force the value n to be at least equal to 9 for practical implementations;
- *graph-theoretic algorithms*: these algorithms are the most dangerous ones, since they can compute the matrix A in time $O(q)$ when $n = 9$, and in time $O(q^4)$ when $n = 11$. The attacks perform better when n is odd, while for n

even there is a (small) fraction of keys that can be broken in constant time, while all the others can still be considered secure.

- *other algorithms*: new algebraic attacks that appear to be promising were presented during CBCrypto2023 [116].

Keeping in mind these attacks, some values of n that appear safe are $n = 10$ or $n \geq 12$. The fraction of broken keys should be discarded. This is easily achieved by choosing as the origin an alternating trilinear form not in this set.

4.2 Applications of zk-SNARKs to the blockchain environment

zk-SNARKs are currently used as building blocks in privacy layers built on top of some blockchains. For example, the Ethererum blockchain alone is developing more than fifteen protocols that use SNARKs as the main cryptographic technique [114]. Here, Semaphore and MACI, two of these protocols, are described more in detail.

4.2.1 Semaphore

Semaphore [124] is a zero-knowledge protocol which allows Ethereum users to prove their membership in a group and to send signals, such as votes or endorsements, without revealing their identity. More in detail, Semaphore provides three functionalities:

- *Creation of private identities*. Each user receives a secret/public pair $(\mathsf{sk}, \mathsf{pk})$. More precisely, the secret key is a tuple $\mathsf{sk} = (\mathsf{IdTrapdoor}, \mathsf{IdNullifier}, \mathsf{IdSecret})$, where $\mathsf{IdTrapdoor}$ and $\mathsf{IdNullifier}$ are generated randomly and $\mathsf{IdSecret} = H(\mathsf{IdTrapdoor} \,||\, \mathsf{IdNullifier})$, where H is a hash function. The nullifier is needed to avoid users signaling more than once. The public key is instead the hash of the secret: $\mathsf{pk} = H(\mathsf{IdSecret})$. sk is used to create zero-knowledge proofs;
- *Insertion of an identity into a group*. A Semaphore group is an anonymity set which users can join without losing the control over their identity. To

be part of the same group, all the users involved must have some element in common, and everyone is sure that all the members possess this element without knowing their real identity. When a user joins a group, his or her public key pk becomes one leaf of the Merkle tree of that group;

- *Sending of anonymous signals.* Signals are signed messages which are broadcast on-chain. The signal contains some data such as a vote, the proof that the user is part of a Semaphore group, and the proof that the same user created both the signal and the first proof. Each signal also contains two additional public nullifiers: $\mathsf{ExtNullifier}$, which is some input generated when the user wants to send a signal, and $\mathsf{Nullifier} = H(\mathsf{IdNullifier}\,||\,\mathsf{ExtNullifier})$. To recap,

  $$\mathsf{Signal} = (\mathsf{Data}, \mathsf{Proof}(\mathsf{pk}), \mathsf{Proof}(\mathsf{Data}, \mathsf{Proof}(\mathsf{pk})), \mathsf{ExtNullifier}, \mathsf{Nullifier}).$$

  If two different signals have the same $\mathsf{Nullifier}$, it means that the same user has signaled more than once.

Semaphore can thus be regarded as a Sybil-resistance mechanism: each sent signal contains certain zero-knowledge proofs, generated off-chain and validated on-chain, about the sender's membership of a certain group, as well as the validity of the signal itself. More details about the implementation of the Semaphore circuits or their smart contracts are available on the Semaphore website [124].
Some protocols, like *UniRep* [134], are already using Semaphore. UniRep is a private and non-repudiable reputation protocol, where *users* can receive positive and negative reputation from *attesters*, and can voluntarily prove that they have at least a certain amount of reputation without revealing the exact amount. Moreover, users cannot refuse to receive reputation from an attester.

4.2.2 MACI

MACI [94] stands for *Minimal Anti-Collusion Infrastructure* and it is a protocol that allows users to vote on-chain with a greatly increased collusion resistance. It was proposed by Buterin in 2019 [28]. All transactions on a blockchain are public, therefore a voter can easily show to a briber which option they voted for. MACI counters this by using, again, zk-SNARKs to hide how each user voted, while still

allowing to know the final vote result. In this way, a user cannot show to a briber which option they voted for anymore.
MACI has two different actors: *users*, the people that send a vote through a smart contract, and a single *trusted coordinator*, which makes the tally of the votes and releases the final result. The coordinator uses zk-SNARKs to prove that the tally is valid without revealing the vote of every user.
Before voting, each user, who must already possess a secret/public key pair $(\mathsf{sk}, \mathsf{pk})$, e.g. generated when joining a Semaphore group, must register their public key pk in a smart contract SC_1. All the public keys, together with a timestamp and the number of voice credits, are saved into a Merkle tree $\mathsf{MerkleTree}_{\mathrm{SignUp}}$. Voice credits can be spent by the registered users to vote. They can do it with any address, but the transaction must contain an information about the registered public key. Each user I shares also a key $\mathsf{SharedKey}_{I,C}$ with the trusted coordinator C, which is used to encrypt and decrypt transactions. In fact, in order to vote, the user I will send the following encrypted transaction to a poll smart contract SC_2:

$$\mathsf{Transaction} = \mathsf{Enc}_{\mathsf{SharedKey}_{I,C}}(\mathsf{Sig}, \mathsf{Command}),$$

$$\mathsf{Command} = (\mathsf{pk}_I, \mathsf{Vote}_{\mathrm{option}}, \mathsf{Vote}_{\mathrm{amount}}).$$

Sig represents the signature of the user that is sending the transaction, while $\mathsf{Vote}_{\mathrm{option}}$ is the list of projects that the user wants to vote for. Finally, $\mathsf{Vote}_{\mathrm{amount}}$ is the list containing the amount of voice credits that the user has allocated to each project they have decided to support.
A user can override their previous vote if they sign a new transaction with its secret key sk_I. In this case, the coordinator will consider only the last message as valid.
A user can also override their public key if they sign a new transaction that contains a different public key $\widetilde{\mathsf{pk}}$ using its secret key sk_I, that is

$$\mathsf{Transaction} = \mathsf{Enc}_{\mathsf{SharedKey}_{I,C}}(\mathsf{Sig}, \mathsf{Command}),$$

$$\mathsf{Command} = (\widetilde{\mathsf{pk}_I}, \mathsf{Vote}_{\mathrm{option}}, \mathsf{Vote}_{\mathrm{amount}}).$$

From then on, only messages containing the public key $\widetilde{pk_I}$ will be considered valid. This feature is known as *public key switching*. When this occurs, only transactions signed with the secret key $\widetilde{sk_I}$ will be considered valid. Public key switching can be used to avoid bribes, since no one except the user and the trusted coordinator knows

whether or not the transaction sent will be considered valid after decryption.
After the voting period, the coordinator will use a third smart contract SC_3 to build a state tree that keeps track of all the valid votes. Then, she does the tally of the votes and publishes the results. Hence, the coordinator must create two different zk-SNARKs proofs:

- the first proof is published to prove that the state tree contains only the valid messages, without revealing all the messages;
- the second proof is created to show that the tally of the votes was done using only valid messages, and that their individual contribution leads to the final result.

Both proofs are verified by another smart contract SC_4, specifically built to read MACI proofs: if both the verifications are correct, the process was executed correctly.
MACI has been already successfully used by some projects like clr.fund [40] to protect the fairness of a voting process for cases in which the funding amounts become very large.

4.3 Quadratic voting

Quadratic voting [85] is a method of collective decision-making in which participants do not just vote for or against an issue, but they also express their opinion on it. This is an alternative to two other more classic voting modes: *One-Dollar-One-Vote*, where each user can vote as many times as they want and each vote costs one dollar, and *One-Person-One-Vote*, where each user can vote exactly once. The first case is used when a certain threshold must be reached to obtain a specific type of service, while the second case is used, for example, for electoral votes.
These two methods have their drawbacks: in the first case, only people who really care will vote, financing the project with a large amount of money that will also cover all the other users who are interested in it, but not enough to pay for that service. In the second case, on the other hand, every user has equal voting power, hence there is no way to express how much they care about what they are voting for.
Quadratic voting is a novel method that solves both these problems in a mathematical way: in short, each person can vote all the times they want, but the n-th vote will

cost n. In this way, if a user votes n times, the total cost of these votes will be around $\frac{n^2}{2}$ (hence, the name quadratic).
The proposed idea can also be expanded in the so-called *quadratic funding* [31]. Suppose there is a list of public goods $\{P_1, \dots P_M\}$, and suppose that N users have to vote (or bid) on these projects to show their appreciation towards them. The aim is to reward more the projects that were voted by more people, rather than rewarding that project which was chosen by fewer people, but with more funds available. This can be achieved by matching donations for these projects proportionally to the square of the sum of the square roots of the individual donations, that is, if the user n places a bid $D_{m,n}$ for the project P_m, then each project will receive

$$\text{Fund}_m \sim (\sqrt{D_{m,1}} + \dots + \sqrt{D_{m,N}})^2.$$

To clarify this concept, a simple example is considered. Suppose there are $M = 3$ projects, $N = 4$ users and a total of 1400 dollars that must be distributed between these projects using the quadratic funding mechanism. The first project is funded by one user, who gives 100 dollars; the second project is funded by two users, each of whom gives 50 dollars, while the last project is funded by all four users, each of whom gives 25 dollars. Mathematically, the following results are obtained:

$$\text{Fund}_1 \sim (\sqrt{100})^2 = 100,$$

$$\text{Fund}_2 \sim (\sqrt{50} + \sqrt{50})^2 = 200,$$

$$\text{Fund}_3 \sim (\sqrt{25} + \sqrt{25} + \sqrt{25} + \sqrt{25})^2 = 400.$$

In conclusion, the first project will receive 200 dollars, the second 400 dollars and the third 800 dollars. Therefore, even though each project received the same appreciation in terms of donations - each received exactly 100 dollars - the project that was voted by the most of the users will receive more money.
However, since quadratic voting (and funding) incentives the cooperation, the idea is vulnerable to groups that are already cooperative, or to collusion attacks. These problems can (partially) be solved, for example using MACI, or by assigning a lower weight in the quadratic mechanism formula to those users for whom it is assumed that they will have the same view of the facts.
Furthermore, in certain situations it may be interesting to use *negative quadratic voting* [30], i.e. a user's vote towards a project is not intended to contribute towards

that project, but to penalize it in relation to all others.
Quadratic voting has already been used in practice, both in centralised and decentralised contexts: notable examples are the *Taiwanese presidential hackathon* or the *Gitcoin grants* [29], which use quadratic funding to allocate funds to projects in the Ethereum ecosystem.

4.4 Soulbound tokens

Soulbound tokens (SBTs) have been theorised by Weyl, Ohlhaver and Buterin in 2022 [138]. They are non-transferable (but possibly revocable), non fungible and publicly visible tokens, that encode subjective qualities like the reputation of a user or the authenticity of a piece of art. SBTs are held into wallets, usually called *souls* in this contest. They are public by default but, given an application, the “right" degree of privacy can be achieved with a mix of cryptographic protocols like zero-knowledge proofs.
Soulbound tokens can be attested by other souls, representing individuals, companies or institutions. A user may thus possess a digital identity linked to the real one, which is represented by a list of SBTs, each of which provides different information about that person. In addition, a soulbound token can also be generated by a dApp when a particular situation occurs.
Since these tokens are non-transferable, when a user does not follow the rules of a certain protocol they might receive one, thus showing to the rest of the network their negative behavior. This type of token is therefore a useful tool to introduce a social incentive/disincentive into the blockchain world.

Part II
What more do we know now?

After introducing the necessary theoretical concepts, the second part of this thesis is devoted to the original results. Chapter 5 introduces a new ring signature, while Chapter 6 presents a generalisation of the Proof-of-Work. Chapter 7 describes an idea for a new dispute resolution protocol that preserves the privacy of the users involved, and finally Chapter 8 extends an already known attack to some specific blockchain addresses.

Chapter 5

TRIFORS: LINKable Trilinear Forms Ring Signature

The result presented in this chapter is a joint work with Giuseppe D'Alconzo, and it can be found in [43].

With recent developments that are bringing us closer to the advent of quantum computing, many cryptographic algorithms can no longer be considered secure. For example, all the signatures described in Chapter 2 are based on cryptographic assumptions that are broken by the well known Shor's algorithm [126].
In the last years, NIST launched a Post-Quantum Standardisation Contest, now come to an end, that aimed at finding protocols based on cryptographic assumptions that appear to be resistant even in case the quantum computer arrives. The most promising ones are based on lattice-based cryptography, multivariate cryptography, hash-based cryptography, isogeny-based cryptography and code-based cryptography. Other interesting post-quantum assumptions derive from Cryptographic Group Actions, introduced by Alamati, De Feo, Montgomery and Patranabis [1] in 2020. Group actions are an interesting tool to generalise some well-known assumptions like the Discrete Logarithm Problem. The most promising and studied post-quantum cryptographic group action is CSIDH [34], and the topic has received in general a lot of interest, both in theoretical and applied fashion [64, 7, 131, 73, 18, 19, 45].

A topic of interest in the recent years is ring signatures based on post-quantum assumptions. In fact, starting from 2019 there have been many proposals. That year, Esgin, Zhao, Steinfeld, Liu and Liu proposed *MatRiCT* [55], while Lu, Au and Zhang described *Raptor* [92]. Both signatures are based on lattices assumptions and, more precisely the former is based on Module Shortest Integer Solution (MSIS) and Module Learning With Errors (MLWE), while the latter is based on the NTRU assumption. Shortly after, Beullens, Katsumata and Pintore proposed two different ring signatures, known as *Calamari* and *Falafl* [19]: Calamari is up to date the only ring signature based on isogenies, more precisely on the CSIDH assumption [34], while Falafl is again based on the MSIS and MLWE assumptions. In 2021 two non-linkable schemes were proposed, both based on MSIS and MLWE. The first one, by Lyubashevsky, Nguyen and Seiler, is called *SMILE* [93] and it is based on set membership proofs. The second one, *DualRing* [141], uses an innovative construction based on two rings. In the same year, Esgin, Steinfeld and Zhao presented a follow-up work optimizing *MatRiCT* [56]. By following the same line of research of [19], Barenghi, Biasse, Ngo, Persichetti and Santini proposed the (linkable) ring version [8] of *LESS* [7], a signature whose security assumption is based on the code equivalence problem. Recently, Bellini, Esser, Sanna and Verbel proposed *MR-DSS* [13], a non-linkable ring signature based on the MinRank problem. Finally, in a work simultaneous to the one described in this chapter [38], Chen, Duong, Nguyen, Qiao, Susilo and Tang, besides analysing a particular class of signatures in the Quantum Random Oracle Model, use the construction in [19] to obtain a ring signature from alternating trilinear forms.

The original result. This chapter presents TRIFORS, a logarithmic post-quantum (linkable) ring signature based on a novel assumption regarding equivalence of alternating trilinear forms. This work is built starting from the digital signature presented by Tang et al. [131] at EUROCRYPT 22, and from the framework introduced by Beullens, Katsumata and Pintore [19] to obtain a linkable ring signature from a group action.
The signature is based on a novel cryptographic assumption, stating that the *search Alternating Trilinear Form Equivalence* (sATFE) problem is intractable. A base OR Sigma protocol, having soundness error of $1/2$, is designed. The term "OR" refers to the fact that the prover knows at least a secret key for the public keys in the ring. Given a security parameter λ, the soundness error is decreased to $1/2^{\lambda}$, running the protocol in parallel and using some optimisations. In particular, a fixed-weight challenge is used and, via a well-known combinatorial technique, it is compressed in a string of length $\sim \lambda$ bits. Later, the protocol is modified in the so-called "main

OR Sigma protocol” in two versions, with and without tag. Then, applying the Fiat-Shamir transform [58], the two sigma protocols are transformed into two digital signatures: a ring signature from the OR sigma protocol without tag, called TRIFORS, and a linkable ring signature Link-TRIFORS from the version with tag.
The described post-quantum ring signature has logarithmic length in the size of the ring and competes with the state-of-the-art, as shown in Table 5.1, for the non-linkable version: the only scheme having signatures significantly shorter (for small rings) is *Calamari* [19], but it pays the heavy computation induced by isogenies. Recent proposals based on lattice assumptions [56, 141] achieve very short signatures for small rings; however, they are comparable to this scheme for medium-size rings. Moreover, TRIFORS performs even slightly better than the novel ring signature on codes [8]. It is worth noting that the signature size grows more slowly than any other ring signature listed in the table: for huge rings, TRIFORS is the signature with the smallest size. Finally, Table 5.2 reports the length of a public key for all the ring signatures just described. It can be seen that a public key in TRIFORS is smaller than in most competitors, implying that the length of the ring signature-public key pair is competitive even for rings of lower cardinality. For practical purposes, it is useful to compare $R|\mathsf{pk}| + |\sigma_R|$, where R is the cardinality of the ring, $|\mathsf{pk}|$ is the size of the public key, and $|\sigma_R|$ is the size of the signature obtained using R public keys. Indeed, to check a signature, the verifier needs all the public keys in the ring.

Scheme	*Assumption*	*R*							
		2^1	2^3	2^5	2^6	2^{10}	2^{12}	2^{15}	2^{21}
TRIFORS	sATFE	**9.1**	**10.6**	**12.0**	**12.8**	**15.7**	**17.2**	**19.4**	**23.8**
Calamari [19]	CSIDH-512	3.5	5.4	-	8.2	-	14	-	23
Falafl [19]	MSIS, MLWE	29	30	-	32	-	35	-	39
MatRiCT+ [56]	MSIS, MLWE	5.4	8.2	11	12.4	18	20.8	25	33.4
DualRing-LB [141]	MSIS, MLWE	4.5	4.6	-	6	-	55	-	-
SMILE [93]	MSIS, MLWE	-	-	16	-	17.3	-	18.7	-
Ring LESS [8]	Perm. Code Eq.	-	10.8	-	13.7	-	19.7	-	28.6
MR-DSS [13]	MinRank	-	27	32	36	145	422	-	-

Table 5.1 Size in KB of the signatures, where the security parameter λ is 128 and R is the size of the ring.

Notation. In the following, let $\mathbb{N}^+ = \{1,2,\ldots\}$ and $\mathbb{R}$ be the sets of positive natural and real numbers, respectively. Denote with λ the security parameter, and

Scheme	*Assumption*	pk
TRIFORS	sATFE	**0.3**
Calamari [19]	CSIDH-512	0.1
Falafl [19]	MSIS, MLWE	2.2
MatRiCT+ [56]	MSIS, MLWE	3.4
DualRing-LB [141]	MSIS, MLWE	$2.8 \sim 3.4$
SMILE [93]	MSIS, MLWE	3.3
Ring LESS [8]	Perm. Code Eq.	11.6
MRr-DSS [13]	MinRank	R

Table 5.2 Size in KB of the public keys, where the security parameter λ is 128 and R is the size of the ring.

with $\mathsf{poly}(\cdot)$ a function that is polynomial in its argument, i.e. there exists a positive integer m such that $\mathsf{poly}(x) = O(x^m)$. A function $\varepsilon : \mathbb{N}^+ \to \mathbb{R}$ is *negligible* if there exists n_0 such that, for every $n > n_0$, $\varepsilon(n) \leq 1/p(n)$ for every polynomial p. A function not having this property is called *non-negligible*. The probability of an event is *overwhelming* if it is equal to $1-\varepsilon$, where ε is a negligible function. An adversary has an *advantage* $\mathfrak{a}$ when playing a game against a challenger, if its probability of winning that game is $\frac{1}{2}+\mathfrak{a}$. For a prime power q, $\mathbb{F}_q$ is the finite field with q elements, and $\left(\mathbb{F}_q\right)^n$ is the n-dimensional vector space over $\mathbb{F}_q$. The group of invertible $n \times n$ matrices with coefficients in $\mathbb{F}_q$ is denoted with $\mathrm{GL}_n(q)$. The *Hamming weight* of a vector x is the number of its non-zero coordinates, and it is denoted with $\mathrm{w}(x)$. Denote with $||$ the concatenation of strings or vectors. Given a binary string x, $|x|$ denotes the bit length of x.
RO denotes the Random Oracle, which is augmented with some functionalities:

- a commitment functionality $\mathsf{RO}_{\mathrm{Com}}(x,r)$, where x is the committed value and r a random string;
- a seed expansion functionality $\mathsf{RO}_E(\mathsf{seed})$, where the output is defined by the context;
- a collision-free hash function RO_H, with output length 2λ.

In the pseudocode, "$\leftarrow\!\$$" denotes the random sampling, "$\leftarrow$" is a variable assignment, and "$=$" is the equality check.

5.1 Index-hiding Merkle trees and seed trees

This section, following the notation used in [19], briefly recaps the Merkle tree algorithms and introduces the seed trees that will be used later to optimize the described Sigma protocol.

Index-hiding Merkle trees. A Merkle tree can be described by the following algorithms:

- $\mathsf{MerkleTree}$: the input of this algorithm is the list of M elements $A = \{a_1, \dots, a_M\}$, that represents the leaves of the tree. The algorithm computes the nodes of the whole binary tree up to its root. To obtain a *index-hiding Merkle tree*, just concatenate the two elements following the lexicographical order, that is $b = hash(b_{\text{left}}||b_{\text{right}})_{\text{lex}}$.
  The two outputs of the algorithm are its root root, which represents the cryptographic commitment, together with a representation of the whole tree, tree.
- $\mathsf{getMerklePath}$: the two inputs of this algorithm are the Merkle tree tree and a certain index $i \in \{1, \dots, M\}$. The output will be a list path, that contains an information about the sibling of a_i (i.e. the node with the same parent of a_i), together with all the siblings of any ancestor of a_i, ordered by decreasing height.
- $\mathsf{ReconstructRoot}$: the inputs of this algorithm are the list of M elements $A = \{a_1, \dots, a_M\}$ and the path path, which is the output of the previous algorithm $\mathsf{getMerklePath}$. The output is the reconstructed root root of the Merkle tree.

Seed trees A *seed tree* is yet again a complete balanced binary tree, but its construction is different with respect to the one of the Merkle tree given is Section 2.5.2. In this case, the tree is built starting from its root as follows: given a node T represented by a binary string, its two children T_1, T_2 are given by $T_i = \mathsf{RO}_E(T, i)$ for $i = 1, 2$, obtained via a 2λ hash evaluated in the value of T. Each binary string has finally length equal to λ. In this way, to compute the leaves of any subtree with root T, it is sufficient to know T.
Seed trees have been used recently as a clever optimisation to decrease the length of

the signature [8, 19, 13]. Here, the algorithms that will be used later to optimise the Sigma protocol are introduced:

- $\mathsf{SeedTree}$: the inputs of this algorithm are a binary string $\mathsf{seed}_{\mathsf{root}}$ of length λ, which represents the root of the tree, and an integer M, the number of leaves of the tree. The algorithm computes the complete balanced binary tree with M leaves, where each node is computed expanding recursively the seed of the previous level of the tree. The algorithm returns a list that contains M seed values, one for each leaf.
- $\mathsf{ReleaseSeeds}$: the inputs of this algorithm are a binary string $\mathsf{seed}_{\mathsf{root}}$ of length λ, which represents the root of the tree, and a challenge c, a binary string of length M. The algorithm outputs the set S containing the seeds that belong to the leaves with index equal to 1 in the challenge. In order to send less information, the seed tree structure can be exploited to send a subset $\mathsf{seeds}_{\mathsf{int}} \subseteq S$ of nodes, where the seeds computed starting from the set $\mathsf{seeds}_{\mathsf{int}}$ is equal to the seeds contained in the set S.
- $\mathsf{RecoverLeaves}$: the inputs of the algorithm are the set $\mathsf{seeds}_{\mathsf{int}}$ and the binary challenge c of length M. The output is given by the set of all the seeds belonging to the leaves that can be computed starting from the set $\mathsf{seeds}_{\mathsf{int}}$, that is the binary strings corresponding to the ones of the challenge c.
- $\mathsf{SimulateSeeds}$: the input of the algorithm is a binary challenge c of length M. The algorithm computes the leaves with index equal to 1 and then it samples a random seed of length λ for each of these leaves. The output is the set $\mathsf{seeds}_{\mathsf{int}}$.

5.2 The Base OR Sigma Protocol

This section describes the sigma protocol on which the ring signature is based. To recap what a sigma protocol is, together with the properties it must satisfy, see Section 2.6.

Let R be a positive integer. Fix an element ϕ in $\mathrm{ATF}(q,n)$ and let $\phi_i = A_i \star \phi$, where A_i is a randomly generated matrix from $\mathrm{GL}_n(q)$ for each $i = 1, \ldots, R$. The NP-relation for the protocol is the following:

$$\mathscr{R} = \{(\{\phi_1, \ldots, \phi_R\}, A) \mid \exists I \in \{1, \ldots, R\} \text{ s.t. } \phi_I = A \star \phi\}.$$

In the signature, consider A_I as the secret key for the public key $\phi_I = A_I \star \phi$; then, the above relation models that the witness is the knowledge of at least a secret key for one of the public keys $\phi_1, \dots, \phi_R$ in the ring.

The problem induced by the relation $\mathscr{R}$ is a variation of sATFE.

Definition 32. *Let R be a positive integer and ϕ a public element of* ATF(q,n)*. The* R-search Alternating Trilinear Form Equivalence *(R-sATFE) problem is given by*

- Input*: $\phi_1, \dots, \phi_R$ in* ATF(q,n) *such that they are pairwise equivalent.*
- Output*: A in* $\mathrm{GL}_n(q)$ *and distinct i, j in $\{1, \dots, R\}$ such that $\phi_j = A \star \phi_i$.*

The proof from [7], Theorem 3, can be adapted to reduce tightly R-sATFE to sATFE.

Proposition 33. *Given an algorithm* Alg *that solves* R-sATFE *with probability ε, there exists a polynomial algorithm* Alg$'$*, using* Alg *as an oracle, that solves* sATFE *with probability $\frac{\varepsilon}{2}$.*

Proof. Given the instance (ϕ, ψ) of sATFE, the aim is to find A in $\mathrm{GL}_n(q)$ such that $\phi = A \star \psi$. Without loss of generality, let R be even. The algorithm Alg$'$ uniformly samples $B_1, \dots, B_R$ from $\mathrm{GL}_n(q)$ and sets

$$\phi_i = \begin{cases} B_i \star \phi & \text{for } i \in \left\{1, \dots, \frac{R}{2}\right\} \\ B_i \star \psi & \text{for } i \in \left\{\frac{R}{2}+1, \dots, R\right\}. \end{cases} \tag{5.1}$$

After a random permutation π, Alg$'$ asks the query $\left\{\phi_{\pi(i)}\right\}_{i=1,\dots,R}$ to Alg. Observe that every ϕ_i is equivalent to both ϕ and ψ, and that there is no way to decide if it is obtained from ϕ or ψ. The oracle Alg returns a matrix C and the indexes h, k such that $\phi_h = C \star \phi_k$, for $k \neq h$.
With probability $\frac{1}{2}$, k and h are not in the same partition from Equation (5.1). Suppose that this is the case, then this implies $\phi_k = B_k \star \phi$ and $\phi_h = B_h \star \psi$ (here, assume without loss of generality that π is the identity, otherwise apply its inverse on the indices of the B's). The algorithm Alg$'$ returns $B_k^{-1} C B_h$. Otherwise, if k and h are in the same partition, Alg$'$ outputs a rejection.
Observe that Alg$'$ is a polynomial-time algorithm, and that it solves sATFE with probability $\frac{\varepsilon}{2}$ using Alg as oracle. □

5.2.1 The protocol

The construction is the same used in [19], with minor changes. For example, here, there is no need to abort. The idea on which the base OR sigma protocol has been built is the following: given a public trilinear form ϕ, secret keys $\{A_1,\dots,A_R\}$, and public keys $\phi_1 = A_1 \star \phi,\dots,\phi_R = A_R \star \phi$, the prover wants to show to the verifier that she knows at least a secret key A_I among the public keys of the ring $\{\phi_1,\dots,\phi_R\}$. In order to prove that, for each $i \in \{1,\dots,R\}$, she generates random matrices B_i, and computes $\psi_i = B_i \star \phi_i$ for each public key ϕ_i in the ring; then, she sends these elements in a random order to the verifier that replies with a random bit b. If $b = 0$, the response resp is $B_I A_I$ and the verifier checks that $\mathsf{resp} \star \phi$ is in $\{\phi_1,\dots,\phi_R\}$. If $b = 1$, the response consists of $B_1,\dots,B_R$ and the verifier checks the set equality $\{B_1 \star \psi_1,\dots,B_R \star \psi_R\} = \{\phi_1,\dots,\phi_R\}$. To make the proof size logarithmic in R, since the matrices $B_1,\dots,B_R$ are generated at random, the prover can send the seed that generates them as response if $b = 1$. A Merkle tree can also be used to commit instead of sending all the elements $\psi_1,\dots,\psi_R$. In this way, when the challenge is $b = 0$, the prover appends a path of the Merkle tree to retrieve the root. Moreover, the same matrix B can be used instead of R different matrices $B_1,\dots,B_R$ without affecting the security of the scheme. All these considerations lead to the OR sigma protocol showed below.

In the following, a commitment scheme is modeled as a random oracle $\mathsf{RO}_{\mathrm{Com}}$, where the input is the committed value x and a random string r. Assume that the randomness produced by the prover derives from a seed seed expanded by the random oracle RO_E. The base OR Sigma protocol, using algorithms described in Figure 5.1, has the standard flow of every Sigma protocol: the prover sends the commitment com returned by $\mathscr{P}_{\mathscr{B}\mathsf{com}}^{\mathsf{RO}}$ to the verifier, that replies with a random challenge ch in $\{0,1\}$; then, the prover computes its response running $\mathscr{P}_{\mathscr{B}\mathsf{resp}}^{\mathsf{RO}}$ and sends it to the verifier, that accepts or rejects it according to $\mathscr{V}_{\mathscr{B}}^{\mathsf{RO}}$.

Theorem 34. *The base OR Sigma protocol with algorithms in Figure 5.1 is correct, special 2-sound, and it possesses special zero-knowledge in the Random Oracle Model, under the assumption that R-*sATFE *is intractable.*

Proof.
- **Correctness.** Given a pair $((\phi_1,\dots,\phi_R),A_I)$, such that $\phi_I = A_I \star \phi$, if the protocol is executed honestly, the verifier accepts with probability 1 by

```
$\underline{\mathscr{P}_{\mathscr{B}_{\mathsf{com}}}^{\mathsf{RO}}(\mathsf{seed}, \{\phi_1, \dots, \phi_R\})}$ :
$(B, r_1, \dots, r_R) \leftarrow \mathsf{RO}_E(\mathsf{seed})$
**for** $i = 1, \dots, R$ **do**
    $C_i \leftarrow \mathsf{RO}_{\mathrm{Com}}(B \star \phi_i, r_i)$
$(\mathsf{com}, \mathsf{tree}) \leftarrow \mathsf{MerkleTree}(C_1, \dots, C_R)$
**return** com

$\underline{\mathscr{P}_{\mathscr{B}_{\mathsf{resp}}}^{\mathsf{RO}}(\mathsf{seed}, A_I, \mathsf{tree}, \mathsf{ch})}$ :
$(B, r_1, \dots, r_R) \leftarrow \mathsf{RO}_E(\mathsf{seed})$
**if** $\mathsf{ch} = 0$ **then**
    $D \leftarrow BA_I$
    $\mathsf{path} \leftarrow \mathsf{getMerklePath}(\mathsf{tree}, I)$
    $\mathsf{resp} \leftarrow (D, \mathsf{path}, r_I)$
**else**
    $\mathsf{resp} \leftarrow \mathsf{seed}$
**return** resp
```

```
$\underline{\mathscr{V}_{\mathscr{B}}^{\mathsf{RO}}(\mathsf{com}, \mathsf{ch}, \mathsf{resp})}$ :
**if** $\mathsf{ch} = 0$ **then**
    $(D, \mathsf{path}, r) \leftarrow \mathsf{resp}$
    $\widetilde{\phi} \leftarrow D \star \phi$
    $\widetilde{\mathsf{root}} \leftarrow \mathsf{ReconstructRoot}(\mathsf{path}, \mathsf{RO}_{\mathrm{Com}}(\widetilde{\phi}, r))$
**else**
    $(B, r_1, \dots, r_R) \leftarrow \mathsf{RO}_E(\mathsf{resp})$
    **for** $i = 1, \dots, R$ **do**
        $\widetilde{C}_i \leftarrow \mathsf{RO}_{\mathrm{Com}}(B \star \phi_i, r_i)$
    $(\widetilde{\mathsf{root}}, \widetilde{\mathsf{tree}}) \leftarrow \mathsf{MerkleTree}(\widetilde{C}_1, \dots, \widetilde{C}_R)$
**if** $\mathsf{com} = \widetilde{\mathsf{root}}$ **then**
    **return accept**
**return reject**
```

Fig. 5.1 Algorithms for the base OR Sigma protocol

construction. If $\mathsf{ch} = 0$, the verifier computes

$$D \star \phi = BA_I \star \phi = B \star \phi_I$$

and uses it to reconstruct the root of the Merkle tree using the path given in the response resp. When $\mathsf{ch} = 1$, the response is the seed used by the prover, and the verifier repeats the same computations of the prover getting the root of the Merkle tree.

- **Special 2-soundness.** Given two transcripts $(\mathsf{root}, 0, (D, \mathsf{path}, r_I))$ and $(\mathsf{root}, 1, \mathsf{seed})$, the extractor $\mathscr{E}$ acts as follows. It expands the seed using RO_E to obtain the random matrix B, and uses it to compute the secret key $A_I = B^{-1}D$.
- **Special zero-knowledge.** The aim is to show that there exists a simulator $\mathscr{S}$ such that, for any $((\phi_1, \dots, \phi_R), A_I)$ with $\phi_I = A_I \star \phi$, for any $\mathsf{ch} = 0, 1$, and for any adversary $\mathscr{A}$ making at most a polynomial number of queries Q to the

random oracle, the quantity

$$\begin{aligned}\Bigg|\mathbf{P}\left[\mathscr{A}^{\mathsf{RO}}(\mathscr{P_B}^{\mathsf{RO}}((\phi_1,\dots,\phi_R),A_I,\mathsf{ch})=1\right]-\\ \mathbf{P}\left[\mathscr{A}^{\mathsf{RO}}(\mathscr{S}((\phi_1,\dots,\phi_R),\mathsf{ch}))=1\right]\Bigg|\end{aligned} \tag{5.2}$$

is negligible in λ. The simulator $\mathscr{S}$ is defined as follows:

- $\mathsf{ch}=1$: it runs the prover and outputs a valid transcript; since in this case the witness A_I is not used in the computation, the response consists of seed.
- $\mathsf{ch}=0$: it picks a random invertible matrix D and a random string r of λ bits, then it computes the commitment $C_1=\mathsf{RO}_{\mathrm{Com}}(D\star\phi,r)$. The simulator randomly generates $R-1$ dummy commitments $C_2,\dots,C_R$ in $\{0,1\}^{2\lambda}$ and creates the Merkle tree $(\mathsf{root},\mathsf{tree})=\mathsf{MerkleTree}(C_1,\dots,C_R)$. Finally, it outputs $(\mathsf{root},0,(D,\mathsf{path},r))$.

In order to prove special zero-knowledge, a sequence of simulators $\mathscr{S}_1,\mathscr{S}_2,\mathscr{S}_3$, $\mathscr{S}_4$, where $\mathscr{S}_1=\mathscr{P_B}$, $\mathscr{S}_4=\mathscr{S}$ and the others are defined below, is introduced. Fix a pair $((\phi_1,\dots,\phi_R),A_I)$, with $\phi_I=A_I\star\phi$ and an adversary $\mathscr{A}$. The case $\mathsf{ch}=1$ is straightforward, since the witness A_I is not used in the response; then, for $\mathsf{ch}=0$, Eq. (5.2) becomes

$$\begin{aligned}\Bigg|\mathbf{P}\left[\mathscr{A}^{\mathsf{RO}}(\mathscr{S}_1^{\mathsf{RO}}((\phi_1,\dots,\phi_R),A_I,0)=1\right]-\\ \mathbf{P}\left[\mathscr{A}^{\mathsf{RO}}(\mathscr{S}^{\mathsf{RO}}((\phi_1,\dots,\phi_R),0)=1\right]\Bigg|.\end{aligned}$$

The second simulator $\mathscr{S}_2$ behaves like both algorithms of the prover $\mathscr{P_B}$, except that, instead of expanding the seed with the random oracle $\mathsf{RO}_E(\mathsf{seed})$, it picks B and $r_1,\dots,r_R$ uniformly at random. This does not change the view of $\mathscr{A}$, unless it queries to the oracle the same input seed in $\{0,1\}^{\lambda}$. This happens with probability at most $\frac{Q}{2^\lambda}$, hence

$$\begin{aligned}\Bigg|\mathbf{P}\left[\mathscr{A}^{\mathsf{RO}}(\mathscr{S}_1^{\mathsf{RO}}((\phi_1,\dots,\phi_R),A_I,0)=1\right]-\\ \mathbf{P}\left[\mathscr{A}^{\mathsf{RO}}(\mathscr{S}_2^{\mathsf{RO}}((\phi_1,\dots,\phi_R),A_I,0)=1\right]\Bigg|\leq\frac{Q}{2^\lambda}.\end{aligned}$$

The third simulator $\mathscr{S}_3$ acts the same way as $\mathscr{S}_2$, except for the fact that commitments C_i, for $i \neq I$, are chosen uniformly at random. The adversary $\mathscr{A}$ does not notice this, unless it queries $\mathsf{RO}_{\mathsf{Com}}$ on input (ψ_i, r_i), where $\psi_i = B \star \phi_i$, for $i \neq I$. Let Q_{ψ_i} be the number of queries of the form $\mathsf{RO}_{\mathsf{Com}}(\psi_i, \cdot)$, since r_i has λ bits of min-entropy, the probability that $\mathscr{A}$ asks $\mathsf{RO}_{\mathsf{Com}}$ on input (ψ_i, r_i) is at most $\frac{Q_{\psi_i}}{2^\lambda}$. Without loss of generality, it is assumed that all the public keys $\{\phi_1, \dots, \phi_R\}$ are distinct, and so are $\{\psi_1, \dots, \psi_R\}$. This implies that $\sum_{i=1}^{R} \frac{Q_{\psi_i}}{2^\lambda} \leq \frac{Q}{2^\lambda}$ and

$$\Big| \mathbf{P}\left[\mathscr{A}^{\mathsf{RO}}(\mathscr{S}_2^{\mathsf{RO}}((\phi_1, \dots, \phi_R), A_I, 0) = 1\right] -$$
$$\mathbf{P}\left[\mathscr{A}^{\mathsf{RO}}(\mathscr{S}_3^{\mathsf{RO}}((\phi_1, \dots, \phi_R), A_I, 0) = 1\right] \Big| \leq \frac{Q}{2^\lambda}.$$

The fourth simulator $\mathscr{S}_4$ is the same as $\mathscr{S}_3$ but, instead of computing $\psi_I = B \star \phi_I$, it picks a uniformly random invertible matrix B', and sets $\psi_I = B' \star \phi_I$. Moreover, it uses $I = 1$ instead of the value of I given in the witness. From [19, Lemma 2.10], it is known that the index-hiding property of the Merkle trees used in the scheme does not change the view of the adversary $\mathscr{A}$, hence

$$\Big| \mathbf{P}\left[\mathscr{A}^{\mathsf{RO}}(\mathscr{S}_3^{\mathsf{RO}}((\phi_1, \dots, \phi_R), A_I, 0) = 1\right] -$$
$$\mathbf{P}\left[\mathscr{A}^{\mathsf{RO}}(\mathscr{S}_4^{\mathsf{RO}}((\phi_1, \dots, \phi_R), 0) = 1\right] \Big| = 0.$$

Combining all the results,

$$\Big| \mathbf{P}\left[\mathscr{A}^{\mathsf{RO}}(\mathscr{P}_{\mathscr{B}}{}^{\mathsf{RO}}((\phi_1, \dots, \phi_R), A_I, \mathsf{ch}) = 1\right] -$$
$$\mathbf{P}\left[\mathscr{A}^{\mathsf{RO}}(\mathscr{S}^{\mathsf{RO}}((\phi_1, \dots, \phi_R), \mathsf{ch}) = 1\right] \Big| \leq \frac{2Q}{2^\lambda},$$

and the thesis is proven since Q is polynomial in λ and hence $\frac{2Q}{2^\lambda}$ is negligible.

$\square$

5.3 Optimisations and the Main OR Sigma Protocol

In this section, the base Sigma protocol from Section 5.2 is modified to decrease the soundness error from $1/2$ to $1/2^{\lambda}$, for a security parameter λ. A straightforward strategy is to run the protocol in parallel λ times. Moreover, some modification to this protocol to obtain shorter responses are reported.

5.3.1 Using fixed weight challenges

Instead of picking the challenge from the space $\{0,1\}^{\lambda}$ uniformly, the number of ones can be forced to be a certain value, since the response for $\mathsf{ch} = 1$ is a λ-bits seed only, and it is shorter compared to the response when $\mathsf{ch} = 0$. Let M be the length of the challenge and K be the number of zeros in the challenge. Parameters have been chosen such that $\binom{M}{K} > 2^{\lambda}$ and $M > \lambda$. The challenge space becomes $C_{M,K}$, i.e. the set of binary strings of length M and weight $M-K$ (or, equivalently, with exactly K zeros).

Here, another optimization is proposed: instead of sending the challenge as a string in $C_{M,K}$, such $\binom{M}{K}$ strings are enumerated, and the integer J_{ch} referring to the position of the challenge ch can be sent in such ordering. To convert an integer into a fixed weight binary string, the combinatorial number system [83] is used. In this way, only $\left\lceil \log_2 \binom{M}{K} \right\rceil$ bits per challenge are sent at the cost of a negligible increment of the computational effort in the signing and in the verify processes. Note that usually the number of ones is fixed to be less than $M/2$, but here a string with a large number of ones and few zeros is needed; because of this, the reverse lexicographic order has been used. To sample an element from $C_{M,K}$, sample an integer J from $\{0,\dots,\binom{M}{K}-1\}$, and see this as the position of the challenge ch_J in the lexicographic order in $C_{M,K}$. The challenge is encoded by the algorithm Unrank in Figure 5.2, having complexity $O(M)$.

5.3.2 Seed tree

Seed trees are used to communicate the seed seed for each repetition of the base sigma protocol having challenge bit $\mathsf{ch} = 1$. Due to Section 5.3.1, the challenge has a larger number of ones, and this structure allows to reduce the response size.

```
$\underline{\mathsf{Unrank}(M,J,K)}$ :
$\mathsf{ch} \leftarrow (1,\ldots,1)$
$m \leftarrow M$
$I \leftarrow J$
for $i = 0,\ldots,K-1$ do
    while $\binom{m}{K-i} \geq I$ do
        $m \leftarrow m-1$
    $I \leftarrow I - \binom{m}{K-i}$
    $\mathsf{ch}_m \leftarrow 0$
    $m \leftarrow m-1$
return $\mathsf{ch}$
```

Fig. 5.2 Unranking algorithm

Given a seed tree with M leaves, if the aim is to send $M-K$ leaves, the upper bound on the number of sent nodes is given by the following proposition. This fact is well-known and given in [68]; here, the proof is added for completeness. Observe that if the number of leaves is not a power of 2, dummy leaves can be added to reach the next power of 2 and complete the binary tree. The case $K = 0$ can be excluded since in that case it is sufficient to send the root.

Proposition 35. *Given a seed tree with $M = 2^c$ leaves, if the aim is to send $M-K$ leaves, with $K \neq 0$, then at most $K\log_2 M - K\lceil\log_2 K\rceil + 2^{\lceil\log_2 K\rceil} - K$ seeds can be transmitted.*

The proof of this proposition will make use of the next technical lemma.

Lemma 36. *Let $b(n)$ be the* binary entropy function *[107, A003314] given by*

$$b(n) = \begin{cases} 0 & \text{if } n = 1 \\ \min_{i=1,\ldots,n-1}\{n + b(n-i) + b(i)\} & \text{if } n > 1 \end{cases}.$$

Then, for each natural number n,

$$b(n) = n + n\lceil\log_2 n\rceil - 2^{\lceil\log_2 n\rceil}.$$

Proof. Set $a(n) = n + n\lceil \log_2 n \rceil - 2^{\lceil \log_2 n \rceil}$. Using the characterization of $b(n)$ reported in [39, Cor. 5],

$$b(n) = n\lfloor \log_2 n \rfloor + 2n - 2 \cdot 2^{\lfloor \log_2 n \rfloor}.$$

Observe that, for any n power of 2, the relation $\lceil \log_2 n \rceil = \lfloor \log_2 n \rfloor$ is valid, otherwise, if n is not a power of 2, then $\lceil \log_2 n \rceil - \lfloor \log_2 n \rfloor = 1$. This implies that, for every n, $a(n) = b(n)$. □

Proposition 35 can now be proved.

Proof. Set $f(c,K)$ to be the function counting the maximum number of nodes to be transmitted in a tree with 2^c leaves, hiding K leaves. It is easy to see that

$$f(c,K) = \max_{i=1,\dots,K-1} \{f(c-1, K-i) + f(c-1, i)\}, \tag{5.3}$$

by induction on c.
The base step is given by $c = 1$ and it is trivial. Now let $c > 1$ and suppose

$$f(c',K) = Kc' - K\lceil \log_2 K \rceil + 2^{\lceil \log_2 K \rceil} - K$$

for every $c' < c$ and every $K = 1, \dots, c' - 1$. Equation (5.3) gives us

$$\begin{aligned} f(c,K) = \max_{i=1,\dots,K-1} \{&(K-i)(c-1) - (K-i)\lceil \log_2 (K-i) \rceil + 2^{\lceil \log_2 (K-i) \rceil} \\ &- (K-i) + i(c-1) - i\lceil \log_2 i \rceil + 2^{\lceil \log_2 i \rceil} - i\} \end{aligned}$$

and, rearranging the terms,

$$\begin{aligned} f(c,K) =& Kc + \\ &\max_{i=1,\dots,K-1} \{-2K - (K-i)\lceil \log_2 (K-i) \rceil + 2^{\lceil \log_2 (K-i) \rceil} - i\lceil \log_2 i \rceil + 2^{\lceil \log_2 i \rceil}\}. \end{aligned}$$

Now, take the minimum by changing the signs in the parentheses

$$\begin{aligned} f(c,K) =& Kc - \\ &\min_{i=1,\dots,K-1} \{2K + (K-i)\lceil \log_2 (K-i) \rceil - 2^{\lceil \log_2 (K-i) \rceil} + i\lceil \log_2 i \rceil - 2^{\lceil \log_2 i \rceil}\} \end{aligned}$$

and, setting $a(n) = n + n\lceil \log_2 n \rceil - 2^{\lceil \log_2 n \rceil}$, one obtains

$$f(c,K) = Kc - \min_{i=1,\dots,K-1} \{K + a(K-i) + a(i)\}.$$

Finally, using Lemma 36,

$$f(c,K) = Kc - K\lceil \log_2 K \rceil + 2^{\lceil \log_2 K \rceil} - K.$$

$\square$

To give a more accurate estimate of the signature length, the minimal number of nodes that must be transmitted for a tree with M leaves has been studied. For any non-negative integer K, denote with $\mathbf{b}(K)$ its binary representation and with $\mathrm{w}(\mathbf{b}(K))$ the number of ones in $\mathbf{b}(K)$.

Proposition 37. *Given a seed tree with $M = 2^c$ leaves, if the aim is to send $M - K$ leaves, with $K \neq 0$, then at least $c - \mathrm{w}(\mathbf{b}(K-1))$ seeds must be transmitted.*

Proof. Suppose to transmit $M - K$ leaves in a seed tree with $M = 2^c$ leaves. Denote with $g(c,K)$ the smallest number of nodes to send, depending on the position of the K leaves that should be kept secret. Given K leaves to hide, the best scenario is when they are close to each other as much as possible. Without loss of generality, assume that these K leaves are all on the right. The height of the largest subtree having only leaves to keep secret is $r = \lfloor \log_2 K \rfloor$. If the tree is examined vertically, from the root to the leaves, a node for each level from $c-1$ to $r+2$ plus $g(r, K-2^r)$ must be sent; $g(r, K-2^r)$ denotes the minimal number of nodes to send having a tree of height r with $K - 2^r$ leaves to hide. A graphic example is reported in Figure 5.3: a node for each level from $c-1$ to $r+2$ is sent, without sending the node at level $r+1$. Then, just iterate on the small sub-tree on the left.

Considering the base case $g(c,0) = 1$, where just the root is sent, the following equation is obtained:

$$g(c,K) = \begin{cases} 1 & \text{if } K = 0 \\ c - \lfloor \log_2 K \rfloor - 1 + g\left(\lfloor \log_2 K \rfloor, K - 2^{\lfloor \log_2 K \rfloor}\right) & \text{if } K > 0 \end{cases}. \tag{5.4}$$

For $K > 0$, write $g(c,K) = c - d(K)$, where
$d(K) = \lfloor \log_2 K \rfloor + 1 - g\left(\lfloor \log_2 K \rfloor, K - 2^{\lfloor \log_2 K \rfloor}\right)$. Let $\mathbf{b}(K) = \left(b_0, \dots, b_{\lfloor \log_2 K \rfloor}\right)$ be

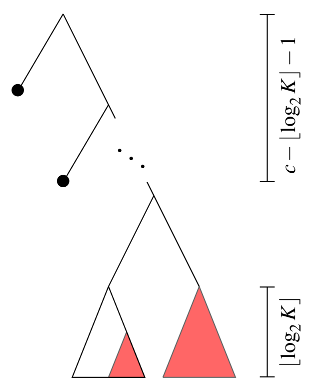


Fig. 5.3 Red sub-trees contain only leaves to hide. Black circles are the sent nodes.

the binary expansion of $K=\sum_{i=0}^{\lfloor\log_2 K\rfloor} b_i 2^i$, and let $a_1<\ldots<a_t$ be its support. Then, $t=\mathrm{w}(\mathbf{b}(K))$ and $\lfloor\log_2 K\rfloor=a_t$. At this point, K can be written as $K=\sum_{i=1}^{t} 2^{a_i}$. To show that $d(K)=a_1+t-1$, define

$$g_j=g\left(a_j,\sum_{i=1}^{j-1}2^{a_i}\right)\qquad \forall j=1,\ldots,t,$$

in particular $g_1=g(a_1,0)=1$ by (5.4). Then, from (5.4), $g_j-g_{j-1}=a_j-a_{j-1}-1$ and, summing over all indices j from 2 to t,

$$\sum_{j=2}^{t}\left(g_j-g_{j-1}\right)=\sum_{j=2}^{t}\left(a_j-a_{j-1}-1\right).$$

The left hand side is $g_t-g_1=g_t-1$, while the right hand side gives $a_t-a_1-(t-1)$. Combining these results give $g_t=a_t-a_1-t+2$. Now, from the definition of d,

$$d(K)=\lfloor\log_2 K\rfloor+1-g_t=a_t+1-g_t=a_1+t-1,$$

and the claim is proven.
It is known that $a_1=1+\mathrm{w}(\mathbf{b}(K-1))-\mathrm{w}(\mathbf{b}(K))$ [107, A007814]. Using this fact,

$$d(K)=a_1+t-1=1+\mathrm{w}(\mathbf{b}(K-1))-\mathrm{w}(\mathbf{b}(K))+t-1=\mathrm{w}(\mathbf{b}(K-1))$$

and this concludes the proof. $\square$

Given $M = 2^c$, and given $M - K$ the number of leaves that must be sent, how many nodes of the tree will be transmitted on average? An analysis given in [109], about the average number of encryption in a CST broadcast encryption scheme, answers to the above question. In fact, the structure of the seed tree is the same as the key tree used in that protocol, and sending a seed coincides with giving to a user the privilege of decrypting data. [109, Th. 8] can be reformulated to obtain the following result on seed trees.

Theorem 38. *Given a seed tree with $M = 2^c$ leaves, if the aim is to send $M - K$ leaves, the average number of seeds to transmit is given by*

$$\sum_{k=0}^{\lfloor \log_2(M-K) \rfloor} \frac{M-K}{2^k} \cdot \frac{\binom{M-2^k}{M-K-2^k} - \binom{M-2^{k+1}}{M-K-2^{k+1}}}{\binom{M-1}{M-K-1}}.$$

These bounds are used to estimate and minimize the length of the signature in Section 5.6.

5.3.3 Salting

In [19], they point out that, to make tight reductions, a salt is needed to call the RO. Moreover, for each repetition i, $1 \leq i \leq M$, of the base sigma protocol, the "salted" random oracle $\mathsf{RO}^i(\cdot) = \mathsf{RO}(\mathsf{salt}||i||\cdot)$ is used, with a random string salt of 2λ bits.

5.3.4 The main OR sigma protocol

In this new protocol, the base OR sigma protocol is run in parallel to achieve a lower soundness error. Moreover, some of the optimizations cited in this section are introduced, namely the use of a fixed weight challenge, the seed tree, and the salting. The scheme is presented in Figure 5.4. Let M be the length of the challenge and K be the number of zeroes. Both M and K are chosen accordingly to the security parameter λ. The challenge space S_{ch} is the set $\{0, \ldots, \binom{M}{K} - 1\}$.

Theorem 39. *The main OR Sigma protocol with algorithms in Figure 5.4 is correct, special 2-sound, and it possesses both high min-entropy and special zero-knowledge in the Random Oracle Model, under the assumption that R-*sATFE *is intractable.*

```
$\mathscr{P}_{\mathscr{M}}{}^{\mathsf{RO}}_{\mathsf{com}}(\{\phi_1,\dots,\phi_R\},M)$ :
$\mathsf{seed} \leftarrow\!\$\ \{0,1\}^{\lambda}$
$\mathsf{salt} \leftarrow\!\$\ \{0,1\}^{2\lambda}$
$\mathsf{RO}' \leftarrow \mathsf{RO}(\mathsf{salt}||\cdot)$
$(\mathsf{seed}^{(1)},\dots,\mathsf{seed}^{(M)}) \leftarrow \mathsf{SeedTree}^{\mathsf{RO}'}(\mathsf{seed},M)$
for $i = 1,\dots,M$ do
    $\mathsf{RO}^i \leftarrow \mathsf{RO}'(i||\cdot)$
    $\mathsf{com}^{(i)} \leftarrow \mathscr{P}_{\mathscr{B}}{}^{\mathsf{RO}^i}_{\mathsf{com}}\left(\mathsf{seed}^{(i)},\{\phi_1,\dots,\phi_R\}\right)$
endfor
$\mathsf{com} \leftarrow (\mathsf{salt},(\mathsf{com}^{(i)})_{i=1,\dots,M})$
return com

$\mathscr{P}_{\mathscr{M}}{}^{\mathsf{RO}}_{\mathsf{resp}}(A_I,J_{\mathsf{ch}})$ :
$\mathsf{ch} \leftarrow \mathsf{Unrank}(J_{\mathsf{ch}})$
for $i$ s.t. $\mathsf{ch}_i = 0$ do
    retrieve $\mathsf{tree}^{(i)}$ and $\mathsf{seed}^{(i)}$
    $\mathsf{resp}^{(i)} \leftarrow \mathscr{P}_{\mathscr{B}}{}^{\mathsf{RO}^i}_{\mathsf{resp}}\left(\mathsf{seed}^{(i)},A_I,\mathsf{tree}^{(i)},\mathsf{ch}_i\right)$
endfor
$\mathsf{seeds}_{\mathrm{int}} \leftarrow \mathsf{ReleaseSeeds}^{\mathsf{RO}'}(\mathsf{seed},\mathsf{ch})$
$\mathsf{resp} \leftarrow (\mathsf{seeds}_{\mathrm{int}},(\mathsf{resp}^{(i)})_{\mathsf{ch}_i=0})$
return resp
```

```
$\mathscr{V}_{\mathscr{M}}{}^{\mathsf{RO}}(\mathsf{com},J_{\mathsf{ch}},\mathsf{resp})$ :
$\mathsf{ch} \leftarrow \mathsf{Unrank}(J_{\mathsf{ch}})$
$\mathsf{RO}' \leftarrow \mathsf{RO}(\mathsf{salt}||\cdot)$
$(\mathsf{seeds}_{\mathrm{int}},(\mathsf{resp}^{(i)})_{\mathsf{ch}_i=0}) \leftarrow \mathsf{resp}$
$(\mathsf{resp}^{(i)})_{\mathsf{ch}_i=1} \leftarrow \mathsf{RecoverLeaves}^{\mathsf{RO}'}(\mathsf{seeds}_{\mathrm{int}},\mathsf{ch})$
for $i = 1,\dots,M$ do
    if $\mathscr{V}_{\mathscr{B}}{}^{\mathsf{RO}'}\left(\mathsf{com}^{(i)},\mathsf{ch}_i,\mathsf{resp}^{(i)}\right)$ rejects then
        return reject
    endif
endfor
return accept
```

Fig. 5.4 Algorithms for the main OR Sigma protocol

Proof.

- **Correctness.** Since the main OR sigma protocol is a parallel repetition of the base OR sigma protocol with some optimisations, the correctness is implied by Theorem 34 and by the correctness of the algorithms of the seed trees.

- **High min-entropy.** Since the commitment com depends on a random salt of 2λ bits and on a λ bits seed, the scheme has high min-entropy.

- **Special 2-soundness.** Let $(\mathsf{com},\mathsf{ch},\mathsf{resp})$ and $(\mathsf{com},\mathsf{ch}',\mathsf{resp}')$ be two accepting transcripts, where $\mathsf{com} = (\mathsf{salt},(\mathsf{root}^{(i)})_{i=1,\dots,M})$ and $\mathsf{ch} \neq \mathsf{ch}'$. Define the extractor $\mathscr{E}$ as follows: since $\mathsf{ch} \neq \mathsf{ch}'$, there exists j such that the j-th bits of ch and ch' are different. Without loss of generality let $\mathsf{ch}_j = 0$ and $\mathsf{ch}'_j = 1$. By construction, $\mathsf{resp} = (\mathsf{seeds_{int}}, (\mathsf{resp}^{(i)})_{\mathsf{ch}_i=0}))$, with $\mathsf{resp}^{(j)} = (D^{(j)}, \mathsf{path}^{(j)}, r_I^{(j)})$. Moreover, $\mathsf{resp}' = (\mathsf{seeds}'_{\mathsf{int}}, (\mathsf{resp}^{(i)})_{\mathsf{ch}'_i=0}))$, and from $\mathsf{seeds}'_{\mathsf{int}}$ the extractor $\mathscr{E}$ retrieves the seed for the j-th parallel execution of the protocol, computing $\mathsf{seed}^{(j)} = \mathsf{RecoverLeaves}(\mathsf{seeds}'_{\mathsf{int}}, \mathsf{ch}')$. In this way, proceeding as in the proof of Theorem 34, the extractor can retrieve the secret key A_I.

- **Special zero-knowledge.** The aim is to show that there exists a simulator $\mathscr{S}$ such that, for any $((\phi_1, \dots, \phi_R), A_I)$ with $\phi_I = A_I \star \phi$, for any ch, and for any adversary $\mathscr{A}$ making at most a polynomial number of queries Q to the "salted" random oracle (the oracle having as prefix the string salt given in the transcript), the quantity

$$\Big|\mathbf{P}\left[\mathscr{A}^{\mathsf{RO}}(\mathscr{P_M}^{\mathsf{RO}}((\phi_1, \dots, \phi_R), A_I, \mathsf{ch}) = 1\right] - \mathbf{P}\left[\mathscr{A}^{\mathsf{RO}}(\mathscr{S}((\phi_1, \dots, \phi_R), \mathsf{ch})) = 1\right]\Big| \tag{5.5}$$

is negligible in λ. The simulator $\mathscr{S}$ is defined as follows:

  – it generates at random the 2λ bit string salt. Then it computes $\mathsf{seeds_{int}}$ using the algorithm $\mathsf{SimulateSeeds}(\mathsf{ch})$, and then $(\mathsf{seed}^{(i)})_{\mathsf{ch}_i=1}$ are given by $\mathsf{RecoverLeaves}(\mathsf{seeds_{int}}, \mathsf{ch})$.

  – For $\mathsf{ch}_i = 0$, the simulator behaves as the simulator of the base OR sigma protocol from the proof of Theorem 34, outputting $(\mathsf{com}^{(i)}, 0, \mathsf{resp}^{(i)})$.

  – For $\mathsf{ch}_i = 1$, the simulator $\mathscr{S}$ computes the root of the Merkle tree $\mathsf{root}^{(i)}$ from $\mathsf{seed}^{(i)}$.

  – Then, $\mathscr{S}$ outputs the transcript

$$((\mathsf{salt}, (\mathsf{com}_i)_{i=1,\dots,M}), \mathsf{ch}, (\mathsf{seeds_{int}}, (\mathsf{resp}_i)_{\mathsf{ch}_i=0})).$$

To prove special zero-knowledge, a sequence of simulators $\mathscr{S}_1 = \mathscr{P_M}, \mathscr{S}_2$, $\mathscr{S}_3 = \mathscr{S}$ is introduced. Fix a pair $((\phi_1, \dots, \phi_R), A_I)$, with $\phi_I = A_I \star \phi$, and an adversary $\mathscr{A}$. The second simulator $\mathscr{S}_2$ behaves like the prover $\mathscr{P_M}$, except for the way it generates $\mathsf{seed}^{(i)}$, for $i = 1, \dots, M$. The simulator runs $\mathsf{SimulateSeeds}$ and $\mathsf{RecoverLeaves}$ as explained above, hence it determines $(\mathsf{seed}_i)_{\mathsf{ch}_i=1}$. Then, it picks uniformly at random the remaining seeds $(\mathsf{seed}_i)_{\mathsf{ch}_i=1}$. Using [19, Lemma 2.11],

$$\left|\mathbf{P}\left[\mathscr{A}^{\mathsf{RO}}(\mathscr{S}_1^{\mathsf{RO}}((\phi_1,\dots,\phi_R),A_I,\mathsf{ch})=1\right]-\right.$$

$$\left.\mathbf{P}\left[\mathscr{A}^{\mathsf{RO}}(\mathscr{S}_2^{\mathsf{RO}}((\phi_1,\dots,\phi_R),A_I,\mathsf{ch})=1\right]\right|\leq\frac{Q}{2^\lambda}.$$

The third simulator $\mathscr{S}_3$ acts the same as $\mathscr{S}_2$, except that it uses the simulator for the base OR sigma protocol to compute $\mathsf{com}^{(i)}$ and $\mathsf{resp}^{(i)}$ for $\mathsf{ch}_i = 0$. Using the zero-knowledge property of the latter, the advantage of any adversary $\mathscr{A}$ for distinguishing the simulator from a honest prover is at most $\frac{2Q_i}{2^\lambda}$, where Q_i is the number of queries of the form $\mathsf{RO}(\mathsf{salt}||i||\cdot)$ that $\mathscr{A}$ makes to the random oracle. Hence,

$$\left|\mathbf{P}\left[\mathscr{A}^{\mathsf{RO}}(\mathscr{S}_2^{\mathsf{RO}}((\phi_1,\dots,\phi_R),A_I,\mathsf{ch})=1\right]-\right.$$

$$\left.\mathbf{P}\left[\mathscr{A}^{\mathsf{RO}}(\mathscr{S}_3^{\mathsf{RO}}((\phi_1,\dots,\phi_R),\mathsf{ch})=1\right]\right|\leq\sum_{\substack{i \text{ s.t.}\\ \mathsf{ch}_i=0}}\frac{2Q_i}{2^\lambda},$$

and, moreover, $\sum_{\substack{i \text{ s.t.}\\ \mathsf{ch}_i=0}}\frac{2Q_i}{2^\lambda}\leq\frac{2Q}{2^\lambda}$.

Combining these results, one gets

$$\left|\mathbf{P}\left[\mathscr{A}^{\mathsf{RO}}(\mathscr{P}_{\mathscr{B}}{}^{\mathsf{RO}}((\phi_1,\dots,\phi_R),A_I,\mathsf{ch})=1\right]-\right.$$

$$\left.\mathbf{P}\left[\mathscr{A}^{\mathsf{RO}}(\mathscr{S}^{\mathsf{RO}}((\phi_1,\dots,\phi_R),\mathsf{ch})=1\right]\right|\leq\frac{3Q}{2^\lambda},$$

and the thesis is proven since Q is polynomial in λ and hence $\frac{3Q}{2^\lambda}$ is negligible.

□

5.3.5 Tags and linkability

Following the construction in [19], "tags" can be added to the sigma protocol. Given the group action of $\mathrm{GL}_n(q)$ over $\mathrm{ATF}(q,n)$ from Definition 30, another action of $\mathrm{GL}_n(q)$ on the set of alternating trilinear forms must be introduced. Invertible matrices can act on a huge variety of sets, such as spaces of matrices or tensors [64]. For example, the following action, similar to ICE from [8], is a good candidate.

Definition 40. *The group action* $(\mathrm{GL}_n(q), \mathrm{ATF}(q,n), \bullet)$ *is defined by*

$$\begin{aligned} \bullet : \mathrm{GL}_n(q) \times \mathrm{ATF}(q,n) &\to \mathrm{ATF}(q,n) \\ (A, \phi) &\mapsto \phi \circ A^{-1}. \end{aligned} \tag{5.6}$$

This action leads to the following problem.

Definition 41. *The* Inverse Alternating Trilinear Form Equivalence *(*IATFE*) problem is given by*

- Input*:* ϕ, ψ *in* $\mathrm{ATF}(q,n)$.
- Output*: YES if there exists* A *in* $\mathrm{GL}_n(q)$ *such that* $\phi = A \bullet \psi$*, and NO otherwise.*

The search Inverse Alternating Trilinear Form Equivalence *(*sIATFE*) problem is given by*

- Input*:* ϕ, ψ *in* $\mathrm{ATF}(q,n)$ *such that they are equivalent.*
- Output*:* A *in* $\mathrm{GL}_n(q)$ *such that* $\phi = A \bullet \psi$.

Given two fixed elements ϕ, ψ in $\mathrm{ATF}(q,n)$ and a set of secret keys $\{A_1, \dots, A_R\}$, corresponding to public keys $\{\phi_1, \dots, \phi_R\}$ such that $\phi_i = A_I \star \phi$ for each i from 1 to R, define the "tag" associated to the I-th public key as $\tau_I = A_I \bullet \psi$. When the I-th user signs a message, he appends its tag to the signature. A verifier can link two signatures if they possess the same tag. The OR sigma protocol is modified to add the proof that τ_I is generated by the same secret key A_I. Here, the base OR Sigma protocol with tags, using the same structure of the base Sigma protocol in Figure 5.1 with algorithms from Figure 5.5, is presented. The main differences with the protocol of Section 5.2 is the introduction of the proof of knowledge for the tag τ_I, the replacement of the commitment as the hash of the concatenation of root and the masked tag τ'.

Starting from the base Sigma protocol with tags, the soundness error can be reduced from $\frac{1}{2}$ to $\frac{1}{2^\lambda}$, for a security parameter λ, performing parallel repetitions of the protocol. The same optimisations used for the main Sigma protocol can also be adopted, obtaining the algorithms $\mathcal{P}_{\mathcal{M}L,\mathsf{com}}$,$\mathcal{P}_{\mathcal{M}L,\mathsf{resp}}$, and $\mathcal{V}_{\mathcal{M}L}$.

Observe that both the base and the main OR sigma protocol with tag share the same security properties of Theorem 34 and Theorem 39.

```
$\underline{\mathscr{P}^{\mathsf{RO}}_{\mathscr{B}L,\mathsf{com}}(\mathsf{seed}, \{\phi_1, \dots, \phi_R\}, \tau)}$ :
$(B, r_1, \dots, r_R) \leftarrow \mathsf{RO}_E(\mathsf{seed})$
for $i = 1, \dots, R$ do
    $C_i \leftarrow \mathsf{RO}_{\mathrm{Com}}(B \star \phi_i, r_i)$
$(\mathsf{root}, \mathsf{tree}) \leftarrow \mathsf{MerkleTree}(C_1, \dots, C_R)$
$\tau' \leftarrow B \bullet \tau$
return $\mathsf{RO}_H(\mathsf{root}, \tau')$

$\underline{\mathscr{P}^{\mathsf{RO}}_{\mathscr{B}L,\mathsf{resp}}(\mathsf{seed}, A_I, \mathsf{tree}, \mathsf{ch})}$ :
$(B, r_1, \dots, r_R) \leftarrow \mathsf{RO}_E(\mathsf{seed})$
if $\mathsf{ch} = 0$ then
    $D \leftarrow BA_I$
    $\mathsf{path} \leftarrow \mathsf{getMerklePath}(\mathsf{tree}, I)$
    $\mathsf{resp} \leftarrow (D, \mathsf{path}, r_I)$
else
    $\mathsf{resp} \leftarrow \mathsf{seed}$
return $\mathsf{resp}$
```

```
$\underline{\mathscr{V}^{\mathsf{RO}}_{\mathscr{B}L}(\tau, \mathsf{com}, \mathsf{ch}, \mathsf{resp})}$ :
if $\mathsf{ch} = 0$ then
    $(D, \mathsf{path}, r) \leftarrow \mathsf{resp}$
    $\widetilde{\phi} \leftarrow D \star \phi$
    $\widetilde{\mathsf{root}} \leftarrow \mathsf{ReconstructRoot}(\mathsf{path}, \mathsf{RO}_{\mathrm{Com}}(\widetilde{\phi}, r))$
    $\widetilde{\tau} \leftarrow D \bullet \psi$
else
    $(B, r_1, \dots, r_R) \leftarrow \mathsf{RO}_E(\mathsf{resp})$
    for $i = 1, \dots, R$ do
        $\widetilde{C}_i \leftarrow \mathsf{RO}_{\mathrm{Com}}(B \star \phi_i, r_i)$
    $\widetilde{\tau} \leftarrow B \bullet \tau$
    $(\widetilde{\mathsf{root}}, \widetilde{\mathsf{tree}}) \leftarrow \mathsf{MerkleTree}(\widetilde{C}_1, \dots, \widetilde{C}_R)$
if $\mathsf{com} = \mathsf{RO}_H(\widetilde{\mathsf{root}}, \widetilde{\tau})$ then
    return accept
return reject
```

Fig. 5.5 Algorithms for the base OR Sigma protocol with tag

5.4 The (Linkable) Ring Signature Scheme

Given the main OR sigma protocol of the previous section, the Fiat-Shamir transform is applied to achieve a ring signature. Observe that the sigma protocol is *commitment reproducible* if the salt used by the prover is added to the signature. In fact, by definition, a sigma protocol is commitment reproducible if, given an accepting transcript $(\mathsf{com}, \mathsf{ch}, \mathsf{resp})$, there exists a polynomial time algorithm RecCom such that, on input ch and resp, it returns the commitment com with overwhelming probability. Figure 5.6 presents the algorithms for both the main OR sigma protocol and the version with tag.

In this way, the signing algorithm Sign returns the challenge and the response, reducing the size of the signature. The scheme uses an hash function $\mathscr{H}_{\mathrm{FS}}$ with digest in the challenge space $\{0, \dots, \binom{M}{K} - 1\}$, modelled by a Random Oracle. Figure 5.7 reports the (non-linkable) ring signature scheme TRIFORS. The algorithm Setup sets the public parameters accordingly to the analysis of Section 5.6; moreover, alternating trilinear forms ϕ and ψ are sampled at random from $\mathrm{ATF}(n, q)$.

```
$\underline{\mathsf{RecCom}\,(\mathsf{salt},\{\mathsf{pk}_1,\dots,\mathsf{pk}_R\},J_{\mathsf{ch}},\mathsf{resp})}$ :
$\{\phi_1,\dots,\phi_R\} \leftarrow \{\mathsf{pk}_1,\dots,\mathsf{pk}_R\}$
$\mathsf{RO}' \leftarrow \mathsf{RO}(\mathsf{salt}||\cdot)$
$\mathsf{ch} \leftarrow \mathsf{Unrank}(J_{\mathsf{ch}})$
$(\mathsf{seeds}_{\mathrm{int}},(\mathsf{resp}^{(i)})_{\mathsf{ch}_i=0}) \leftarrow \mathsf{resp}$
for $i$ s.t. $\mathsf{ch}_i = 0$ then
    $\mathsf{RO}^i \leftarrow \mathsf{RO}'(i||\cdot)$
    $(D^{(i)},\mathsf{path}^{(i)},r^{(i)}) \leftarrow \mathsf{resp}^{(i)}$
    $\phi^{(i)} \leftarrow D^{(i)} \star \phi$
    $c^{(i)} \leftarrow \mathsf{RO}_{\mathrm{Com}}(\phi^{(i)},r^{(i)})$
    $\mathsf{root}^{(i)} \leftarrow \mathsf{ReconstructRoot}(\mathsf{path}^{(i)},c^{(i)})$
$(\mathsf{seed}^{(i)})_{\mathsf{ch}=1} \leftarrow \mathsf{RecoverLeaves}(\mathsf{seeds}_{\mathrm{int}},\mathsf{ch})$
for $i$ s.t. $\mathsf{ch}_i = 1$ then
    $\mathsf{RO}^i \leftarrow \mathsf{RO}'(i||\cdot)$
    $(B^{(i)},r_1^{(i)},\dots,r_R^{(i)}) \leftarrow \mathsf{RO}^i_E(\mathsf{seed}^{(i)})$
    for $j = 1,\dots,R$ do
        $C_j^{(i)} \leftarrow \mathsf{RO}^i_{\mathrm{Com}}(B^{(i)} \star \phi_j, r_j^{(i)})$
    $(\mathsf{root}^{(i)},\mathsf{tree}^{(i)}) \leftarrow \mathsf{MerkleTree}(C_1^{(i)},\dots,C_R^{(i)})$
$\mathsf{com} \leftarrow (\mathsf{root}^{(i)})_{i=1,\dots,M}$
return $\mathsf{com}$
```

```
$\underline{\mathsf{RecCom}_L\,(\mathsf{salt},\{\mathsf{pk}_1,\dots,\mathsf{pk}_R\},\tau,J_{\mathsf{ch}},\mathsf{resp})}$ :
$\{\phi_1,\dots,\phi_R\} \leftarrow \{\mathsf{pk}_1,\dots,\mathsf{pk}_R\}$
$\mathsf{RO}' \leftarrow \mathsf{RO}(\mathsf{salt}||\cdot)$
$\mathsf{ch} \leftarrow \mathsf{Unrank}(J_{\mathsf{ch}})$
$(\mathsf{seeds}_{\mathrm{int}},(\mathsf{resp}^{(i)})_{\mathsf{ch}_i=0}) \leftarrow \mathsf{resp}$
for $i$ s.t. $\mathsf{ch}_i = 0$ then
    $\mathsf{RO}^i \leftarrow \mathsf{RO}'(i||\cdot)$
    $(D^{(i)},\mathsf{path}^{(i)},r^{(i)}) \leftarrow \mathsf{resp}^{(i)}$
    $\phi^{(i)} \leftarrow D^{(i)} \star \phi$
    $c^{(i)} \leftarrow \mathsf{RO}_{\mathrm{Com}}(\phi^{(i)},r^{(i)})$
    $\mathsf{root}^{(i)} \leftarrow \mathsf{ReconstructRoot}(\mathsf{path}^{(i)},c^{(i)})$
$(\mathsf{seed}^{(i)})_{\mathsf{ch}=1} \leftarrow \mathsf{RecoverLeaves}(\mathsf{seeds}_{\mathrm{int}},\mathsf{ch})$
for $i$ s.t. $\mathsf{ch}_i = 1$ then
    $\mathsf{RO}^i \leftarrow \mathsf{RO}'(i||\cdot)$
    $(B^{(i)},r_1^{(i)},\dots,r_R^{(i)}) \leftarrow \mathsf{RO}^i_E(\mathsf{seed}^{(i)})$
    for $j = 1,\dots,R$ do
        $C_j^{(i)} \leftarrow \mathsf{RO}^i_{\mathrm{Com}}(B^{(i)} \star \phi_j, r_j^{(i)})$
    $(\mathsf{root}^{(i)},\mathsf{tree}^{(i)}) \leftarrow \mathsf{MerkleTree}(C_1^{(i)},\dots,C_R^{(i)})$
$\iota \leftarrow i$ s.t. $\mathsf{ch}_i = 1$
$\tau' \leftarrow B^{(\iota)} \bullet \tau$
$\mathsf{com} \leftarrow (\mathsf{RO}^i_H(\mathsf{root}^{(i)},\tau'))_{i=1,\dots,M}$
return $\mathsf{com}$
```

Fig. 5.6 Commitment Reproducibility algorithms

Using standard techniques, the following result on the security of TRIFORS can be proved.

Theorem 42. *The ring signature TRIFORS from Figure 5.7 is correct, unforgeable and non-frameable in the Random Oracle Model under the assumption that* R-sATFE *is intractable.*

The same techniques can also be used to obtain a linkable ring signature from the main OR sigma protocol with tags. The construction is the same as the non-linkable scheme, this time using the main OR sigma protocol with tags of Subsection 5.3.5. Its name is Link-TRIFORS and it is reported in Figure 5.8.

```
$\underline{\mathsf{Setup}(1^\lambda)}$ :
$q, n$ s.t. $q^{\frac{2}{3}} \leq n^{12}$, $2^\lambda \leq q^{\frac{n}{2}}$ (Section 5.5)
$M, K$ s.t. $2^\lambda < \binom{M}{K}$, $\lambda < M$ (Section 5.3.1)
$\phi \leftarrow \mathrm{ATF}(q, n)$
$\mathsf{pp} \leftarrow (q, n, M, K, \phi)$
return $\mathsf{pp}$

$\underline{\mathsf{Sign}(\mathsf{sk}_I, \mathsf{msg}, \{\mathsf{pk}_1, \dots, \mathsf{pk}_R\})}$ :
$\mathsf{com} \leftarrow \mathscr{P}\mathscr{M}^{\mathsf{RO}}_{\mathsf{com}}(\{\mathsf{pk}_1, \dots, \mathsf{pk}_R\})$
$(\mathsf{salt}, (\mathsf{com}_i)_{i=1,\dots,M}) \leftarrow \mathsf{com}$
$J_{\mathsf{ch}} \leftarrow \mathscr{H}_{\mathrm{FS}}(\mathsf{msg}, \{\mathsf{pk}_1, \dots, \mathsf{pk}_R\}, \mathsf{com})$
$\mathsf{ch} \leftarrow \mathsf{Unrank}(J_{\mathsf{ch}})$
$\mathsf{resp} \leftarrow \mathscr{P}\mathscr{M}^{\mathsf{RO}}_{\mathsf{resp}}(\mathsf{sk}_I, \mathsf{ch})$
return $\sigma \leftarrow (\mathsf{salt}, J_{\mathsf{ch}}, \mathsf{resp})$

$\underline{\mathsf{KGen}(\mathsf{pp})}$ :
$A \leftarrow\$ \mathrm{GL}_n(q)$
$\mathsf{sk} \leftarrow A$
$\mathsf{pk} \leftarrow A \star \phi$
return $(\mathsf{sk}, \mathsf{pk})$

$\underline{\mathsf{Verify}(\{\mathsf{pk}_1, \dots, \mathsf{pk}_R\}, \mathsf{msg}, \sigma)}$ :
$(\mathsf{salt}, J_{\mathsf{ch}}, \mathsf{resp}) \leftarrow \sigma$
$\mathsf{com} \leftarrow \mathsf{RecCom}(\mathsf{salt}, \{\mathsf{pk}_1, \dots, \mathsf{pk}_R\}, J_{\mathsf{ch}}, \mathsf{resp})$
$J' \leftarrow \mathscr{H}_{\mathrm{FS}}(\mathsf{msg}, \{\mathsf{pk}_1, \dots, \mathsf{pk}_R\}, \mathsf{com})$
return $J' = J_{\mathsf{ch}} \wedge \mathscr{V}_{\mathscr{M}}{}^{\mathsf{RO}}(\mathsf{com}, J_{\mathsf{ch}}, \mathsf{resp})$
```

Fig. 5.7 TRIFORS algorithms.

Theorem 43. *The linkable ring signature Link-TRIFORS from Figure 5.8 is correct, linkable, linkable anonymous and non-frameable in the Random Oracle Model under the assumption that both R-*sATFE *and* sIATFE *are intractable.*

The proof of the above theorem is quite common in the literature and it is a slight modification of the one given in [19].

5.5 Solving sATFE to Attack the Schemes

Two scenarios can be distinguished: attacking the ring signature scheme and attacking the linkable ring signature scheme. Forging the non-linkable ring signature can be reduced to attacking the construction from [131], while the linkable version involves additional information regarding the tag τ.

```
$\underline{\mathsf{Setup}_L(1^\lambda)}$ :
$q,n$ s.t. $q^{\frac{2}{3}} \leq n^{12}$, $2^\lambda \leq q^{\frac{n}{2}}$ (Section 5.5)
$M,K$ s.t. $2^\lambda < \binom{M}{K}$, $\lambda < M$ (Section 5.3.1)
$\phi, \psi \leftarrow \mathrm{ATF}(q,n)$
$\mathsf{pp} \leftarrow (q,n,M,K,\phi,\psi)$
**return** $\mathsf{pp}$

$\underline{\mathsf{KGen}_L(\mathsf{pp})}$ :
$A \leftarrow\!\!\$\ \mathrm{GL}_n(q)$
$\mathsf{sk} \leftarrow A$
$\mathsf{pk} \leftarrow A \star \phi$
**return** $(\mathsf{sk},\mathsf{pk})$

$\underline{\mathsf{Sign}_L(\mathsf{sk}_I,\mathsf{msg},\{\mathsf{pk}_1,\dots,\mathsf{pk}_R\})}$ :
$\tau \leftarrow \mathsf{sk}_I \bullet \psi$
$\mathsf{com} \leftarrow \mathscr{P}_{\mathscr{M}L,\mathsf{com}}^{\mathsf{RO}}(\{\mathsf{pk}_1,\dots,\mathsf{pk}_R\},\tau)$
$(\mathsf{salt},(\mathsf{com}_i)_{i=1,\dots,M}) \leftarrow \mathsf{com}$
$J_{\mathsf{ch}} \leftarrow \mathscr{H}_{\mathrm{FS}}(\mathsf{msg},\{\mathsf{pk}_1,\dots,\mathsf{pk}_R\},\tau,\mathsf{com})$
$\mathsf{ch} \leftarrow \mathsf{Unrank}(J_{\mathsf{ch}})$
$\mathsf{resp} \leftarrow \mathscr{P}_{\mathscr{M}L,\mathsf{resp}}^{\mathsf{RO}}(\mathsf{sk}_I,\mathsf{ch})$
**return** $\sigma \leftarrow (\mathsf{salt},\tau,J_{\mathsf{ch}},\mathsf{resp})$

$\underline{\mathsf{Link}_L(\sigma_1,\sigma_2)}$ :
$(\mathsf{salt}_1,\tau_1,\mathsf{ch}_1,\mathsf{resp}_1) \leftarrow \sigma_1$
$(\mathsf{salt}_2,\tau_2,\mathsf{ch}_2,\mathsf{resp}_2) \leftarrow \sigma_2$
**if** $\tau_1 = \tau_2$ **then**
    **return true**
**return false**

$\underline{\mathsf{Verify}_L(\{\mathsf{pk}_1,\dots,\mathsf{pk}_R\},\mathsf{msg},\sigma)}$ :
$(\mathsf{salt},J_{\mathsf{ch}},\mathsf{resp}) \leftarrow \sigma$
$\mathsf{com} \leftarrow \mathsf{RecCom}_L(\mathsf{salt},\{\mathsf{pk}_1,\dots,\mathsf{pk}_R\},\tau,J_{\mathsf{ch}},\mathsf{resp})$
$J' \leftarrow \mathscr{H}_{\mathrm{FS}}(\mathsf{msg},\{\mathsf{pk}_1,\dots,\mathsf{pk}_R\},\tau,\mathsf{com})$
**return** $J' = J_{\mathsf{ch}} \wedge \mathscr{V}_{\mathscr{M}L}^{\mathsf{RO}}(\tau,\mathsf{com},J_{\mathsf{ch}},\mathsf{resp})$
```

Fig. 5.8 Link-TRIFORS algorithms.

5.5.1 Attacks to the ring signature TRIFORS

A possible approach to solve a generic instance (ϕ, ψ) of sATFE consists in solving the following polynomial system

$$\begin{cases} XY = I_n \\ \phi(Xu,Xv,w) = \phi_I(u,v,Yw). \end{cases} \tag{5.7}$$

Here, the public key A is represented by variables in X, imposed to be invertible with inverse Y, while the second row models how the matrix A acts on the form ϕ. This leads to a system of $n^2 + \binom{n}{3}$ quadratic equations in $2n^2$ variables.

[131] shows an estimation of the complexity of the F5 algorithm (one of the most efficient algorithms to compute a Gröbner basis) for the above system. They obtain an upper bound of $O(2^{6\omega n \log_2 n})$, where ω is the matrix multiplication exponent, taken equal to 2 for conservative reasons.

[131, Sect. 5.2] also recalls the heuristic attack *Gröbner basis with partial information* that solves sATFE in time

$$O(q^{2n/3} \cdot n^{2\omega} \cdot \log_2(q)). \tag{5.8}$$

A collision finding approach is used to find partial information, based on the birthday attack. The "partial information" means that a column of A, and hence of X, is known. This implies constraints also on the variables in Y, leading to a system of polynomials in $2(n^2 - n)$ variables. Some experiments show that this new system is solvable in polynomial time but finding the partial information needed is still an exponential task, leading to the overall complexity given in Eq. (5.8).

In [17], various improvements of the approach introduced above are presented. The author focuses on the particular cases where $n = 9, 10, 11$, improving the way of finding "partial information" in order to solve the system 5.7 easily. Table 5.3 reports the results . These are the best-known attacks to the problem and the parameters

n	Complexity	Note
9	$O(q)$	-
10	$O(q^6)$	-
10	$O(1)$	For $1/q$ of the keys
11	$O(q^4)$	-
$n > 10$ even	$O(q^{\frac{n}{2}+c})$	Conjectured, c constant

Table 5.3 Attacks to sATFE from [17].

have been selected according to them. The values $n \geq 10$ have been chosen and, when $n = 10$, to avoid the costant-time attack, in the setup phase the algorithm must check if the selected origin ϕ does not fall in the $1/q$ fraction of weak keys. This means that the randomly generated trilinear form ϕ should not have a rank-4 point (see [17] for further details and definitions). This computation needs to be performed just once, in the Setup algorithm. Then, q is generated such that

- $2^{\lambda} \leq q^{2/3} \cdot n^{2\omega} \cdot \log_2(q)$, from Eq (5.8);

- $q^{2/3} \leq n^{12}$, from the F5 algorithm analysis;
- $q^6 \geq 2^\lambda$ for $n = 10$, and $q^{\frac{n}{2}} \geq 2^\lambda$ for $n > 10$ even, from Table 5.3.

5.5.2 Attacks to the linkable ring signature Link-TRIFORS

In the case of the linkable scheme, the use of the action $\bullet$ gives more information about the secret key A. The same approach of the previous section can be used, by building the system

$$\begin{cases} XY = I_n \\ \phi(Xu, Xv, w) = \phi_I(u, v, Yw) \\ \psi(Yu, Yv, w) = \tau(u, v, Xw), \end{cases} \tag{5.9}$$

where τ is the tag and it is given by the action of the inverse of A, that is modelled by variables in Y. This system has $n^2 + 2\binom{n}{3}$ quadratic equations in $2n^2$ variables. The degree of regularity is estimated asymptotically as $\frac{3}{2}n$. Hence, the F5 algorithm runs in time at most

$$O\left((2n^2)^{\omega n 3/2}\right) = O\left(n^{3\omega n}\right).$$

The number of equations is less than the double of equations in the case without linkability and the analysis done before can be adapted here, since the collision finding argument from [131] does not involve the number of equations and it is only based on the secret matrix A. The best-known algorithm to attack the scheme runs in time

$$O(q^{2n/3} \cdot n^{2\omega} \cdot \log_2(q)).$$

Even in this case, attacks from [17] must be taken into account. The same preventive measures given in the non-linkable case (Subsection 5.5.1) are used: for $n = 10$, the choice of the forms ϕ and ψ must be careful to avoid weak instances of the sIATFE and sATFE problems. Furthermore, additional information about the secret key, in the form of how its inverse acts on the trilinear form ϕ, could be crucial to devise an efficient attack: this fact requires further analysis in the future.
For the linkable scheme, $n \geq 10$ is chosen, and then q is picked such that

- $2^\lambda \leq q^{2/3} \cdot n^{2\omega} \cdot \log_2(q)$, from Eq (5.8);
- $q^{2/3} \leq n^6$, from the F5 algorithm analysis;

- $q^6 \geq 2^\lambda$ for $n = 10$, and $q^{\frac{n}{2}} \geq 2^\lambda$ for $n > 10$ even, from Table 5.3.

5.6 Parameters

Given a security parameter λ, the aim is to find parameters n, q, M and K that minimise the signature size. Using the analysis done in Section 5.5, the best known attack to sATFE must have a running time greater than 2^λ. The analysis reported in [131, 17] has been also used for the choice of parameters q and n.
The size in bits of the (linkable) ring signature, with respect to parameters n, q, M and K, is given by the following values:

- the secret key sk, an invertible matrix A with coefficients in $\mathbb{F}_q$ represented by $n^2\lceil\log_2 q\rceil$ bits;
- the public key pk, an alternating trilinear form which can hence be stored with $\binom{n}{3}\lceil\log_2 q\rceil$ bits;
- the signature length, given by $|\mathsf{salt}| + |\mathsf{tag}| + |\mathsf{ch}| + K\left|\mathsf{resp}_{\mathsf{ch}_i=0}\right| + \left|\mathsf{resp}_{\mathsf{ch}_i=1}\right|$.

Then:

- the length of the salt is taken as the double of λ: $|\mathsf{salt}| = 2\lambda$;
- a tag is an alternating trilinear form: $|\mathsf{tag}| = \binom{n}{3}\lceil\log_2 q\rceil$;
- the challenge is a positive integer smaller than $\binom{M}{K}$: $|\mathsf{ch}| = \left\lceil\log_2\binom{M}{K}\right\rceil$;
- whenever the challenge is 0, the response contains an invertible $n \times n$ matrix D, a path in the Merkle tree with R leaves, where R is the size of the ring, and λ random bits r, hence

$$\left|\mathsf{resp}_{\mathsf{ch}_i=0}\right| = |D| + |\mathsf{path}| + |r| = n^2\lceil\log_2 q\rceil + 2\lambda\lceil\log_2 R\rceil + \lambda;$$

- whenever the challenge is 1, the response is equal to the set of internal nodes of the seed tree needed to obtain the associated leaves. The number of such nodes is studied in Subsection 5.3.2 and different approaches can be followed: minimising the average, the best case or the worst case. The best choice of the parameters should not be affected by which approach has been chosen. For this reason, the worst case is reported here:

$$\left|\mathsf{resp}_{\mathsf{ch}_i=1}\right| = \lambda\left(K\lceil\log_2 M\rceil - K\lceil\log_2 K\rceil + 2^{\lceil\log_2 K\rceil} - K\right).$$

Hence, the (non-linkable) signature has a bit length of

$$\begin{aligned}2\lambda + \left\lceil \log_2 \binom{M}{K} \right\rceil &+ K\left(n^2\lceil\log_2 q\rceil + 2\lambda\lceil\log_2 R\rceil + \lambda\right) \\ &+ \lambda\left(K\lceil\log_2 M\rceil - K\lceil\log_2 K\rceil + 2^{\lceil\log_2 K\rceil} - K\right).\end{aligned} \tag{5.10}$$

Observe that, in the case of a linkable ring signature, the size of a tag $\binom{n}{3}\lceil\log_2 q\rceil$ in the above equation is added. Given a security parameter λ, values for n, q, M and K minimizing Eq. (5.10) have been found such that they match the security required. Since $n = 9$ is broken [17], $n = 10$ has been chosen with the requirement that the origin ϕ generated in the Setup algorithm does not fall in the fraction of weak keys reported in [17]. Since parameters M and K contribute as $\left\lceil \log_2 \binom{M}{K} \right\rceil$ ($\sim \lambda$) and only K is involved linearly in Eq. (5.10), the number of repetitions M is selected to be not too high, to avoid a slowdown in the performance.
Proposed parameters and signature lengths for $\lambda = 128$ bits of security are reported in Table 5.4. The approach behind the second set of parameters is conservative, taking into account the conjecture from [17, Sect. 5]. Observe that the sizes refer to the non-linkable ring signature scheme: if the linkability is needed, the tag must be included in the signature, adding $\binom{n}{3}\lceil\log_2 q\rceil$ bits. Concretely, for the first set of parameters and for a ring of cardinality R, the upper bound on the signature length consists of a fixed part of 8.4 KB, plus a variable part, depending logarithmically on the size of the ring, of $0.7\lceil\log_2 R\rceil$ KB. Moreover, the average length is given by $8.1 + 0.7\lceil\log_2 R\rceil$ KB. The dimension of the public key is 330 Bytes, while the private key is 275 Bytes. Alternatively, since the secret key consists of a random invertible matrix, it can be generated expanding a λ bit seed, hence its size can be reduced to λ bits. For $\lambda = 128$, a secret key has weight 0.016 KB.

λ	q	n	M	K	R (size of the ring) 2^1	2^3	2^6	2^{12}
128	$2^{22}-3$	10	512	23	8.3 ± 0.8	9.8 ± 0.8	12.0 ± 0.8	16.4 ± 0.8
128	$2^{22}-3$	12	512	23	11.1 ± 0.8	12.6 ± 0.8	14.8 ± 0.8	19.2 ± 0.8

Table 5.4 Parameters and signature sizes in KB of the non-linkable ring signature.

Chapter 6

Bitcoin: a new Proof-of-Work system with reduced variance

The result presented in this chapter is a joint work with Danilo Bazzanella, and it can be found in [10].

Proof-of-Work (PoW) is a consensus protocol that guarantees high security (see Chapter 3 for more details). A user would have to have more computing power than the rest of the network to be able to attack it, the so-called *51% attack*. The drawback is that it is very complicated to add a block to the blockchain without a major investment in adequate equipment, such as Application Specific Integrated Circuits. This greatly discourages autonomous mining and pushes all users to join a mining pool, paying fees to reduce the business risk.
One problem with this natural tendency is that the five largest pools together currently account for more than 80% of the computing power of the entire network, which makes the network very poorly decentralised. These data were taken on December 1, 2022 from [89], a daily updated resource to consult this kind of data.
Another problem stems from the fact that, in Bitcoin, the miner (or pool) that inserts the block takes the whole reward (that is, 6.25 BTC as of 2023), while all the other miners take nothing. This makes the variance in miners' profit very high and the only way to remedy to this effect is to be a huge miner, with the associated investments, or to work for a pool.
Furthermore, in Bitcoin the insertion time of a block has a high variance. It is easy to verify that only 7% of blocks are produced between 9 and 11 minutes, while only

14% of blocks are added to the chain between 8 and 12 minutes. It has been shown that the high variance of block insertion time is at the root of critical attacks on PoW blockchains [21].

The idea of a more equitable distribution of the reward for successfully inserting a block is not new. Sarkar [120] introduced in 2019 the *Multi-stage Proof-of-Work*: the proof-of-work for mining a block is divided into multiple stages, and this has the effect to split the block reward into an equal number of stage rewards. Once a block gets onto the blockchain, the miner which successfully completed a particular stage can claim the reward for that stage. These types of protocols have been analysed in great detail in [44]. They proved that the mining probability might not be strictly related to the miner hashing power, and this could open up potential fairness issues in mining. In addition, some attacks have been proposed which appear to increase the vulnerability of the system.
A lot of work has also been done to discourage the creation of large Bitcoin mining pools. A line of research is trying to disincentivize large pools making it difficult to delegate the Proof-of-Work. Eyal and Sirer [57] proposed the *Two-Phase PoW*, that consists of two separate cryptopuzzles. The first one is identical to the one already existing on Bitcoin, so the miner must find a nonce such that the hash of the block header is less than a set target. In the second part, the block header must be signed with the coinbase transaction's private key, and the hash of this digital signature must be below a second difficulty parameter. Miller, Kosba, Katz and Shi [101] proposed to disable mining pool enforcement mechanisms in a cryptographically strong manner, through *strongly nonoutsourceable puzzles*.
A very interesting proposal, even if only theoretical, is the one proposed by Shi in [125]. The idea is to find a problem with several solutions, rather than a single solution. The first miner (or pool) that finds all the solutions will cause the block to be inserted. The block will be inserted by one of the miners at random among those who have found at least one solution.
This is similar to the line taken by the Bobtail algorithm, suggested by Bissias and Levine in 2020 [21]. Bobtail is a possible concrete realization of Shi's model, but it is rather complex and difficult to manage. They proposed to maintain a public ranking of the lowest k hashes in the network. The block is inserted by whoever is at the top of the ranking (i.e. has found the lowest hash), when the average of these k values is less than a certain target t_k. If $k = 1$, this is equivalent to the current Bitcoin, and the variance of interblock times decreases as k increases. One of the advantages

of this idea is that they have a number of forks comparable to that of the current Bitcoin protocol, even if $k > 1$. Conversely, the great disadvantage is that identifying a single miner with the responsibility for entering the next block carries the risk of blocking the system, in case the miner identified by the protocol is malicious and refuses to proceed.

The original result. This chapter introduces a variant of Proof-of-Work that improves on Shi's idea and can be easily implemented in practice. In order to insert a block, the network must not find a single nonce, but must find a few, let's say j, of them. This simple change allows for a fairer distribution of rewards and at the same time has the effect of regularizing the insertion time of blocks.
The proposed model is similar to Bobtail, but it is simpler, the amount of communications is smaller, and it does not suffer from the problem of system blocking. Due to the greater simplicity of the protocol, it can be mathematically proved that, while the expected value of miners' profit remains unchanged compared to Bitcoin, its variance decreases, and it decreases more than Bobtail does. On the other hand, the probability of having forks increases, but luckily this does not create any significant issue.

6.1 New Proof-of-Work systems

Currently, the right to insert a new block into the Bitcoin blockchain, and thus to receive the reward, is given to the miner (or pool) that first finds a nonce that, when inserted into the block, returns a hash of the block header that is less than a certain target. The key idea is to modify Bitcoin's proof-of-work protocol by searching for multiple nonces, each of which, when inserted into the producer block, returns a hash of the header that is less than the target. Each miner will have the opportunity to find multiple nonces and thus to be rewarded in proportion to the number of nonces found.
More precisely, one can set an integer parameter $j \geq 1$ and establish that a new block B is inserted when the entire network finds j nonces. Note that the case $j = 1$ is essentially the current Bitcoin protocol, but with a small difference: in fact, since it is in general necessary to reward multiple miners, it is not possible to do so with the coinbase transaction of the block B; indeed, while proceeding with the PoW, it is not

yet known which miners will deserve a fraction of the block reward.
Every nonce found must be published and verified by the network, and it must be inextricably linked to the Bitcoin address of the miner that found them. The coinbase transaction of the block $B+1$ will then redistribute the rewards among the miners who found the nonces during the creation of the block B.
To maintain the desired block speed, the current Bitcoin game is switched to a repeated game that is j times easier, but the block is inserted when the network is successful j times.
To link a nonce with the address of the miner who found it, a hash function h, e.g. SHA256 [111] already used by Bitcoin, can be utilised. The miners will look for a particular value nonce_1 to calculate another value, nonce_2, with the following formula:

$$\text{nonce}_2 = h(\text{miner address} \mid\mid \text{nonce}_1).$$

nonce_2 is the value that, when inserted into the block header, will return a digest that is below the set target.
When a miner finds a suitable nonce_2, she will publish their block, the value nonce_1, and their Bitcoin address. The network is then able to verify that the PoW has been correctly executed. Since every hash function is collision resistant, it is expected that another miner will not be able to find another nonce_1 which, concatenated with his address, will return the same value nonce_2.

The model can be summarised as follows:

1. When a miner finds a correct nonce_2, she publishes the value nonce_1, the block header, and the Bitcoin address where she will receive the reward. The network can easily verify the correctness of the published nonces;

2. When the number of nonces published by the entire network equals j, the nonces are sorted and the miner who owns the smallest nonce_2 gets the right to insert the new block. To prove possession of the address, the miner must use his digital signature. Rewards will be distributed in the next block;

3. The network starts to work for the insertion of a new block. The miners can insert the transactions they prefer, but the coinbase transaction will be fixed by the protocol and every miner is required to use the same. This shared coinbase distributes the reward plus the fees in proportion to the number of nonces found in the mining of the last inserted block;

4. The process starts again from step 1.

The easiest way to switch from the current Bitcoin protocol to the new protocol is to establish that a single block is entered without the coinbase transaction, postponing the reward distribution to the next block.
This system selects a single miner to be the only one able to insert a block and this opens the way for the Denial of Service attack, already introduced in Bobtail [21]. However, this is not a issue. If the miner who has the right to insert the next block, i.e. the one with the smallest nonce$_2$, does not proceed, the network continues to perform calculations looking for a new nonce$_2$. A few moments later (on average, $\frac{10}{j}$ minutes) the network will find a new nonce, and at that point there will be two miners able to insert the block. In fact, there will be $j+1$ nonce$_2$ that are valid, and thus the two miners who found the two smaller nonces both have the right to insert the new block. Any malicious miner who tries to stop the system by not inserting the block, when he is the only one with this right, therefore risks losing his reward shortly thereafter, as soon as there is a second miner who acquires this right.

Hence, for each choice of j, the system has the same average block insertion time rate as the current Bitcoin protocol, but the variance of the insertion time is smaller and decreases as j increases. It is also more likely that a block will be inserted in a time close to the average. See Section 6.2 for proofs about these facts.
The second advantage of this modified PoW model is that miners who do not insert a block, but who can still show that they worked for the system, are rewarded accordingly. The reward for each block is divided among the miners in proportion to the number of nonces they have found.
A possible drawback are the forks. If two miners find the j-th nonce almost simultaneously, there are two users (the ones with the two smallest nonces) that are able to propose the insertion of their block to the network. The network is then left with two different last blocks (a fork). Each miner then resumes mining by choosing which branch of the network to try and to attach a block to. In a short time, one of the two branches prevails over the other and the blockchain resolves the fork. The value of j must therefore be optimised so as to reward as many miners as possible, without disproportionately increasing the number of communications and the numbers of forks.
In any case, the number of communications is less than in Bobtail. This happens because any suitable nonce is immediately accepted and transmitted to the network,

whereas in Bobtail nonces are repeatedly substituted and before identifying the winning ones the protocol needs many communications.
Note that this model follows Shi's idea, but making an essential change. Shi proposed to insert the new block once a single miner has reached a fixed number of solutions to the problem. By doing so, however, large miners would have an unfair advantage and would tend to have a reward greater than their weight. The choice of inserting the block when the entire network finds a fixed number j of solutions, and not when a single miner does, allows to avoid this issue and shares the reward more fairly.

6.2 Analysis of the new systems

To model the Bitcoin protocol, a Bernoulli process $\{X_n\}_{n\in\mathbb{N}}$, where X_n are independent and identically distributed Bernoulli random variables of parameter $p = \frac{1}{600H}$ and H is the total hashrate of Bitcoin, can be used. The parameter p has been chosen in this way because on average the Bitcoin blockchain adds a new block every 10 minutes. Hence, $X_n = 1$ means that the network was successful in mining the block in the n-th second. The hash function involved in the Bitcoin consensus protocol is not strictly random, but it is so unpredictable that approximating it with a random variable seems to be the most reasonable solution.
Consider now the random variable S, that models the time of first success of a Bernoulli process:

$$S = \min\{n \in \mathbb{N} : X_n = 1\}.$$

Clearly, S is a geometric random variable of parameter p and its expected value is $\mathbb{E}(S) = \frac{1}{p} = 600\,H$, the average number of hashes required to insert a new block into the Bitcoin blockchain. Since the network processes about H hashes in one second, this average waiting time is about 10 minutes. The variance of S is also straightforward and it is equal to

$$\mathbb{V}(S) = \frac{1-p}{p^2} \sim \frac{1}{p^2},$$

hence the standard deviation of S is of the same order as its expected value.

Similarly, to model the new protocol, define a Bernoulli process $\{Y_n\}_{n\in\mathbb{N}}$, where Y_n are independent and identically distributed Bernoulli random variables of parameter jp and $j < 600\,H$. Let T_j be the random variable that models the time of j-th success of the Bernoulli process $\{Y_n\}_{n\in\mathbb{N}}$:

$$T_j = \min\{n \in \mathbb{N} : Y_1 + \ldots + Y_n = j\}.$$

Notice that T_j is distributed as a negative binomial random variable with parameters (j, jp), where the first parameter models the required number of successes, while the second parameter models the success probability in each Bernoulli trial. A closed formula to compute its expected value and its variance is known:

$$\mathbb{E}(T_j) = j\,\frac{1}{jp} = 600\,H, \qquad \mathbb{V}(T_j) = j\,\frac{1-jp}{(jp)^2} = \frac{600^2\,H^2}{j} - 600\,H.$$

(Observe that we are counting how many trials are needed to get j successes, and not the number of failures before j successes occur, which is usually how a negative binomial variable is defined; in the second case, the expected value would equal $\frac{j(1-jp)}{jp}$).
Therefore, the new protocol has an average waiting time for inserting a block which is identical to that of the current Bitcoin protocol, while the variance of this waiting time is smaller for each $j \geq 2$ and decreases as j increases.
The probability of inserting a block between 9 and 11 minutes ($540H$ and $660H$ hashes, respectively) can be computed in the actual Bitcoin protocol and then compared with the same probability of the models for some fixed j values, such as $j = 10$ or $j = 100$:

$$\mathbb{P}[540\,H < S < 660\,H] \sim 7.31\%,$$

$$\mathbb{P}[540\,H < T_{10} < 660\,H] \sim 24.69\%,$$

$$\mathbb{P}[540\,H < T_{100} < 660\,H] \sim 72.36\%.$$

Switching to the generalised protocol would therefore make it rarer for a block to be inserted in a much smaller or much larger time than the expected average time of 10 minutes. This considerable reduction in the variance of the block insertion time would significantly improve the security of the system, since it is well known that the high variance of block insertion time is at the root of critical attacks on PoW blockchains [21].

Also, it has been calculated how the profit is distributed among different miners in the new protocol compared to the current Bitcoin protocol. Let q be the probability

of a generic Bitcoin miner to find a suitable nonce such that the hash of the block header is less than the network target. Note that this probability does not depend on the target at all, but only on the miner's weight (i.e., the ratio of his hashrate to the network's hashrate). A reasonable estimate of this probability can be calculated from past block insertion statistics. Let $M = 52560$ be the average number of blocks mined in a year. Define now the Bernoulli process $\{B_k\}_{k\in\mathbb{N}}$, where B_k are independent and identically distributed Bernoulli random variables of parameter q, that models the probability of adding a block for the considered miner.
Let

$$U = 6.25 \sum_{k=1}^{M} B_k$$

be the profit of the miner in a year. It is clear that

$$\mathbb{E}(B_k) = q,\ \ \mathbb{V}(B_k) = q(1-q),$$

$$\mathbb{E}(U) = 6.25\, M\, q,\ \ \mathbb{V}(U) = (6.25)^2\, M q(1-q).$$

On the other side, define a Bernoulli process $\{C_i\}_{i\in\mathbb{N}}$, where C_i are independent and identically distributed Bernoulli random variables of parameter q, and a Binomial random variable N_k

$$N_k = \sum_{i=1}^{j} C_i,$$

that counts the number of nonces found for the block k by the considered miner. Finally, let V be the random variable defined as

$$V = 6.25 \sum_{k=1}^{M} \frac{N_k}{j},$$

that models the profit of the miner in a year in the new protocol. Thus,

$$\mathbb{E}(N_k) = jq, \qquad \mathbb{V}(N_k) = jq(1-q),$$

$$\mathbb{E}(V) = 6.25\, M\, q, \qquad \mathbb{V}(V) = (6.25)^2\, M\, \frac{q(1-q)}{j},$$

i.e. the profit of each miner has the same expected value as in the current Bitcoin protocol, while the variance of this profit is always lower and decreases by a factor $\frac{1}{j}$.

Note that, in the Bobtail protocol, the decrease in variance occurs through a multiplicative factor close to $\frac{4}{3k}$ [21]. Since their parameter k is the analogue of the parameter j, the decrease in variance in the presented protocol is always greater for all values of the parameter.

To evaluate in a further way the improvement that would be made with the new system, it may be convenient to estimate the probability for a miner of weight p to obtain an annual profit not too much lower than the expected value. For example, in Bitcoin, a miner has a greater than 95% chance of making an annual profit U greater than 70% of the expected profit if it holds a weight equal to 0.0005304, that is

$$\mathbb{P}[U > 0.7\,\mathbb{E}(U)] \geq 0.95 \qquad \text{if } p = 0.0005304.$$

The same estimation can be repeated for the new model: for example, if $j = 10$, to get the same enterprise risk it is sufficient for a miner to have weight $p = 0.00005305$, i.e.

$$\mathbb{P}[V > 0.7\,\mathbb{E}(V)] \geq 0.95 \qquad \text{if } p = 0.00005305.$$

More generally, the weight a miner has on Bitcoin can be divided by j on the new protocol, so as to maintain essentially the same probability of making a suitably high profit.

Thus, in the new system, one could create a mining company with the same business risk but with a lower initial investment by one order of magnitude, in the case $j = 10$, or by two orders of magnitude, in the case $j = 100$.

6.3 Blockchain forks

Decker and Wattenhofer [46] have done an excellent study on the number of forks in Bitcoin, estimating it through a mathematical model and verifying it with the actually observed blockchain fork rate. To better match their model with past observed data, they estimated the probability of having a fork in Bitcoin with

$$p_1 = \mathbb{P}[\text{have a fork}] = 1 - \left(1 - \frac{1}{633.68}\right)^{11.37} \approx 1.78\%,$$

justifying the presence of 633.68 instead of the expected value of 600 by the decrease in the computational power of the network during the period under consideration. For a better fit of the model to the average situation, the analysis described in this chapter will be conducted with 600 instead.

To estimate the average probability of having a fork of length two, that is, a fork with two blocks applied in both branches, it is necessary to take into account the size of the two parts of the network that work in competition to attach the blocks to the two different ends of the fork.
Let x be the rate of the network that works to attach a block to the first end of the fork, and consequently let $1-x$ be the rate of the network that works to attach a block to the second end of the fork.

When the network produces a second fork following the first, there are three possible cases:

- Case 1: the two blocks of the new fork are both attached to the first end of the previous fork;
- Case 2: the two blocks of the new fork are both attached to the second end of the previous fork;
- Case 3: one block from the new fork is attached to the first end, while the other block is attached to the second end.

A fork of length 2 is created only in the third case. In fact, in the first two cases the fork remains of length 1. For the Bitcoin protocol, these three probabilities are:

$$\mathbb{P}[\text{case 1}] = x^2 p_1,$$

$$\mathbb{P}[\text{case 2}] = (1-x)^2 p_1,$$

$$\mathbb{P}[\text{case 3}] = 2x(1-x)p_1.$$

Note that the above probability estimates were made assuming that the probability of a node finding a block is uniformly distributed at random among all nodes, as assumed by Decker and Wattenhofer [46].

Then, the probability to have in Bitcoin a fork of length 2 is

$$\mathbb{P}[\text{fork of length 2}] = \int_0^1 p_1 2x(1-x) p_1 dx = \frac{p_1^2}{3}.$$

In the same way, it can be shown that, on average,

$$\mathbb{P}[\text{fork of length } k] = \frac{p_1^k}{3^{k-1}}.$$

In particular,

$$\mathbb{P}[\text{fork of length 6}] = \frac{p_1^6}{3^5} \approx 1.8 \times 10^{-13}.$$

A Bitcoin transaction is generally considered secure when it is on a block with six confirmations, that is, a block to which six other blocks on the blockchain have been attached. The probability of such a transaction being voided due to a fork is negligible.

Therefore, following the same strategy, the probability of having a fork in the new protocol may be estimated as

$$p_j = \mathbb{P}[\text{have a fork}] = 1 - \left(1 - \frac{j}{600}\right)^{11.37},$$

and then

$$\mathbb{P}[\text{fork of length } k] = \frac{(p_j)^k}{3^{k-1}},$$

as the parameter j varies. Hence, it follows that

$$\mathbb{P}[\text{fork of length 11 and } j = 10] = \frac{(p_{10})^{11}}{3^{10}} \approx 7.4 \times 10^{-14},$$

which implies that, if $j = 10$ and the network waits for 11 confirmations, the new system is safer than the classic Bitcoin with 6 confirmations. In the same way,

$$\mathbb{P}[\text{fork of length 24 and } j = 100] = \frac{(p_{100})^{24}}{3^{23}} \approx 1.2 \times 10^{-13},$$

which implies that, if $j = 100$ and the network waits for 24 confirmations, the new system is safer than the classic Bitcoin with 6 confirmations.
This shows that the higher average fork probability is not an issue at all, because for

real-time payments it is not suitable, but neither is Bitcoin, and for payments where it is sufficient to fix the day of the transaction it does not matter if you have a four hour delay instead of a one hour delay.
It should be noted that this section does not consider forks involving three or more blocks competing to be the new block of the blockchain because this eventuality has a negligible probability. For further details, see Figure 6.4 in the next section.

6.4 Numerical simulations

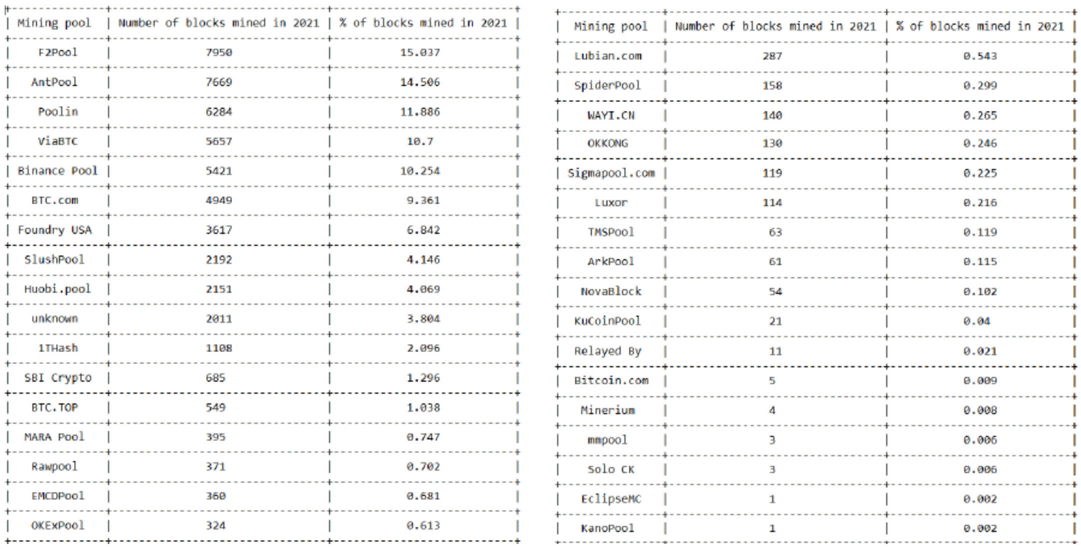
This section shows some plots that report how the results obtained and described in theory are also confirmed in practice. The protocol has been simulated for different values of j (in particular $j = 1, 5, 10, 25, 50$ and 100) and then the obtained results are compared with the real data of the Bitcoin blockchain during the year 2021. Keep in mind that the model with $j = 1$ is basically equivalent to the current Bitcoin protocol.
To simulate the data, it was checked who mined all the blocks during the year 2021 and then assigned to each mining pool or single miner the corresponding computing power, that is each miner i has a chance to insert a new block equal to

$$p_i = \frac{\text{Number of blocks mined by } i \text{ in 2021}}{\text{Total number of blocks mined in 2021}}.$$

It turned out that 52868 blocks were mined, by 34 different mining pools or single miners, with the distribution indicated in Figure 6.1.

First of all, the expected annual profit for each mining pool has been simulated. As depicted from Figure 6.2, whatever j is, the expected profit is the same.

In the second simulation, the aim is to show that the annual profit of each miner is closer to the expected profit if j increases, that is the variance of the profit becomes lower if j grows. Denote as $\{x_1, \dots, x_{34}\}$ the profit of each miner during the year 2021. For each chosen j value, the network has been simulated 10 times, obtaining for each simulation profit values $\{\hat{x}_1, \dots, \hat{x}_{34}\}$. Then, for each miner or mining pool, the *discrepancy* MSD has been computed, that is the mean of the square of the



Mining pool	Number of blocks mined in 2021	% of blocks mined in 2021
F2Pool	7950	15.037
AntPool	7669	14.506
Poolin	6284	11.886
ViaBTC	5657	10.7
Binance Pool	5421	10.254
BTC.com	4949	9.361
Foundry USA	3617	6.842
SlushPool	2192	4.146
Huobi.pool	2151	4.069
unknown	2011	3.804
1THash	1108	2.096
SBI Crypto	685	1.296
BTC.TOP	549	1.038
MARA Pool	395	0.747
Rawpool	371	0.702
EMCDPool	360	0.681
OKExPool	324	0.613

Mining pool	Number of blocks mined in 2021	% of blocks mined in 2021
Lubian.com	287	0.543
SpiderPool	158	0.299
WAYI.CN	140	0.265
OKKONG	130	0.246
Sigmapool.com	119	0.225
Luxor	114	0.216
TMSPool	63	0.119
ArkPool	61	0.115
NovaBlock	54	0.102
KuCoinPool	21	0.04
Relayed By	11	0.021
Bitcoin.com	5	0.009
Minerium	4	0.008
mmpool	3	0.006
Solo CK	3	0.006
EclipseMC	1	0.002
KanoPool	1	0.002

Fig. 6.1 Mining pools active during the year 2021

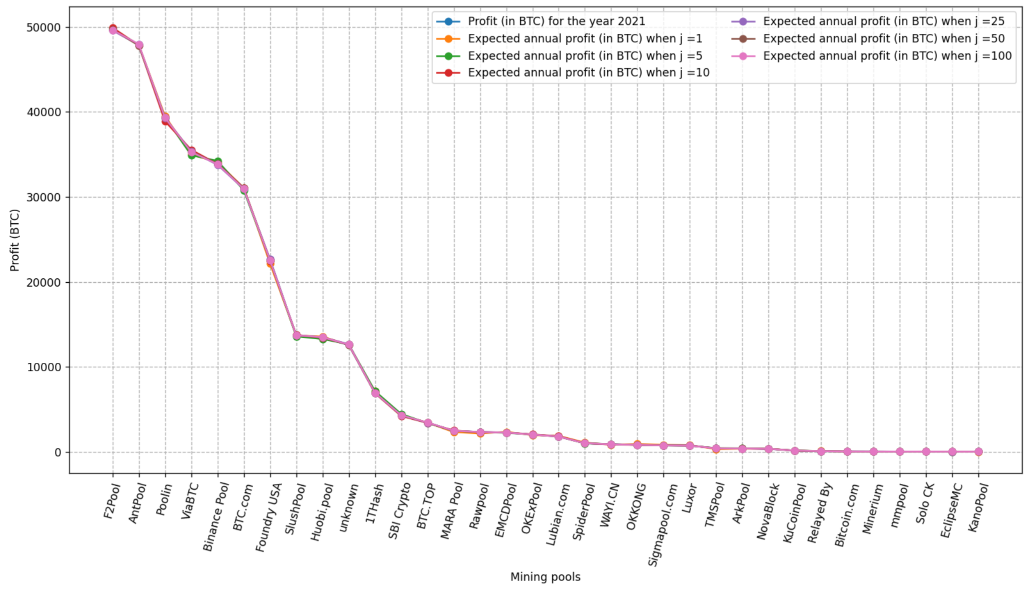


Fig. 6.2 Expected annual profit for different values of j

differences between the real and the simulated data, in the following way:

$$\mathrm{MSD}_i = \sqrt{\frac{1}{34}\sum_{k=1}^{34}(x_k - \hat{x}_k)^2} \;\; \forall i \in \{1, 10\}.$$

Finally, these numbers have been used to draw the boxplots for each value of j (see Figure 6.3). A boxplot is a standard method for graphically showing the spread of numerical data through their quartiles. It is represented by a box divided into two parts, from which two segments emerge. The box is bounded by the first and third quartiles, and divided within it by the median. The segments are instead bounded by the minimum and maximum values. Figure 6.3 shows that the estimated profit is

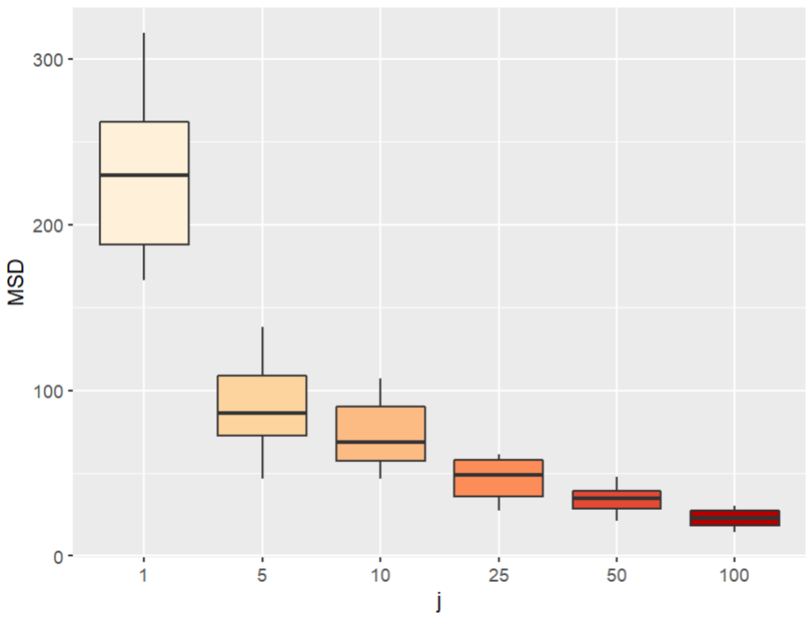


Fig. 6.3 Decrease in variance for different values of j

closer to the expected average profit if the parameter j increases, as expected. Finally, the last simulation (Figure 6.4) shows the expected number of forks for the new protocol. Notice that, up to $j = 25$, the number of forks involving three or more miners is negligible. Hence, even if the number of forks is obviously increasing, the assumptions and analysis done in Section 6.3 stay valid.

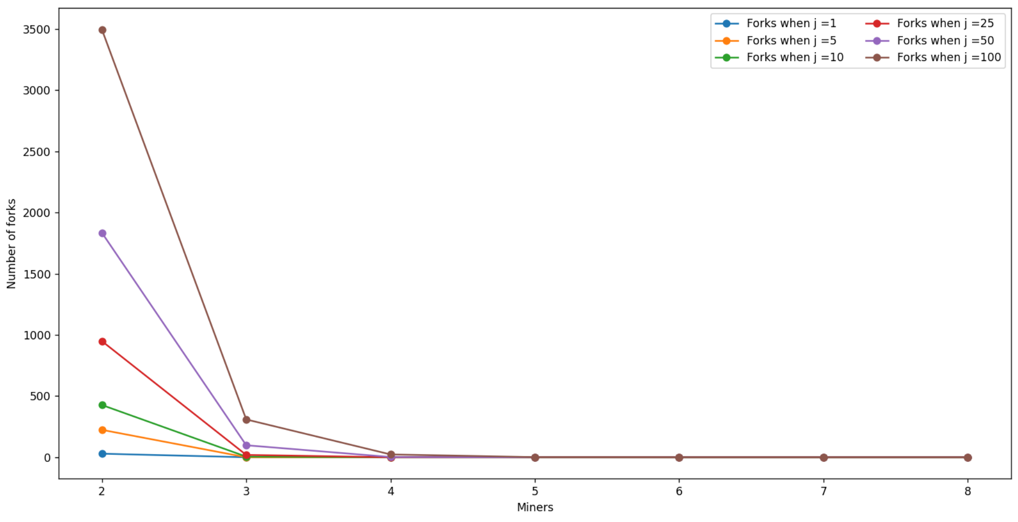


Fig. 6.4 Number of forks involving two or more miners for different values of j

Chapter 7

Towards a Privacy-preserving Dispute Resolution Protocol on Ethereum

The result presented in this chapter can be found in [59].

Blockchains were introduced in 2008 by Satoshi Nakamoto and, since that date, the technology has literally exploded. There are mainly two types of blockchains: blockchains whose cryptocurrency represents a medium of exchange, such as Bitcoin [103], and blockchains whose cryptocurrency has more of a utility function, as is the case, for instance, on Ethereum [33] or Polkadot [139]. Blockchains belonging to this second category allow *smart contracts* to be written, i.e. computer programs that perform deterministic functions.
The transfer of cryptocurrency from one address A to another address B is guaranteed by the use of cryptographic primitives such as hash functions and digital signatures (see Chapter 2 for more details), while the code of a smart contract will always work as expected, barring bugs in the code. Therefore, within such an environment, it is unusual to expect conflicts between users, since the entire functioning of the platform is handled by mathematics.
However, the ever-increasing use of this technology has led to disputes that cannot be resolved by the courts or traditional dispute resolution systems. In fact, in the last years more and more applications are emerging, both centralized and decentralized, with increasingly specific goals, mixing the classic internet (Web2) and the new blockchain-enhanced (Web3) worlds. For example, companies such as Binance [20] allow users to buy tokens or cryptocurrencies using fiat currency, while a dApp

such as *Proof of Humanity* (PoH)[113] is a system that creates a Sybil-proof list of humans. Mixing two worlds as different as Web2 and Web3 can lead to disputes of various kinds: for instance, Binance must be sure that all tradable tokens on their site are related to valid blockchain projects, whereas the Proof of Humanity protocol must only add “real" humans to its registry, not bots or fake accounts. Indeed, if Binance were to allow the trading of a token relating to a project that turns out to be a scam, or if PoH were to allow bots to register with the service, there would be undesirable side effects. For this reason, third parties must be able to initiate a dispute if they believe that something within the application is not working as intended.

Several dApps that specifically aim to resolve these new conflicts have sprung up. Recently, various blockchain dispute resolution applications have been emerged. The three most important ones are *Kleros* [87, 75], *Aragon* [41, 4] and *Jur* [74].
Kleros is the most active dispute resolution platform on the Ethereum blockchain. Its service is active since 2019 and it has already solved more than 1500 disputes. Judges are randomly drawn and then asked to vote among different options: the voting system implemented is the *plurality system*. The winning option coincides with the option that receives the most votes. Examples of disputes that have been effectively resolved by Kleros include the following areas: curated lists, escrow, insurance, token listings, and some minor areas such as social networks or Gitcoin grants [76–81].
Aragon [41, 4] is a dApp that enables the development and maintenance of decentralized organizations running on the Ethereum Virtual Machine. The Aragon Network is an internal organisation responsible for providing the infrastructure and a range of services to support users of the platform. Aragon token holders can access to the Aragon services, and one of these services is the Aragon Court, which tries to solve disputes arising from the Aragon DAOs through a crowdsourcing method which works exactly like the Kleros one. The Aragon court solved less than 50 disputes during its life cycle.
Finally, Jur [74] is a legal technology company that aims to create a legal ecosystem for the management of contractual relations. They are having a transition phase, switching from the VeChainThor blockchain [136] to Polkadot [139]. They still did not solve any dispute, but are one of the most promising projects. They plan to use three different arbitration protocols for dispute resolution, depending on the level of severity of the conflict in question.

All the active services are *crowdsourced* arbitration protocols: this model consists of submitting the dispute to a randomly drawn group of (possibly) untrained volunteer jurors which try to find a fair solution in the least possible time. They use a game-theory model known as *Schelling game* [121]. The key point behind this idea is that a large group of independent agents with different opinions holds a collective knowledge that achieves better results in decision-making and problem-solving than individual experts. Unlike professional arbitration, which guarantees quality judgments due to the reputation of arbitrators, crowdsourced protocols are based on a combination of game theory and crypto-economic system of rewards and penalties that incentivize jurors to vote fairly. As a result, the entire legal process can be modeled as a Schelling game, with a Schelling point corresponding to the decision considered fairest by the majority.
Arbitrators have to deposit some amount of token in a stake, and if a dispute happens they can be selected with a probability proportional to the amount they have put in that stake. The same judge can be selected more than once, giving to them more decisional power. The selected arbiters will then vote the option they repute to be the correct one, and at the end the option that has been chosen by the majority of arbiters will be enforced. The judges that chose that option receive an economical reward, while the stake of the other judges will be slashed.
As a strength, this protocol can potentially be implemented on any blockchain with smart contracts, it allows disputes to be resolved much faster than traditional ones, and it allows anyone to vote, thus fully conveying the sense of “freedom" and “decentralization" that a blockchain should offer. In addition, there is the certainty that the dispute will be resolved no matter what happens. However, the latter can also be considered a disadvantage, as the users must accept the decision made by Schelling’s game mechanism. Two other potential problems are the lack of a system to prevent corruption attacks, i.e. judges may collude and agree to favor one party over the other, and only partial resistance to Sybil attacks, that is the same judge can register on the platform more than once, using fake identities.

Many other blockchain dispute resolution projects failed or were just theorised. Examples are given by Sagewise, Oath, Juris (for all of them, see [99]) and Aspera [6]. Aspera was one of the first ideas designed to provide a dispute resolution service through mediation. However, the project foundered partly because of the complexity of the design, which was largely based on machine learning and artificial intelligence, and partly because at the moment mediation does not yet seem to be really necessary on the blockchain, as most disputes are handled by Kleros through a simpler arbitration service.

The original result. This chapter proposes a new idea for dispute resolution on the Ethereum blockchain. The process is divided into two phases. Whenever a conflict arises, a list of qualified judges, registered within the application, can try to resolve the dispute. They will have the opportunity to read the reason for the conflict; then, they can express a preference, through *quadratic voting*, as to which party they believe is more right, as well as propose an agreement that would end the dispute. In the second phase, the users will vote on their favorite judges' proposals using quadratic voting again, and the number of voice credits they can spend will be equal to the votes received during the first phase. Finally, the proposal that received the most votes will be the winning one and, hence, applied. In both phases, who votes will not be able to know the preference of other participants until the end, and the judges do not know who are the others involved in the process.

7.1 The new Dispute Resolution Protocol

This section describes in detail the new dispute resolution mechanism. The aim is to utilise the positive aspects of Kleros and Aragon, while trying to improve on the shortcomings already highlighted above. In particular, the protocol aims to:

- let the users have the final say on the conflict, i.e. they should not be forced to accept the judges' decisions;
- make it unnecessary to own a stake of tokens to express an opinion on a dispute, as not everyone can afford it;
- make it impossible for both users and judges to change their opinions after a certain time, and at the same time make it impossible for different judges to collude;
- assign governance of the dApp not to the users who possess a high number of tokens, but to those who have built their reputation over time, resolving disputes and contributing to the development of the platform itself.

The dispute resolution process can be divided into two phases:

- *Phase 1*: after judges are getting notified about a dispute, they will send a message to a certain smart contract, containing a vote and a possible solution to the problem. At the end of this process, the MACI coordinator does the tally of the votes and gives a score to each user involved in the dispute. Thanks to the properties of MACI (see Chapter 4, Section 4.2.2 for more details), users are not able to know the individual scores given to them by each judge;
- *Phase 2*: the users will be able to vote on their favourite solutions to the dispute. The votes they have received during the first phase represent the total of the voice credits they can spend. At the end, the proposal that has received the biggest preference will be enforced.

There must also be a setup phase where judges register on the platform and associate their identity with some *soulbound tokens* (SBTs) that certify their field of knowledge.

7.1.1 Setup Phase: judges' registration

To participate, judges must be registered on the dispute resolution application. They must be part of a *Semaphore group* (see Chapter 4, Section 4.2.1, for more details), which guarantees that each user is connected to a real identity. Moreover, in order to certify their areas of knowledge (e.g., “law", “maths", “sports", etc.), judges will have to link various souldbound tokens to their digital identity.
If the registration is successful, each judge i will receive a secret/public key pair $(\mathsf{sk}_i, \mathsf{pk}_i)$. This public key is then also registered to the MACI smart contract SC_1: this step is essential to ensure the judges do not collude. This last step also assigns to each judge a fixed number V of voice credits, which will be used to score the users involved in the dispute.

7.1.2 Phase 1: voting and proposals by the judges

Suppose now that a dispute involving n different users arises. For simplicity, assume $n = 2$, but the model can be easily generalized for $n > 2$. One of these users

can activate the dispute resolution mechanism by sending to the smart contract a transaction containing a fee f.
That user will also have to list some *tags* indicating the reason for the dispute. The other party will have to send a transaction with the same fee f to join the dispute: non-adherence implies the impossibility of being a winner in the conflict. Note that there is no incentive to start a dispute fraudulently, as in this case the other party can simply join the conflict and the judges will choose that as the winner, penalising the misbehaviour of the party that started the conflict.
If both sides join the dispute, they must send some evidence that explains their side of the conflict. Since storing documents on the blockchain is an expensive operation, one solution can be to store data off-chain, saving within the blockchain only the hash of these data. How to store data off-chain is a secondary issue, which can be left to the users involved: one can choose a centralized solution, such as a cloud server, or a decentralized solution such as the *InterPlanetary File System* (IPFS) protocol [71], or *Web3 storage* [137]. Another solution could be the mechanism introduced by Kleros and explained in the ERC-1497 proposal [54].
Not all judges can participate in dispute resolution: in fact, only those who have the expertise to understand what the dispute is about can vote. This expertise is certified by SBTs: for example, to participate to a dispute whose topic is “maths", a judge must possess one soulbound token that shows its knowledge in that field. Also, for simplicity's sake, suppose that if a dispute arises in a field in which no judge is an expert, then it cannot be resolved.
There is also the need to set two time thresholds t_1, t_2: judges that want to solve the dispute need to start participating before time t_1, and they can express their opinion until time t_2, which represents the end of this first phase. Finally, suppose that a mimimum number of judges m needs to vote, to guarantee some decentralization.
Every judge will vote using its V voice credits and the voting system used is the quadratic voting. However, it must be taken into account that judges with similar backgrounds are likely to have the same opinion, and thus will tend to favor one side over the other. One possible solution is to weight contributions differently, like already explained in Section 4.3.
Here, T_k indicates the number of affiliations (or tags) the judge k has in common with the dispute tags. Let $\Sigma = \{\sigma_1, \dots \sigma_J\}$ be the set containing how many times each dispute tag is shared by a judge, that is each $|\sigma_j|$ counts how many judges share the tag σ_j with the current dispute tags, and $|\Sigma|$ is equal to the number of dispute

tags. Denote with $v_{i,k}$ the score assigned by the judge k to the user i. Then, the score obtained by users A, B may, for example, be equal to

$$V_i = \left(\sum_{j=1}^{|\Sigma|} \sqrt{\sum_{l=1}^{|\sigma_j|} \frac{v_{i,l}}{T_l}} \right)^2 - \sum_{k=1}^{m} v_{i,k} \quad i = A, B.$$

Bear in mind that this weighted voting mechanism is only an example, inspired by Weyl's work [138]; however, depending on the use case, it can be replaced by another voting model.
When the time t_2 is reached, the MACI coordinator makes the tally of the votes: the commit of this tally is saved on-chain. The users will then be able to see the total scores V_A and V_B they obtained, but they will not know the partial vote they received from each judge.
At the end of the first phase, there are three options:

- $V_A = V_B$: both users received the same amount of total votes, that is no one received an advantage for the second phase;
- $V_A > V_B$: the user A received more votes than the user B. This does not mean that A won the dispute, but only that the judges expressed a preference toward that user;
- $V_A < V_B$: this case is symmetrical to the previous one.

7.1.3 Phase 2: users vote on the judges' proposals

Having reached this point, the users involved in the conflict will have the possibility to read all the proposals made by the m judges and vote the ones they prefer. The mechanism used can be the quadratic voting again, and the voice credits they can spend are equal to the scores V_A, V_B they obtained during Phase 1. In this case, users can also assign a *negative vote* - see Section 4.3 - towards the proposals they do not find appealing.
At the end of the process, the proposal that received the higher score is the winning one and it will be enforced.

Keep in mind that, to date, disputes arising from smart contracts are rather limited and are well covered by already active services such as Kleros. However, the new method described here can resolve the same disputes, while ensuring the privacy of the judges and giving everyone a chance to vote, since there is not the need to stake tokens.

7.2 Incentives

At this point, one question appears to be natural: what are the incentives for the judges and the users to use this system? It turns out that, for both actors, there are two kinds of incentives, an *economic* one and a *social* one.
The idea of the social incentive is to reward those who behave honestly and purposefully toward the platform, while penalizing bad behavior. This is possible through a reputation mechanism and by the use of soulbound tokens. Following the paper of Weyl et Al. [138], the ultimate goal is to have the governance of the DAO in the hands of those judges who have spent their time on the proper functioning of the platform, instead of giving it to those who own the ERC-20 tokens, as it is usually the case.

7.2.1 Incentives for users

As for users who have a conflict, it is clear that their main benefit is the resolution of the dispute. They will have to pay, in equal parts, a fee f that at the end of the process will go into the pockets of the judge who made the proposal that is enforced. Thus, users have an *economic incentive*.
Regarding the social incentive, soulbound tokens can be issued to those users who have been cooperative during the dispute process and have complied with the agreement made. They are linked to the wallet of these users, so the whole Ethereum network can see that that particular user behaved in a certain (positive) way.
Similarly, SBTs can be issued to those users who do not comply with the agreement made. For example, suppose the dispute is about removing a token from the pool of those that can be purchased on Binance. The final proposal accepted by the users is to remove this token from the platform within a certain period of time t. If at the passage of this time t the token is still purchasable, an SBT will be associated with the wallet of the one who was in charge of removing it. This SBT has a negative

meaning, because the entire Ethereum blockchain will see that the one managing that given wallet has not fulfilled the agreement made.

7.2.2 Incentives for judges

Unlike Kleros, judges do not need a token stake to participate to a dispute: in fact, they only need to be experts in the field that the conflict indicates. However, a way to incentivize good behavior and especially to penalize bad behavior is still needed, otherwise a judge can simply register and then vote effortlessly each time, effectively offering a disservice to the users. For these reasons, both an economic incentive and a social incentive can again be introduced.
The economic incentive is the fee f required by the users to start the process: it will be given to the judge that suggested the enforced proposal. All the other judges will not gain or lose anything (if gas fees are not considered).
That is why a social incentive based on a reputation mechanism and SBTs can be useful. Nevertheless, it is necessary to find a way to reward or penalize judges based on objective rather than subjective evaluations. For example, a judge who expresses a preference for the user who received fewer votes at the end of the tally has not necessarily voted in bad faith or without paying attention, and for that reason it cannot be penalized. However, judges can be rewarded or penalized based on the dispute resolution proposals they make, i.e., the quality of their job towards the community of the platform. During Phase 2, the users vote on the proposals received using the quadratic voting mechanism. They can give a positive score if they like the proposal received, but at the same time they can give a negative score if the proposal does not satisfy them. The score obtained from the proposals may, for example, be proportionally converted into reputation score and assigned to the judges who made that proposal.
At this point, some fixed thresholds $\varepsilon_- < 0 < \varepsilon_1 < \dots$ can be introduced. If a judge has a reputation score lower than ε_-, she will be removed by the Semaphore group, hence she will not be able to express their opinion on the disputes anymore. Moreover, they will receive a SBT that certifies the bad behaviour towards the application. Notice that she cannot even create a new account and start again, since their identity must be registered to be part of the Semaphore group, e.g. using the Proof of Humanity protocol.
On the other way around, judges with a reputation score higher than ε_1 will receive a

SBT which will certify their honesty and dedication: they will be considered *trusted*. All these SBTs are issued by the application itself.
All the trusted judges have the possibility to contribute towards the governance of the platform. Thus, unlike classic dApps where each token represents a stock stake and the more tokens you own, the more decision-making power you have, here everything is divided proportionally among all the users who have spent their time on the platform to make the service work. Indeed, in an application such as the one described in this chapter there is no need to introduce any native ERC-20 token.

Chapter 8

Special Subsets of Addresses for Blockchains using the secp256k1 Curve

The result presented in this chapter is a joint work with Antonio Di Scala, Giuliano Romeo and Gabriele Vernetti, and it can be found in [47].

As described in Chapter 3, in a blockchain, transaction security is ensured by public key cryptography and, in particular, by the difficulty of solving the discrete logarithm problem. Using the best known algorithms it is practically impossible, in a general case, to recover the private key starting from the knowledge of the public address. For all these reasons, it has been quite unexpected when, in 2020, Sala, Sogiorno and Taufer [118] found the private keys of some existing Bitcoin addresses, being able to spend cryptocurrencies on their behalf. They mounted an attack on a small multiplicative subgroup of a group that is mapped to the group of points of Bitcoin's elliptic curve secp256k1. For this small set, a brute-force attack has been feasible and, against any odds, 4 of those addresses coincided with some actually used in Bitcoin's history.
The same authors left open the problem of understanding the reasons for such pathological behavior and the analysis of other cryptocurrencies, together with other small algebraic structures. A brute-force Bitcoin address generation has been object of study also in [26].

The original result. In this chapter, the analysis is expanded to the inspection of other 7 subsets of the same order, obtained as cosets of the subgroup used in [118]. Furthermore, the same examination is performed for Ethereum addresses and for some other famous cryptocurrencies that share the same elliptic curve: Dogecoin [50], Litecoin [90], Zcash [69], Dash [51], and Bitcoin Cash [22]. This analysis also explores why this particular peculiarity may have arisen in Bitcoin and the potential for conducting a similar investigation on other critical elliptic curves for blockchains, such as Curve25519 and NIST P-256.
Moreover, detailed descriptions are provided of the techniques developed for extracting data from the entire history of these blockchains and executing queries within a feasible timeframe. All programming scripts have been made publicly accessible at [60].

8.1 The small subsets

As described in Chapter 3, the group $E(\mathbb{F}_p)$ of the *secp256k1* curve has prime order

$$\begin{aligned} q =& 115792089237316195423570985008687907852837564279074904 3\\ & 826051631415181614943 37, \end{aligned}$$

and it is then isomorphic to the additive group $(\mathbb{F}_q, +)$. Hence, the private key can be chosen among the non-zero elements of $(\mathbb{F}_q, +)$, which are $q-1$. The set of all private keys is thus in bijection with the multiplicative group $(\mathbb{F}_q^*, \cdot)$, which also has order $q-1$ and it is a cyclic group. Therefore, it is well known from elementary algebra that it has a unique cyclic subgroup of order d for any divisor d of $q-1$. Its factorisation is

$$q - 1 = h \cdot p_1 \cdot p_2 \cdot p_3,$$

where

$$\begin{aligned} h &= 18051648 = 2^6 \cdot 3 \cdot 149 \cdot 631, \\ p_1 &= 107361793816595537, \\ p_2 &= 174723607534414371449, \\ p_3 &= 341948486974166000522343609283189. \end{aligned}$$

In [118], the authors examined the multiplicative subgroup $H \leq \mathbb{F}_q^*$, whose order is

$$|H| = h = 18051648,$$

since it can be fully investigated in a short time. As pointed out in the same paper, it is worth to analyse also some other structures of the same order, for example the cosets of H. Consider, for example, the generator $g = 7$ of the group $(\mathbb{F}_q^*, \cdot)$. In this way, denoted by

$$\begin{aligned} g_0 &= 7^{p_1p_2p_3}, & g_1 &= 7^{hp_2p_3}, & g_2 &= 7^{hp_1p_3}, & g_3 &= 7^{hp_1p_2}, \\ g_4 &= 7^{hp_1}, & g_5 &= 7^{hp_2}, & g_6 &= 7^{hp_3}, & g_7 &= 7^{h}, \end{aligned}$$

then

$$\begin{aligned} |\langle g_0\rangle| &= |H| = h, & |\langle g_1\rangle| &= p_1, & |\langle g_2\rangle| &= p_2, \\ |\langle g_3\rangle| &= p_3, & |\langle g_4\rangle| &= p_2p_3, & |\langle g_5\rangle| &= p_1p_3, \\ |\langle g_4\rangle| &= p_1p_2, & |\langle g_7\rangle| &= p_1p_2p_3. \end{aligned}$$

The cosets investigated are g_iH, for $i \in \{0,\dots,7\}$. Notice that $g_0H = H$, that is the same subgroup considered in [118].

8.2 Experimental environment and development

This section describes the experimental environment used to obtain the results presented in the following sections. To conduct a comprehensive investigation of the existence of the generated addresses, a database containing all addresses that had ever appeared on the blockchains of interest was utilized. The MySQL *Laragon* tool was used for the search operation [86].

8.2.1 Blockchain addresses extraction

The extraction of all the addresses ever appeared on a blockchain can be done in different ways, depending on the time availability.
Generally, the two most common methods are:

- setting up a *full node* (which contains a complete local copy of the relative blockchain) and reading data from it;
- getting data through public Application Programming Interfaces (APIs) (available on the web).

The decision was made to set up full nodes when data retrieval from any public API was unfeasible.

8.2.2 Development

Python scripts were developed for certain blockchains (Dogecoin, Litecoin, Bitcoin Cash, and Ethereum) to utilize the public APIs provided by *Tatum* [133] and *Ankr* [3]. On the other hand, full nodes were set up to analyze Dash and Zcash, while the list of Bitcoin addresses was obtained from [88], a website that provides useful data and statistics about the Bitcoin blockchain. The publicly accessible code can be found at [60].
Table 8.1 presents the total number of addresses examined, which constitutes the entirety of addresses ever appeared in the history of these blockchains, up to May 2022.

Blockchain	Addresses
Bitcoin	923.414.052
Ethereum	144.596.346
Dogecoin	69.817.509
Litecoin	134.530.241
Dash	92.456.113
Zcash	6.813.058
Bitcoin Cash	334.965.092

Table 8.1 Total number of blockchain addresses, up to May 2022.

8.3 Cosets examination

This section lists the obtained results from inspecting the eight cosets g_iH, $i \in \{0,\dots,7\}$, of order $h = 18051648$ for the seven investigated blockchains. The eight cosets contain a total of $8h \approx 144$ million addresses.

The Python code used in the analysis starts from the list of private keys and computes the corresponding addresses in the correct format (see Chapter 3, Section 3.5 for more details). The code then checks whether these addresses have ever appeared on the analyzed blockchains. The results of the analysis are presented in Table 8.2, which summarizes all the findings. The code used in this analysis is publicly accessible here [60].

	H	$g_iH, i = 1,\dots,7$
Uncompressed Bitcoin addresses	4 addresses found [118]	No address found
Compressed Bitcoin addresses	3 addresses found	No address found
Segwit Bitcoin addresses	1 address found	No address found
Ethereum	1 address found	No address found
Dogecoin	3 addresses found	No address found
Litecoin	2 addresses found	No address found
Dash	2 addresses found	No address found
Zcash	No address found	No address found
Bitcoin Cash	6 addresses found	No address found

Table 8.2 Number of addresses used in the analysed blockchains found by this attack.

From this analysis, it turns out that only three addresses are non-trivial, i.e. they do not have 1 or -1 as the private key. Notice that, for example, for Dogecoin the script has found 3 trivial addresses, since each private key can be associated to two different encodings of the public key (compressed and uncompressed), for a total of 4 potential trivial addresses (Section 3.5). All of the three non-trivial addresses belong to the Bitcoin blockchain. The two addresses already found by Sala et al. [118] were generated in 2013 and 2014, so they are present in Bitcoin Cash as well, while the third one is the address

1H1jFxaHFUNT9TrLzeJVhXPyiSLq6UecUy,

and it was generated starting from a compressed public key. It was created on October 15, 2019, after the Bitcoin Cash fork, which is dated August 1, 2017. The address has no cryptocurrency into it nowadays.

The story of the address

1PSRcasBNEwPC2TWUB68wvQZHwXy4yqPQ3,

already found by Sala et al., is somewhat interesting. That address was created on March 15, 2014, and the funds got removed in June 2018 only after the authors of the previously cited papers contacted them (read [118] to know more about what they have done). Hence, the address is old enough to appear on BCH. It is indeed present:

qrmzrdndlfxpnkk3w5d5l7etnysnqfgk5yxsf6k0qq,

after the new encoding with the CashAddress format. However, the address is empty as someone moved the funds on that address on May 1, 2019.
On November 15, 2018, Bitcoin Cash had yet another hard fork, that led to the birth of *Bitcoin SV* [23]. Hence, this address is also present on that blockchain. By examining a Bitcoin SV explorer, it turns out that the funds from that address got also moved on May 1, 2019, two minutes earlier than the transaction on Bitcoin Cash! It seems likely that the funds were moved by the same entity.

Finally, it was not possible to find any address in the seven cosets that were examined. Although this result is interesting, it was not unexpected as the probability of finding an address using a brute-force attack is extremely low [118]. Therefore, the security of these blockchains is not compromised, as it is more profitable to mine and participate in the protocol in an honest manner rather than attempting to generate random private keys to steal cryptocurrency [26]. This result may also suggest that the non-trivial addresses were generated as a result of a suboptimal implementation of certain wallets. It would be worthwhile to investigate which Bitcoin wallets may have generated these addresses.

8.3.1 Other curves to examine

The most used elliptic curve in blockchains, beside *secp256k1*, is certainly *Curve25519* [14]. It is employed in several blockchains, including Monero [2], Cardano [70], Solana [140], and Algorand [36]. The defining curve is the Montgomery curve

$$E(\mathbb{F}_p):\ y^2 = x^3 + 486662x^2 + x,$$

where $p = 2^{255} - 19$ is a prime number. The number of rational points is $n = 8l$, where

$$l = 2^{252} + 27742317777372353535851937790883648493.$$

This curve is *birationally equivalent* to the edwards25519 curve presented in Chapter 3 (that is, they are isomorphic except for a finite set of points). Reasoning as in the case of the curve *secp256k1*, the private key can be chosen among the non-zero points, whose number is

$$l - 1 = 2^2 \cdot 3 \cdot 11 \cdot q_1 \cdot q_2,$$

where

$$\begin{aligned} q_1 &= 2766026242816422399372186805571398266687 47, \\ q_2 &= 1982114232309307540130845257 63697. \end{aligned}$$

The private key space is then in bijection with $\mathbb{Z}_l^*(\cdot)$, it is cyclic, and it has a unique subgroup for each divisor of its order $l-1$. Notice that an analogue analysis cannot be performed, since the small subgroup contains only $132 = 2^2 \cdot 3 \cdot 11$ points.

Finally, another curve that might be worth analysing in the future is the NIST *P-256* curve, defined in the specification [37]. It is actually used by NEO [104], Tezos [62] and Ontology [108]. The curve is

$$E(\mathbb{F}_p):\ y^2 = x^3 - 3x + B,$$

where

$$\begin{aligned} B =& 410583637251521421293261297800472684091144410159937255 \\ & 54835256314039467401291, \end{aligned}$$

and p is the prime number

$$p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1.$$

The order of this curve is

$$q = 115792089210356248762697446949407573529996955224135760 \\ 342422259061068512044369,$$

and $q-1$ factorises as

$$q - 1 = 2^4 \cdot 3 \cdot 71 \cdot 131 \cdot 373 \cdot 3407 \cdot 17449 \cdot 38189 \cdot 187019741 \cdot 622491383 \cdot q_1 \cdot q_2,$$

where

$$q_1 = 2624747550333869278416773953, \\ q_2 = 1002328039319.$$

This case is certainly more interesting than the previous one, since there are several subgroups of order dividing $q-1$ that may be inspected. Blockchains that use or intend to use this curve in the future should pay attention to the attack presented in this chapter.

Chapter 9

Conclusions

This work has described three algorithms that can be potentially implemented in a blockchain environment, along with a key recovery attack on a particular subset of keys for the secp256k1 curve.
The ideas proposed and described have different motivations, but with one common goal: the improvement of cryptographic tools already used by blockchains, from the point of view of security, privacy or scalability, and the verification of some addresses generated by wallets, as in the case of the attack described in Chapter 8.
In more detail, the original part of the thesis begins with a description of TRIFORS, a ring signature based on a modern cryptographic assumption, sATFE, believed to be post-quantum. The study of post-quantum algorithms is a hot research topic, in part because of NIST's call to arms to standardise new algorithms based on assumptions that are considered secure even in case of the advent of quantum computing. TRIFORS, and its linkable version, have a competitive length compared to other ring signatures based on post-quantum assumptions, and could therefore be usable as a transaction signing method in the future.
Next, a new framework that generalizes the Proof-of-Work consensus algorithm used by Bitcoin and many other blockchains is described. In this case, the motivation behind this research work is the excessive concentration of computing power within a few mining pools, as well as the fact that the generated bitcoins are totally distributed to the miner (or mining pool) that inserts a new blockchain that continues the chain, leaving all the rest of the actively participants of the network empty-handed. The proposed system solves both problems, at the cost of slightly increasing the number of forks.

The third article describes a system to solve disputes derived from decentralised applications or, more generally, smart contracts. In this case, the idea came after observing that, at the moment, applications that deal with dispute resolution in this area use arbitration techniques, and that all of them give very few guarantees of privacy to the users involved. The proposed idea exploits two protocols already active on Ethereum's blockchain, Semaphore and MACI, and it ensures that the parties involved in the dispute have the final say through a clever use of quadratic voting.
Finally, the last work described in this thesis generalises an attack proposed in 2020 that seemed to potentially undermine the security of certain addresses generated from public keys that were elements of the secp256k1 elliptic curve. To understand the scope of the attack, the work examines different address formats and different blockchains; fortunately, it seems that the problem is limited to Bitcoin's blockchain, and it most likely stems from a bad implementation of some wallet.

Each of the proposed results, however, still has many open issues that need to be studied in more detail.
TRIFORS should be implemented to estimate the time it takes to generate a signature; in addition, it should be taken into account that the cryptographic hypothesis on which it is based is very recent and thus further cryptographic analysis is needed to be convinced of the security of the signature.
Similarly, the proposed new Proof-of-Work systems should be implemented to obtain more information, both about the parameters to be used and the actual security of the model. In particular, the j parameter should be optimized so as to minimize the variance of the mining activity, but without exaggeratedly increasing the possibility of having a fork within the protocol.
As far as the dispute resolution platform is concerned, the proposed idea might be a good starting point, provided that soulbound tokens can indeed be assigned to judges for "knowledge fields" without having to use a trusted third party again. In addition, more research is needed to figure out how to solve the problem of "malicious but trusted" judges trying to exclude other trusted judges from the platform for their own interests. Another avenue for future research might be to see if this model can also be used for different situations, such as a mechanism within another decentralized application to vote on proposals made by participants to improve the platform.
Finally, with regard to the attack made on the elliptic curve used by Bitcoin and many other blockchains, it might be interesting to analyze more thoroughly the wallets

used during the period when these addresses were generated, to try to confirm the hypothesis that this behavior really stems from a flawed implementation of the wallet itself.

References

[1] Navid Alamati, Luca De Feo, Hart Montgomery, and Sikhar Patranabis. Cryptographic group actions and applications. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 411–439. Springer, 2020.

[2] Kurt M Alonso et al. Zero to monero, 2020.

[3] Ankr. Ankr website, 2023. https://www.ankr.com/, Last accessed on 2023-05-17.

[4] Aragon. Aragon website, 2020. https://aragon.org/, Last accessed on 2023-05-17.

[5] Adam Back et al. Hashcash-a denial of service counter-measure. 2002.

[6] Fadi Barbára, Andrea Gangemi, and Giulio Stefano Ravot. Overcoming the legal challenges of smart contracts through a “smart” commercial mediation. *Quaderni di conciliazione*, 24(2):271, 2020.

[7] Alessandro Barenghi, Jean-François Biasse, Edoardo Persichetti, and Paolo Santini. LESS-FM: fine-tuning signatures from the code equivalence problem. In *International Conference on Post-Quantum Cryptography*, pages 23–43. Springer, 2021.

[8] Alessandro Barenghi, Jean-François Biasse, Tran Ngo, Edoardo Persichetti, and Paolo Santini. Advanced signature functionalities from the code equivalence problem. *International Journal of Computer Mathematics: Computer Systems Theory*, 7(2):112–128, 2022.

[9] Base58. Base58 Encoding, 2023. https://en.bitcoin.it/wiki/Base58Check_encoding, Last accessed on 2023-05-17.

[10] Danilo Bazzanella and Andrea Gangemi. Bitcoin: a new proof-of-work system with reduced variance. *Financial Innovation*, 9(1):1–14, 2023.

[11] BCH Addresses encoding. Bitcoin Cash Addresses, 2023. https://github.com/bitcoincashorg/bitcoincash.org/blob/master/spec/cashaddr.md, Last accessed on 2023-05-17.

[12] Bech32. Bech32 Encoding, 2023. https://en.bitcoin.it/wiki/Bech32, Last accessed on 2023-05-17.

[13] Emanuele Bellini, Andre Esser, Carlo Sanna, and Javier Verbel. MR-DSS–Smaller MinRank-based (Ring-) Signatures. *Cryptology ePrint Archive*, 2022.

[14] Daniel J Bernstein. Curve25519: new Diffie-Hellman speed records. In *Public Key Cryptography-PKC 2006: 9th International Conference on Theory and Practice in Public-Key Cryptography, New York, NY, USA, April 24-26, 2006. Proceedings 9*, pages 207–228. Springer, 2006.

[15] Daniel J Bernstein, Peter Birkner, Marc Joye, Tanja Lange, and Christiane Peters. Twisted edwards curves. *Lecture Notes in Computer Science*, 5023: 389–405, 2008.

[16] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Keccak. In *Advances in Cryptology–EUROCRYPT 2013: 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings 32*, pages 313–314. Springer, 2013.

[17] Ward Beullens. Graph-theoretic Algorithms for the Alternating Trilinear Form Equivalence problem. *Cryptology ePrint Archive*, 2022.

[18] Ward Beullens, Thorsten Kleinjung, and Frederik Vercauteren. CSI-Fish: efficient isogeny based signatures through class group computations. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 227–247. Springer, 2019.

[19] Ward Beullens, Shuichi Katsumata, and Federico Pintore. Calamari and Falafl: logarithmic (linkable) ring signatures from isogenies and lattices. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 464–492. Springer, 2020.

[20] Binance. Binance website, 2021. https://www.binance.com/en, Last accessed on 2023-05-17.

[21] George Bissias and Brian Neil Levine. Bobtail: Improved Blockchain Security with Low-Variance Mining. In *NDSS*, 2020.

[22] Bitcoin Cash. Bitcoin Cash website, 2023. https://bitcoincash.org/, Last accessed on 2023-05-17.

[23] Bitcoin SV. Bitcoin SV website, 2023. https://www.bitcoinsv.com/, Last accessed on 2023-05-17/.

[24] Dan Boneh and Victor Shoup. A graduate course in applied cryptography. *Draft 0.5*, 2020.

[25] Sean Bowe, Jack Grigg, and Daira Hopwood. Recursive proof composition without a trusted setup. *Cryptology ePrint Archive*, 2019.

[26] Alessandro Brighente, Martina Camaioni, Mauro Conti, and Emilio Olivastri. Should I mine or Should I Break: On the Worthiness of Brute-Forcing Cryptocurrency Addresses. In *2022 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*, pages 279–284. IEEE, 2022.

[27] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE symposium on security and privacy (SP)*, pages 315–334. IEEE, 2018.

[28] Vitalik Buterin. MACI proposal, 2019. https://ethresear.ch/t/minimal-anti-collusion-infrastructure/5413, Last accessed on 2023-05-17.

[29] Vitalik Buterin. Quadratic Payments: a Primer, 2019. https://vitalik.ca/general/2019/12/07/quadratic.html, Last accessed on 2023-05-17.

[30] Vitalik Buterin. Negative votes in quadratic funding, 2020. https://ethresear.ch/t/negative-votes-in-quadratic-funding/6855, Last accessed on 2023-05-17.

[31] Vitalik Buterin, Zoë Hitzig, and E Glen Weyl. A flexible design for funding public goods. *Management Science*, 65(11):5171–5187, 2019.

[32] Vitalik Buterin, Diego Hernandez, Thor Kamphefner, Khiem Pham, Zhi Qiao, Danny Ryan, Juhyeok Sin, Ying Wang, and Yan X Zhang. Combining GHOST and casper. *arXiv preprint arXiv:2003.03052*, 2020.

[33] Vitalik Buterin et al. A next-generation smart contract and decentralized application platform. *white paper*, 3(37):2–1, 2014.

[34] Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. CSIDH: an efficient post-quantum commutative group action. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 395–427. Springer, 2018.

[35] David Chaum and Eugène van Heyst. Group Signatures. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 257–265. Springer, 1991.

[36] Jing Chen and Silvio Micali. Algorand. *arXiv preprint arXiv:1607.01341*, 2016.

[37] Lily Chen, Dustin Moody, Karen Randall, Andrew Regenscheid, and Angela Robinson. Recommendations for discrete logarithm-based cryptography: Elliptic curve domain parameters. 2023.

[38] Zhili Chen, Dung Hoang Duong, Ngoc Tuong Nguyen, Youming Qiao, Willy Susilo, and Gang Tang. On digital signatures based on isomorphism problems: QROM security and ring signatures. *Cryptology ePrint Archive*, 2022.

[39] Dmitry Chistikov, Szabolcs Iván, Anna Lubiw, and Jeffrey Shallit. Fractional Coverings, Greedy Coverings, and Rectifier Networks. In *34th Symposium on Theoretical Aspects of Computer Science*, 2017.

[40] Clr.fund. clr.fund website, 2023. https://clr.fund/#/, Last accessed on 2023-05-17.

[41] Luis Cuende and Jirge Izquierdo. Aragon network a decentralized infrastructure for value exchange. *Available at SSRN 4105763*, 2017.

[42] Giuseppe D'Alconzo. Monomial Isomorphism for Tensors and Applications to Code Equivalence Problems. *Cryptology ePrint Archive*, 2023.

[43] Giuseppe D'Alconzo and Andrea Gangemi. TRIFORS: LINKable Trilinear Forms Ring Signature. *Cryptology ePrint Archive*, 2022.

[44] Paolo D'Arco, Zahra Ebadi Ansaroudi, and Francesco Mogavero. Multi-stage Proof-of-Works: Properties and Vulnerabilities. *Cryptology ePrint Archive*, 2020.

[45] Luca De Feo and Steven D Galbraith. SeaSign: compact isogeny signatures from class group actions. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 759–789. Springer, 2019.

[46] Christian Decker and Roger Wattenhofer. Information propagation in the bitcoin network. In *IEEE P2P 2013 Proceedings*, pages 1–10. IEEE, 2013.

[47] Antonio J Di Scala, Andrea Gangemi, Giuliano Romeo, and Gabriele Vernetti. Special subsets of addresses for blockchains using the secp256k1 curve. *Mathematics*, 10(15):2746, 2022.

[48] Whitfield Diffie and Martin E Hellman. New directions in cryptography. In *Democratizing Cryptography: The Work of Whitfield Diffie and Martin Hellman*, pages 365–390. 2022.

[49] Hans Dobbertin, Antoon Bosselaers, and Bart Preneel. RIPEMD-160: A strengthened version of RIPEMD. In *Fast Software Encryption: Third International Workshop Cambridge, UK, February 21–23 1996 Proceedings 3*, pages 71–82. Springer, 1996.

[50] Dogecoin. Dogecoin website, 2023. https://dogecoin.com/, Last accessed on 2023-05-17.

[51] Evan Duffield and Daniel Diaz. Dash: A privacycentric cryptocurrency, 2015.

[52] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *Annual international cryptology conference*, pages 139–147. Springer, 1992.

[53] EIP55 Specifications. EIP55 standard, 2016. https://github.com/Ethereum/EIPs/blob/master/EIPS/eip-55.md, Last accessed on 2023-05-17.

[54] ERC1497 Specifications. ERC1497 standard, 2018. https://github.com/ethereum/EIPs/issues/1497, Last accessed on 2023-05-17.

[55] Muhammed F Esgin, Raymond K Zhao, Ron Steinfeld, Joseph K Liu, and Dongxi Liu. Matrict: efficient, scalable and post-quantum blockchain confidential transactions protocol. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 567–584, 2019.

[56] Muhammed F Esgin, Ron Steinfeld, and Raymond K Zhao. MatRiCT+: More efficient post-quantum private blockchain payments. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 1281–1298. IEEE, 2022.

[57] Ittay Eyal and Emin Gün Sirer. How to disincentivize large bitcoin mining pools. *Blog post: http://hackingdistributed. com/2014/06/18/how-to-disincentivize-large-bitcoin-mining-pools*, 2014.

[58] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Conference on the theory and application of cryptographic techniques*, pages 186–194. Springer, 1986.

[59] Andrea Gangemi. Towards a Privacy-Preserving Dispute Resolution Protocol on Ethereum. *arXiv preprint arXiv:2303.00533*, 2023.

[60] Github folder with the code for the secp256k1 attack paper. Github folder, 2022. https://github.com/GitGab19/blockchain-address-list-generation, Last accessed on 2023-05-17.

[61] Brandon Goodell, Sarang Noether, and Arthur Blue. Concise Linkable Ring Signatures and Forgery Against Adversarial Keys. *Cryptology ePrint Archive*, 2019.

[62] LM Goodman. Tezos: A self-amending crypto-ledger position paper. *Aug*, 3: 2014, 2014.

[63] Joshua A Grochow and Youming Qiao. Isomorphism problems for tensors, groups, and cubic forms: completeness and reductions. *arXiv preprint arXiv:1907.00309*, 2019.

[64] Joshua A Grochow and Youming Qiao. On the complexity of isomorphism problems for tensors, groups, and polynomials I: Tensor Isomorphism-completeness. In *12th Innovations in Theoretical Computer Science Conference (ITCS 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

[65] Joshua A Grochow, Youming Qiao, and Gang Tang. Average-case algorithms for testing isomorphism of polynomials, algebras, and multilinear forms. *arXiv preprint arXiv:2012.01085*, 2020.

[66] Jens Groth. On the size of pairing-based non-interactive arguments. In *Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II 35*, pages 305–326. Springer, 2016.

[67] Jorge Guajardo and Christof Paar. Efficient algorithms for elliptic curve cryptosystems. In *Annual International Cryptology Conference*, pages 342–356. Springer, 1997.

[68] Shay Gueron, Edoardo Persichetti, and Paolo Santini. Designing a Practical Code-Based Signature Scheme from Zero-Knowledge Proofs with Trusted Setup. *Cryptography*, 6(1):5, 2022.

[69] Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. Zcash protocol specification. *GitHub: San Francisco, CA, USA*, 4:220, 2016.

[70] Charles Hoskinson. Why we are building Cardano. *IOHK (accessed 18 December 2017) https://whycardano. com*, 2017.

[71] IPFS. IPFS website, 2022. https://ipfs.tech/, Last accessed on 2023-05-17.

[72] Markus Jakobsson and Ari Juels. Proofs of work and bread pudding protocols. In *Secure information networks*, pages 258–272. Springer, 1999.

[73] Zhengfeng Ji, Youming Qiao, Fang Song, and Aaram Yun. General linear group action on tensors: a candidate for post-quantum cryptography. In *Theory of Cryptography Conference*, pages 251–281. Springer, 2019.

[74] Jur. Jur website, 2022. https://jur.io/, Last accessed on 2023-05-17.

[75] Kleros. Kleros website, 2019. https://kleros.io/, Last accessed on 2023-05-17.

[76] Kleros dispute example 1. Curated list, 2023. https://resolve.kleros.io/cases/552, Last accessed on 2023-05-17.

[77] Kleros dispute example 2. Escrow, 2023. https://resolve.kleros.io/cases/561, Last accessed on 2023-05-17.

[78] Kleros dispute example 3. Insurance, 2023. https://resolve.kleros.io/cases/548, Last accessed on 2023-05-17.

[79] Kleros dispute example 4. Token, 2023. https://resolve.kleros.io/cases/547, Last accessed on 2023-05-17.

[80] Kleros dispute example 5. Social networks, 2023. https://resolve.kleros.io/cases/145, Last accessed on 2023-05-17.

[81] Kleros dispute example 6. Gitcoin grant, 2023. https://resolve.kleros.io/cases/406, Last accessed on 2023-05-17.

[82] Anthony W Knapp. *Elliptic curves*, volume 40. Princeton University Press, 1992.

[83] Donald E Knuth. Generating All Combinations and Partitions, volume 4, fascicle 3 of The Art of Computer Programming, 2005.

[84] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of computation*, 48 (177):203–209, 1987.

[85] Steven P Lalley and E Glen Weyl. Quadratic voting: How mechanism design can radicalize democracy. In *AEA Papers and Proceedings*, volume 108, pages 33–37, 2018.

[86] Laragon. Laragon website, 2023. https://laragon.org/, Last accessed on 2023-05-17.

[87] Clément Lesaege, Federico Ast, and W George. Kleros. *Whitepaper available at https://kleros. io/assets/whitepaper. pdf*, 2018.

[88] List of all Bitcoin Addresses. Bitcoin Addresses, 2023. http://alladdresses.loyce.club/?C=M;O=D, Last accessed on 2023-05-17/.

[89] List of Bitcoin pools. Bitcoin pools, 2023. https://btc.com/stats/pool, Last accessed on 2023-05-17/.

[90] Litecoin. Litecoin website, 2023. https://litecoin.com/en/, Last accessed on 2023-05-17.

[91] Joseph L. Liu, Victor K. Wei, and Duncan S. Wong. Linkable Spontaneous Anonymous Group Signature for Ad Hoc Groups. In *Australasian Conference on Information Security and Privacy*, pages 325 – 335. Springer, 2004.

[92] Xingye Lu, Man Ho Au, and Zhenfei Zhang. Raptor: a practical lattice-based (linkable) ring signature. In *International Conference on Applied Cryptography and Network Security*, pages 110–130. IEEE, 2019.

[93] Vadim Lyubashevsky, Ngoc Khanh Nguyen, and Gregor Seiler. SMILE: set membership from ideal lattices with applications to ring signatures and confidential transactions. In *Annual International Cryptology Conference*, pages 611–640. Springer, 2021.

[94] MACI. MACI github page, 2022. https://github.com/privacy-scaling-explorations/maci, Last accessed on 2023-05-17.

[95] Gregory Maxwell, Andrew Poelstra, Yannick Seurin, and Pieter Wuille. Simple schnorr multi-signatures with applications to bitcoin. *Designs, Codes and Cryptography*, 87(9):2139–2164, 2019.

[96] Kevin S McCurley. The discrete logarithm problem. In *Proc. of Symp. in Applied Math*, volume 42, pages 49–74. USA, 1990.

[97] Alessio Meneghetti, Massimiliano Sala, and Daniele Taufer. A survey on pow-based consensus. *Annals of Emerging Technologies in Computing (AETiC), Print ISSN*, pages 2516–0281, 2020.

[98] Ralph C Merkle. A digital signature based on a conventional encryption function. In *Conference on the theory and application of cryptographic techniques*, pages 369–378. Springer, 1987.

[99] James Metzger. The current landscape of blockchain-based, crowdsourced arbitration. *Macquarie Law Journal*, 19(Nov 2019):81–101, 2019.

[100] Ian Miers, Christina Garman, Matthew Green, and Aviel D Rubin. Zerocoin: Anonymous distributed e-cash from bitcoin. In *2013 IEEE Symposium on Security and Privacy*, pages 397–411. IEEE, 2013.

[101] Andrew Miller, Ahmed Kosba, Jonathan Katz, and Elaine Shi. Nonoutsourceable scratch-off puzzles to discourage bitcoin mining coalitions. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 680–691, 2015.

[102] Victor S Miller. Use of elliptic curves in cryptography. In *Conference on the theory and application of cryptographic techniques*, pages 417–426. Springer, 1986.

[103] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review*, page 21260, 2008.

[104] Neo. Neo website, 2023. https://neo.org/, Last accessed on 2023-05-17.

[105] Jonas Nick, Tim Ruffing, and Yannick Seurin. MuSig2: simple two-round Schnorr multi-signatures. In *Annual International Cryptology Conference*, pages 189–221. Springer, 2021.

[106] Shen Noether, Adam Mackenzie, et al. Ring confidential transactions. *Ledger*, 1:1–18, 2016.

[107] OEIS Foundation Inc. The On-Line Encyclopedia of Integer Sequences, 2022. Published electronically at http://oeis.org.

[108] Ontology. Ontology website, 2023. https://ont.io/, Last accessed on 2023-05-17.

[109] EC Park and Ian F Blake. On the mean number of encryptions for tree-based broadcast encryption schemes. *Journal of Discrete Algorithms*, 4(2):215–238, 2006.

[110] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Annual international cryptology conference*, pages 129–140. Springer, 1992.

[111] Wouter Penard and Tim van Werkhoven. On the secure hash algorithm family. *Cryptography in context*, pages 1–18, 2008.

[112] Maharage Nisansala Sevwandi Perera, Toru Nakamura, Masayuki Hashimoto, Hiroyuki Yokoyama, Chen-Mou Cheng, and Kouichi Sakurai. A Survey on Group Signatures and Ring Signatures: Traceability vs. Anonymity. *Cryptography*, 6(1):3, 2022.

[113] PoH. Proof of humanity website, 2021. https://www.proofofhumanity.id/, Last accessed on 2023-05-17.

[114] PSE. PSE website, 2023. https://appliedzkp.org/, Last accessed on 2023-05-17/.

[115] Poonam Rani and Rajul Bhambay. A Comparative Survey of Consensus Algorithms Based on Proof of Work. In *Emerging Technologies in Data Mining and Information Security*, pages 261–268. Springer, 2023.

[116] Krijn Reijnders, Simona Samardjiska, and Monika Trimoska. Algebraic Attack on the Alternating Trilinear Form Equivalence Problem. *CBCrypto 2023 International Workshop on Code-Based Cryptography*, 2023.

[117] Ronald Rivest, Adi Shamir, and Yael Tauman. How to Leak a Secret. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 552–565. Springer, 2001.

[118] Massimiliano Sala, Domenica Sogiorno, and Daniele Taufer. A small subgroup attack on bitcoin address generation. *Mathematics*, 8(10):1645, 2020.

[119] Lakshmi Siva Sankar, M Sindhu, and M Sethumadhavan. Survey of consensus protocols on blockchain applications. In *2017 4th international conference on advanced computing and communication systems (ICACCS)*, pages 1–5. IEEE, 2017.

[120] Palash Sarkar. A New Blockchain Proposal Supporting Multi-Stage Proof-of-Work. *Cryptology ePrint Archive*, 2019.

[121] Thomas C Schelling. *The Strategy of Conflict: with a new Preface by the Author*. Harvard university press, 1980.

[122] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *Conference on the Theory and Application of Cryptology*, pages 239–252. Springer, 1990.

[123] Segwit. segwit Encoding, 2023. https://github.com/bitcoin/bips/blob/master/bip-0173.mediawiki, Last accessed on 2023-05-17.

[124] Semaphore. Semaphore website, 2022. https://semaphore.appliedzkp.org/, Last accessed on 2023-05-17.

[125] Ning Shi. A new proof-of-work mechanism for bitcoin. *Financial Innovation*, 2(1):1–8, 2016.

[126] Peter W Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th annual symposium on foundations of computer science*, pages 124–134. Ieee, 1994.

[127] Joseph H Silverman. *The arithmetic of elliptic curves*, volume 106. Springer, 2009.

[128] Joseph H Silverman and Joe Suzuki. Elliptic curve discrete logarithms and the index calculus. In *Advances in Cryptology—ASIACRYPT'98: International Conference on the Theory and Application of Cryptology and Information Security Beijing, China, October 18–22, 1998 Proceedings*, pages 110–125. Springer, 1998.

[129] Solidity. Solidity website, 2023. https://soliditylang.org/, Last accessed on 2023-05-17/.

[130] Nick Szabo. Formalizing and securing relationships on public networks. *First monday*, 1997.

[131] Gang Tang, Dung Hoang Duong, Antoine Joux, Thomas Plantard, Youming Qiao, and Willy Susilo. Practical Post-Quantum Signature Schemes from Isomorphism Problems of Trilinear Forms. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 582–612. Springer, 2022.

[132] Taproot. Taproot update, 2023. https://en.bitcoin.it/wiki/BIP_0341, Last accessed on 2023-05-17/.

[133] Tatum. Tatum, 2023. https://tatum.io/, Last accessed on 2023-05-17.

[134] Unirep. Unirep github page, 2022. https://github.com/Unirep/Unirep, Last accessed on 2023-05-17.

[135] Paul C Van Oorschot and Michael J Wiener. Parallel collision search with cryptanalytic applications. *Journal of cryptology*, 12:1–28, 1999.

[136] VeChainThor. VeChainThor website, 2023. https://www.vechain.org/, Last accessed on 2023-05-17/.

[137] Web3 storage. Web3 storage website, 2022. https://web3.storage/, Last accessed on 2023-05-17.

[138] E Glen Weyl, Puja Ohlhaver, and Vitalik Buterin. Decentralized Society: Finding Web3's Soul. *Available at SSRN 4105763*, 2022.

[139] Gavin Wood. Polkadot: Vision for a heterogeneous multi-chain framework. *White Paper*, 21:2327–4662, 2016.

[140] Anatoly Yakovenko. Solana: A new architecture for a high performance blockchain v0. 8.13. *Whitepaper*, 2018.

[141] Tsz Hon Yuen, Muhammed F Esgin, Joseph K Liu, Man Ho Au, and Zhimin Ding. Dualring: Generic Construction of Ring Signatures with Efficient Instantiations. In *Advances in Cryptology–CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part I*, pages 251–281, 2021.