Cryptographic Tools For Blockchain

Bercovici Adrian, Erhan Adrian, Fechită George-Antonio

July 30, 2024

Abstract

In this paper we will cover the main aspects of the cryptographic tools for blockchain. We will go over the various techniques, protocols and procedures that focus on offering and most importantly guaranteeing the security of the confidential data and anonymity for the users of the blockchain. First we offer an introduction to what does the blockchain actually imply, providing a quick overview of the technology. Afterwards we will gradually dive into the main cryptographic techniques used and applied to blockchain.

Keywords— Blockchain, Hash Functions, Merkle Trees, Digital Signatures, Commitment Schemes, Non-interactive Proofs, Ring Signatures, Secure Multi-Party Computation

1 Introduction to Blockchain - Bercovici Adrian 3A6

Blockchain is a technology that seeks to be a kind of digital, decentralized ledger. If formally, a ledger would be kept by just one entity, that the other participants have to trust, Blockchain seeks to give every participant their own copy of ledger and remove the need for trust entirely.

To digitalize a ledger, one of the first problems that must be solved is the need for transactions to reliably be traced back to one particular user who started the transaction. Consider Alice and Bob, two participants in the ledger and a transaction "Alices gives Bob 100$". They key cryptographic properties of interest in this particular situation are:

- Authentication: or being able to know that this transaction was made by Alice and Alice alone
- Integrity: or being able to know that this transaction was not tempered with by anyone, and that indeed Alice gave Bob 100$ and not 10$ or 1000$
- Non repudiation: or the property that Alice cannot retroactively deny making the transaction

Digital signatures are the cryptographic tool that provides all 3 of the above mentioned, required properties. Now consider that these operations require the context of public key cryptography. Each user will have a pair of keys, one private, used to sign the transaction and one public key, used to verify it. A transaction is not valid unless signed by the user.

When a transaction is made, given there is no one central authority to report it to, the sender has to broadcast the transaction to the other participants in the Blockchain. Consider a situation where Alice gives Bob 100 $, but only broadcasts this transaction to Bob, in an attempt to fool him. Now Bob's ledger will show he received 100$ from Alice, but all the other participants' ledgers will not contain that, as it was not broadcasted to them.

This brings forth the second important problem that Blockchain has to solve:

Whose ledger should be trusted? Given two ledgers A and B, the participants will trust will trust the ledger that has the most work put into it

To understand the concept of work and proof of work, in the context of Blockchain, a general overview of cryptographic hash functions must first be provided. A cryptographic hash function, like any hash function, maps an infinite space to a finite space. However, unlike regular hash functions, cryptographic hash functions are resistant to collisions, strongly not reversible and appear random. Given two strings of bits, A and A' where the Hamming distance between the two sequences is exactly one, the digests of a cryptographic hash function ran on the two strings will produces two outputs separated by very large, often maximum Hamming distance, or in simpler terms the two digests will be very distinct from one another.

Consider a block (a list) of transactions, and a nonce - the proof of work is a nonce that when applied to the block of transactions causes the cryptographic hash to have some sort of predefined property such as starting with a particular number of zeros. Due to the pseudo random nature of cryptographic hash functions, it can be assumed with reasonable certainty that computational work had to be performed to obtain a nonce

that creates the desired result. This describes the process known as 'Mining'.

So, once a transaction is made, it is broadcasted to the network, miners add the transaction to the block and search for the proof of work. Once the proof of work is found, the currently last block in the miner's ledger is hashed and linked to the new block that was just created. Note that a block is not valid without a proof of work. Every to block, except the first, contains transactions, the proof of work and the hash of the previous block. This links the block in a chain that makes it computationally difficult to modify any links in the chain, as that would compromise the following links and would require re-computation of the proof of work.

To incentives participants to partake in chain building or mining, the user that obtains a proof of work is rewarded with newly generated tokens. Also, since most Blockchain technologies implement a hard cap on the number of transactions that can be added to a block, users can incentives miners to add their transaction to the block by offering a commission.

It is important to note that the further behind a transaction is in the Blockchain, the more trust worthy it is, because more computational work has been done. Consider the aforementioned situation where Alice makes a transaction to Bob and only broadcasts it to him. Bob will compare Alice's ledger to the ledgers of the network peers. In order to maintain the illusion that Alice has transferred tokens to Bob, Alice would have to continuously provide more proof of work than the entire network, which is an unfeasible task. Additionally, if a user was to modify any block from the chain, that user would have to recomputed the proof of work for all the links in the chain after that block – which is again, unfeasible.

These mechanisms describe a system in which forging information Is unfeasible.

2 Hash functions and Merkel trees - Bercovici Adrian 3A6

Hash functions: Hash functions take an input from an infinite space and map it to an output in a finite space. Due to the pigeonhole principle, any hash function will produce "collisions" or a situation where two inputs map to the same output. Cryptographic hash functions however must have a property known as collision resistance - meaning that given any pair (input˙1, output) an adversary only has a negligible chance of finding input˙2 that maps to the same output.

Another important property of cryptographic hash functions is their pseudo-random behavior. A cryptographic hash must not leak any information about the original message and reversing a hash into the input must only be achievable with negligible probability. Additionally the hamming distance any two digests must leak no information about the hamming distance between the inputs.

In more theoretical terms, cryptographic hash functions can be viewed as an oracle that takes any message of any length and assigns to it a random strings of bits of length L - producing a kind of cryptographic fingerprint of the message.

(Cryptographic) Hash functions are a corner stone of the Blockchain – linking blocks together and providing trust through proof of work being the two most important uses. Each link in chain contains transactions and a hash to the previous block. The proof of work is then computed over the entire data of the block, including the hash of the previous link. This makes modifying data in any way unfeasible. For example, consider users Alice, Bob and Charlie and a block where the transactions "Alice gives Bob 100$" and "Alice gives Charlie 50$". If Alice wanted to modify the block in any, or remove it, the change would corrupt the proof of work of the entire chain from that link forward.

Merkel trees

A Merkel tree is a tree in which the value of any node that is not leaf is equal to the hash of the concatenation of the values of its children, and the values of the nodes that are leafs are hashes of the some fragment of data, alongside a pointer to that data.

Consider some data, a hash of the data and a Merkel tree of that data. If the original data is corrupted in any way, both the hash and the root of the Merkel tree will be able to indicate that, however the Merkel tree is able to tell precisely which fragments have been altered/corrupted. This structure can be used on distributed systems to verify integrity of data.

In the Blockhain, Merkle trees are used to create a cryptographic summary of the block, with transactions being leaves in the tree.

3 Digital Signature - Bercovici Adrian 3A6

Digital signatures are mechanisms derived from public key cryptography that ensure the authenticity, integrity and non-repudiation of a message. A digital signature scheme has 3 components: Key generation algorithm, Signature generation algorithm, Signature verification algorithm. There are multiple types of digital signatures:

- plain
- aggregate
- multi-signatures
- threshold
- forward-secure
- Plain digital signatures:
  A way of signing message by one user. The current standard is RSA that generates the signature by taking a message, hashing it and then encrypting it with the private RSA key. To verify the signature, the algorithm takes the received signature and decrypts it using the public key
- Aggregate signatures:
  Given the signatures of multiple users on one particular message, aggregate signatures create a more compact single signature. In Blockchain, aggregation can be used to decrease the storage footprint of multiple signatures. Given that in Blockchain, the same data is stored in multiple places, the impact can be quite important.
- Multi-signatures: Multi-signatures allow multiple parties to create a joint signature on a message. More naïve implementations concatenate the signatures of all users, but there exist specialized multi-signature algorithms create more compact signatures in terms of length. In Blockchain, multi signatures can be useful for MIMO transactions – multiple input multiple output.
- Threshold Signatures:
  A specific kind of multi signatures where, given N parties, a common signature will be generated if and only if at least K parties, with K ¿= Threshold T collaborate.

4 Commitment schemes - Fechită George Antonio 3A6

Using a commitment scheme, a party can commit to a certain value, encrypt it, and deliver it to another party to be decoded at a later time. A commitment scheme can be seen as a digital safe in the following way: a sender chooses a message and commits to it by putting it inside the safe and locking that safe with a key. The sender then gives that safe containing the message to a receiver without the key to unlock it. At the reveal phase the sender will give the key to the receiver to open the safe and reveal the committed message.

Initially some public parameters (params) must be generated by an algorithm Gen, an example of such parameter could be the choosen hash functions if the commitment scheme is based on hash functions. Then the sender commits to a message (m) and uniformly chooses a random value r that is going to act like a key. An algorithm Com(params, m, r) outputs a value com that is sent to the receiver. Because commitment schemes are based on one-way functions the receiver cannot discover the message m from the value com. At the reveal phase the receiver acquires m and r from the sender and can verify the integrity of the message by computing Com(params, m, r) themselves. If the verified value is equal with the received com value then a honest verifier must accept it, otherwise reject it.

A commitment scheme must satisfy the following three properties:

- Hiding:
  Until the sender chooses to reveal the message to the receiver it is hidden
- Binding:
  The sender cannot successfully reveal another value than the committed value

- Correctness:
  A honest sender should be able to efficiently encode a committed message and reveal it in the future while a honest receiver should be able to efficiently check

Commitment experiment:

1. Public parameters are generated by the Gen algorithm
2. The adversary (A) can access the parameters and choose two messages: m_0 and m_1 to give the challenger
3. The challenger randomly picks one of the given messages and sends the committed output back to the adversary
4. The adversary must decide from which of the two sent messages m_0 and m_1 the received commitment originates
5. If the adversary is correct the output of the experiment is 1, otherwise it is 0

For a commitment scheme to be secure one of the conditions is that no PPT adversary should be able to correctly guess with more than $\frac{1}{2} + negl(n)$ probability from which message was the commitment generated.

Hiding experiment:

1. Public parameters are generated by the Gen algorithm
2. The adversary receives the public parameters and chooses two distinct pairs of messages and random values (m_0, r_0), (m_1, r_1) to send back to the challenger
3. If Com(params, m_0, r_0) = Com(params, m_1, r_1) the output of the experiment is 1, otherwise it is 0

For the commitment scheme to be secure another condition is that no PPT adversary should be able to find two pairs of messages and random values (m_0, r_0), (m_1, r_1) such that $m_0 \neq m_1$ and Com(params, m_0, r_0) = Com(params, m_1, r_1) with more than negligible probability.

Commitment schemes based on hash functions:

In a commitment scheme based on hash functions the sender computes $\mathrm{H}(m \,\|\, r)$ and sends it to the receiver as the commitment value. For the receiver to verify the correctness of the commitment, the message m and value r will be given by the sender at the reveal phase, such that a honest verifier could compute $\mathrm{H}(m \,\|\, r)$ and accept or reject the commit based on whether the value matches with the received commit.

Because hash functions are one way functions we know that it is hard to invert the function, satisfying the hiding property. However it is not correct to say that the protocol is computationally hiding because the hash function in use could still leak some information about its input, as the n-th bit for example. A good reason for using the randomly chosen variable r is to have different possible outputs for the exact same committed message every time. If that wasn't the case, a dishonest receiver who knows that there are only a limited number of message a sender could commit to, could compute beforehand the hash for every single one of those candidate messages and compare the results with the given commit value.

For the binding property to be broken, a dishonest sender would have to find two pairs of messages and values (m_0, r_0), (m_1, r_1) such that $\mathrm{H}(m_0 \,\|\, r_0) = \mathrm{H}(m_1 \,\|\, r_1) = \mathrm{com}$, but if the hash function is collision resistant finding such a pair would be computationally too expensive so we can conclude that the protocol is computationally binding.

Pedersen commitment:

In Pedersen commitment the hard problem of discrete logarithm is used in the following way:

two elements g and y from a group G of large enough prime order are agreed upon by both parties but for similar reasons as in the case of hash functions some randomness is required for the scheme to be secure. For this purpose the element y is raised to the power of r, a random variable chosen by the sender. The commitment value is given by the formula $g^m \cdot y^r$ where m is the message to be committed. At the reveal phase the sender reveals the message m and random value r for the verifier to compute $g^m \cdot y^r$ and compare the result with the received value.

The protocol is computationally binding. For the binding propriety to be broken a dishonest sender would have to find two pairs (m_0, r_0), (m_1, r_1) such that $m_0 \neq m_1$ and $g^{m_0} \cdot y^{r_0} = g^{m_1} \cdot y^{r_1}$. The reason for which the protocol is computationally binding is that finding values for those pairs that satisfy the condition $g^{m_0} \cdot y^{r_0} = g^{m_1} \cdot y^{r_1}$ is equivalent to solving the discrete logarithm problem:

$$g^{m_0} \cdot y^{r_0} = g^{m_1} \cdot y^{r_1}$$

$$g^{m_0} = \frac{g^{m_1} \cdot y^{r_1}}{y^{r_0}}$$

$$z \overset{\text{not.}}{=} \frac{g^{m_1} \cdot y^{r_1}}{y^{r_0}}$$

$$g^{m_0} = z$$

The protocol is perfectly hiding. For any message m_1 there is an r_1 value such that $g^{m_0} \cdot y^{r_0} = g^{m_1} \cdot y^{r_1}$. There is a one to one mapping between the message space and the random value that leads to the same commitment value. It is important to note that even though this pair (m_1, r_1) that leads to the same commitment output as the pair (m_0, r_0) exists, it doesn't mean that it is easy to find.

It is not possible to have a commitment scheme that is both perfectly hiding and perfectly binding. If we suppose that a commitment scheme could be perfectly binding that would mean that a dishonest sender with unlimited computational power could not find two messages that have the same commitment output, therefore in such a commitment scheme there must exist a one to one mapping between the message space and the commitment outputs space, so any commitment output will lead to a unique message. If that was the case a dishonest receiver with unlimited computational power could brute-force the message space to find the unique message related to the received commitment value. Therefore the scheme will not be perfectly hiding.

When privacy is considered, the function of commitment schemes in blockchains is essential. In fact, there are a number of possible uses for blockchains where private data must be processed in a way that can be verified by the public while yet maintaining its privacy. Using commitment schemes it is possible to maintain the privacy of sensitive data while also knowing that the message (m) behind the encoded commitment (c) is unique so that c cannot come from a message $m' \neq m$.

5 Non-interactive Proofs - Fechită George Antonio 3A6

Sometimes it can be useful to be able to prove someone that you know a secret without revealing the secret itself. In this scope Goldwasser, Micali and Rackoff presented the idea of interactive zero knowledge proofs. The point of these proofs is to allow a prover to demonstrate to a verifier that a statement is true without having to disclose any private information. When a proof is interactive, it indicates that the prover and verifier communicate back and forth during the proof process. In this situation, the prover can transmit different bits of their input to the verifier, and the verifier can request additional information as required, helping to secure the prover's input's privacy.

The distinction between argument systems and proof systems should be made clear. Systems of proof demand that a false assertion cannot be shown to be true with more than a negligible probability, not even by an adversarial prover with unbounded computational power. Argument systems are similar, however the adversary in these systems is viewed as a polynomial-time algorithm. This is also known as the soundness property.

Some disadvantages of the interactive zero knowledge proofs are the fact that the back and forth communication between the prover and the verifier can be resource intensive, and the proof is available only for the verifier that interacts with the prover. To address these issues non-interactive zero knowledge proofs will be discussed next.

As stated above, one prover cannot use an interactive zero knowledge proof to convince more

verifiers without engaging with each of them, which would be inefficient. This lead to interest in creating non-interactive zero knowledge proof that can be verified by anyone since they are complete and no longer require additional communication from the prover. In 1986 Amos Fiat and Adi Shamir invented the Fiat-Shamir heuristic that is used to turn some interactive zero knowledge proofs into non-interactive zero knowledge proofs. Usually an interactive zero knowledge proofs follows the next steps:

1. The prover computes a commitment output for some information, usually called the witness to the verifier
2. The verifier chooses a random challange value as a response for the prover
3. The prover builds the proof based on the received challange value and the initial commitment value

The rationale behind the Fiat-Shamir transformation is that the prover can compute the challenge value on their own using a random function, such as a cryptographic hash function, as opposed to having the verifier give it to them. The resulting non-interactive zero knowledge remains secure even against adversary provers with unlimited computational powers as long as they are limited to polynomial number of queries to the random function.

5.1 SNARGs and SNARKs

A SNARG is a succinct non interactive argument system that produces short proof that can be quickly verified while a SNARK, a succint non interactive argument of knowledge, is similar but uses arguments of knowledge. This implies that SNARGs enable verifiers to conclude that a proof is true but do not permit verifiers to identify the kind of knowledge that the prover possesses.

The computational soundness property that SNARGs fulfill states that a computationally bounded adversary prover cannot prove a false claim to be true with more than negligible probability. A witness in the context of zero knowledge proofs refers to a piece of information that is used by the prover to prove a statement to be true. The computational knowledge soundness property that SNARKs have says that the verifier will not accept the proof unless the prover actually knows a legitimate witness. Both SNARGs and SNARKs have the property of perfect completeness which states that if a statement is true then a prover can always prove it.

It is not rigorous defined what is meant by succinct or fast, but in the context of zero knowledge SNARKs these requirements are related to the length of the proof that the prover generates and the verifier's execution time. The proof generation algorithm can be time and resource demanding, but the proof's size should still be short, considerably less than the amount of the information being proved, whether expressed in terms of the number of bits or operations required to verify it. In terms of the time duration needed to verify the proof, it should also be effective compared to the proof generation algorithm.

Some of the advantages of the zero knowledge SNARKs in the context of blockchain are preserving the privacy of transactions while still being able to be validated by anyone in the network, reason for which the Zcash blockchain uses them, compliance by making sure that regulations are followed, for instance by enabling a regulator to confirm a transaction's legitimacy without disclosing the transaction details, the privacy of smart contracts and many more.

Some of the disadvantages of SNARKs are the requirement of an initial trusted set-up and the fact that they lack quantum-resistance.

5.2 STARKs

STARKs are scalable transparent arguments of knowledge and they are considered to be alternatives to SNARKs with a few important differences. Since the proof generation in STARKs does not involve a trusted setup, the prover does not have to execute a pre-processing step prior to creating a proof, STARKs are supposed to be more transparent than zero knowledge SNARKs. This increases STARKs' resistance to potential security flaws.

In comparison to zero knowledge SNARKs, STARKs have bigger proof sizes. Typically, the size

of the proof produced by a STARK is linear in the size of the statement being proved. This happens because STARKs' proofs contain the entire computation, but zero knowledge SNARKs' proofs only contain a short, fixed-size computation. The consequence is that STARKs are more transparent and flexible than zero knowledge SNARKs, but less space-efficient. STARKs are also quantum resistant. The usage of STARKs in blockchains has not yet materialized, but they have been suggested as potential tools for future versions of some blockchains like Ethereum.

6 Verifiable random functions - Fechită George Antonio 3A6

A verifiable random function (VRF) is a probabilistic algorithm that receives a secret key (SK) as input along with a seed value (s) and outputs a pseudorandom result (r), along with a non-interactively proof that it is the correct output (π). A verifiable random function is composed from the following algorithms:

- Gen(K) is an algorithm that generates a pair of public (PK) and secret (SK) keys of K bits
- $F_{SK}(s)$ that based on the generated secret key and a seed value generates a pseudorandom value r
- $Prove_{SK}(s) = (F_{SK}(s), \pi_{SK}(s))$ that produces the pseudorandom value along with the proof $\pi_{SK}(s))$ that it is correct
- $Verify_{SK}(s, r = F_{SK}(s), \pi_{SK}(s))$ that outputs a boolean value to either accept or reject the pseudorandom value

A verifiable random function must fulfill the following properties:

- Uniqueness: For any seed s there cannot be more than one pair of pseudorandom value r and proof π such that $Verify_{SK}(s, r = F_{SK}(s), \pi_{SK}(s))$ evaluates as true.

$$\nexists(s, y_0, \pi_0), (s, y_1, \pi_1) \quad \text{s.t.} \quad (y_0, \pi_0) \quad \neq (y_1, \pi_1) \quad \text{and}$$
$$Verify_{SK}(s, y_0, \pi_0) = Verify_{SK}(s, y_1, \pi_1) = True$$

- Provability: If the pseudorandom output and proof have not been tampered with and correctly computed the verification function should always accept it.

$$Prove_{SK}(s) = (s, \pi) \iff Verify_{SK}(s, r = F_{SK}(s), \pi) = True$$

- Pseudorandomness: No pseudorandom value r generated by the verifiable random function can be distinguished with more than negligible probability from pure randomness even after multiple queries with different seeds.

In the context of blockchains verifiable random functions have many use cases such as selecting at random validators in a proof-of-stake consensus algorithm in a fair and verifiable manner. Some examples of proof-of-stake algorithms making use of verifiable random functions are Algorand, , Ouroboros Praos, and Ouroboros Genesis. Since verifiable random functions proofs are typically small in size, around a few hundred bytes, they can be included in a blockchain transaction without significant overhead. Verifiable random functions can also prove useful in the context of smart contracts, where they can be used to execute certain actions only if a certain condition is met. For example, a smart contract could release funds only if the verifiable random function output is below a certain threshold.

In conclusion, VRFs are helpful in the context of blockchain because they enables decentralized, verifiable, and unbiased randomness, which is necessary for many blockchain use cases. Additionally, VRF is effective and scalable, and it can be used to execute smart contracts, guarantee fairness in decision making, secure the system, and introduce randomness in distributed systems.

7 Privacy-Enhancing Protocols - Erhan Adrian 3A6

Among the different available types of privacy-enhancing protocols we can highlight the zero-knowledge proofs, ring signatures, and stealth addresses. In this section we will cover the ring signatures, group signatures and zero-knowledge proofs.

As a matter of improving the privacy for the users of the blockchain technology, a type of cryptographic technique known as Privacy-Enhancing Signatures is often being used. These Signatures allow the users to sign patches of data while keeping their identity and other sensitive information hidden. This is achieved by offering the user the ability to prove the authenticity of a signature without revealing unnecessary information about the signer or the signed data.

One of the key advantages of using Privacy-Enhancing Signatures is the ability for the user to remain anonymous whilst still being able to participate in the network. Another advantage is that the users' personal information is protected and it is impossible to trace their activities. Overall the Privacy-Enhancing Signatures is one of the main factors that enables the users to interact with the blockchain systems securely and anonymously.

7.1 Ring Signatures

Ring Signatures are a type of digital signature that represents a cryptographic technique which allows any member of a group of user to sign a piece of information, while the identity of the signer remains concealed to the rest of the group.

The term Ring Signature was elaborated by the MIT professor and cryptographer Ronald Rivest. In 2009 Adam Bender pointed out a flaw of the previous method of ring signature construction: it was based on a presumption that the public keys were generated honestly. He provided a stricter security definition while also describing a generic way of constructing the ring signatures along with a more efficient construction method based on computational assumptions.

In a ring signature, a group of users each have a public key, and a message can be signed by any member of the group without revealing which member actually signed it. At the same time the Ring Signature allows the signer to sign a message such that its authenticity can be verified relative to a ring (set) of users. In contrast with group signatures, ring signatures do not require a central authority for a good and maintained coordination within a group of users. In fact each individual user doesn't even need to know about the existence of any other specific individual in the group. This distinct trait of the Ring Signatures is especially important for the blockchain architecture and its use in cryptocurrency, as decentralization is one of the stable ideas that lay in the foundation of the cryptocurrency.

As we can see, it is worth emphasizing three important properties of the Ring Signature: the guarantee of untraceability of the signer, anonymity and the validation of the authenticity of the signature.

The anonymity of the signer is ensured by the fact that a verifier will at most know that the original signer is part of the ring of users, without knowing who the exact user is. For an outsider it is as well computationally infeasible to determine which user signed a message, thus the whole set of users may preserve their anonymity.

The scalability and size of a ring signature is an important aspect to keep track of, as the number of users in the ring may increase over time. Ring signatures in which the size of the signature increases as the number of users gets larger are relatively efficient. For example, Rivest's RSA-based scheme, Herranz and Sáez based on Schorr signatures or Boneh's scheme based on BLS signatures, all fall under this category. Some schemes even achieved a signature size that remains constant in the number of participants such as Dodis' scheme based on RSA or Qin's scheme based on the hardness of discrete logarithm problem.

Ring signatures with advanced security properties are especially important to be used for blockchains. Such signatures usually include threshold ring signatures, linkable ring signatures, accountable ring signatures and traceable ring signatures.

Threshold ring signatures are a variant of ring signatures that requires a certain number of users to cooperate in the signing protocol. For instance for a group of n users, a set t amount of users (u1, u2, ..., ut) need to compute a (t, n)-ring signature σ on a message m. In other words, the threshold ring signature combines threshold signing with the anonymity guarantees of the ring signatures.

- **Linkable ring signatures** make it possible for a user to identify whether any two signatures have been created by the same member of the ring, basically under the same private key. Even though one may determine if somebody has produced those two signatures, the identity of the producer still remains unknown. In 2005 Tsang and Wei proposed a linkable ring scheme with constant-size signatures based on RSA.
- **Accountable ring signatures** are a variant of ring signatures that is extended with an additional verifiable element allowing one designated receiver to revoke the anonymity of the signer if necessary. Having this property, accountable ring signatures are more suitable for use in situations where it is important to verify the identity of the signer while preserving the anonymity of the rest of users of the group. Jonathan Bootle proposed an implementation based on DDH assumption in which the signature size grows logarithmically with the number of ring users. Later that implementation has been improved to a constant-size signature.
- **Traceable ring signatures** are in a way a combination of linkable and accountable ring signatures. The scheme is similar to the linkage ring signature, but in case a signer issues more than one signature using the same private key, that key is revealed. This property forces the users to use a signature only once for a certain context in order to prevent double spending. Originally the scheme was based on the DDH assumption.

As we can see, depending on the type of ring signature and their respective building algorithm, the overall complexity is variable. For some algorithms the complexity is O(n), where n is the number of public keys being used, i.e. the number of members of the ring. Although a lot of powerful algorithms have been developed that allow to decrease substantially the complexity of generating a ring signature. Some of them managed to decrease it to a logarithmic function O(log n), while other have come even further and gained a constant complexity O(1).

The original scheme for ring signature includes a combination: function C_k, a key k, and an initialization vector v. The plaintext to be signed is signified as m and the ring's public keys by $P_1, P_2, \ldots, P_n$.

The algorithm for signature generation can be described as follows:

1. A key k is calculated using cryptographic hash functions: $k = H(m)$ where H is assumed to have a random oracle, because k will later be used as the key for a symmetric encryption function E_k;
2. A random vector v is generated;
3. A random number x_i is picked for all ring members except the signer (x_s), as xs will be calculated using the signer's private key. Then for every member compute the arbitrary values $y_i = g_i(x_i)$, where g_i is a trap-door function;
4. Solve the ring equation $C_{k,v}(y_1, y_2, ..., y_n) = E_k(y_n \oplus E_k(y_{n-1} \oplus E_k(... \oplus E_k(y_1 \oplus v)...)))$ for y_s
5. X_s is calculated from $g_s^{-1}(y_s)$
6. The ring signature is the $(2n+1)$-tuple $(P_1, P_2, \ldots, P_n; x_1, x_2, \ldots, x_n)$.

The algorithm for signature verification has three steps:

1. The public key trap door is applied on all x_i;
2. The symmetric key k is calculated: $k = H(m)$;
3. The equality $C_{k,v}(y_1, y_2, \ldots, y_n) = v$ is verified.

7.1.1 Application of ring signatures

Ring signatures have been used by various cryptocurrencies to improve transaction privacy. The idea behind it is that a transaction needs to be signed with a ring signature, that is signed relative to a certain number of transaction outputs, thus not revealing which exact output was spent. The original scheme of ring signatures might not satisfy all the needs for proper transactions, thus schemes with enhanced properties are being used. For example accountable ring signatures are used in order to prohibit double spending.

As for the cryptocurrencies, different cryptocurrencies use different protocols based on various types of ring signatures. For istance CryptoNote protocol is based on the traceable ring signature scheme. The goal of using ring signatures may vary as well. For example ShadowCash uses traceable ring signatures to hide the identity if the sender, but Monero, on the other hand, uses ring signatures to hide the real value of a transaction.

Although ring signatures are really powerful tools for enhancing privacy, they are not flawless. For example the cryptocurrency ShadowCash managed to cause a leak of the identities of a considerable number of its users, because of a poor implementation of the traceable ring signatures that caused a liability in the system. Luckily only 20% of all the keys in the system were affected by the bug and even though the sender's anonymity has been compromised, the receiver remained incognito.

7.2 Group Signatures

A group signatures scheme is a method that allows a member from a group to sign a message on behalf of the entire group. The concept was initially proposed by Chaim and van Heist and generally can be treated as a ring signature with less strict and powerful privacy measures.

Group signatures imply that a group manager needs to be designated that would act as an overseer of the group. It can add or remove group members and revoke the anonymity of each individual signer.

Usually group signatures are not being applied in blockchain networks, because of the centralized role of the manager, which contradicts with the main philosophy of the blockchain architecture.

7.3 Zero-knowledge Proofs

Zero-knowledge proof, also known as zero-knowledge protocol, is a method that allows one party to prove to a second one that a given statement is true, while not disclosing any additional information about the statement apart from its validity. The method implies that it needs to be trivial to prove that one possesses a piece of valid information by simply stating the fact and not sharing the information itself.

In a zero-knowledge proof, both parties (the prover and verifier) engage in a series of interactions in order for the prover to try to prove the validity of the knowledge and the verifier to contest the proof for determining if the prover really knows the information.

As is the case with the ring signatures, the main advantage for zero-knowledge proofs is the concealment of certain parts of the communication or transactions between two users, while guaranteeing the authenticity of the messages that are being communicated. The power of zero-knowledge proofs is properly displayed when it comes to transferring sensitive data, as the participants of the communication want to keep the information concealed, but at the same time need to have a rock solid proof of its validity. This property implies that zero-knowledge proofs see various implementations in domains like authentication, secure communication and blockchain technology.

A protocol implementing zero-knowledge proofs must necessarily require interactive input from the verifier. This requirement is due to the fact that the verifier may challenge the prover with different requests and then if the responses from the prover are true statements, the verifier is convinced that the prover really possesses the knowledge. If it is not the case and the verifier didn't challenge the prover with the right requests or the prover hasn't given the right responses according to the verifier, then the prover has failed the test.

The Zero-knowledge nature of the protocols also imply an interesting property:

Even though one can record the whole validation process between a verifier and a prover, thus may try to bring it as evidence for a zero-knowledge proof to a different verifier, the record must not be regarder as a valid evidence. This is due to the fact that without achieving any information about the prover's knowledge itself, we can never be sure that both the prover and the verifier where in fact honest and didn't "cheat" while validating the statement. Therefore a new validation process is required vor every new verifier.

To solve this problem, a trusted third party might take the role of centrelized validator that would verify all the provers' statements and give them a label of true or false depending on the authenticity of the statement. This solution though contradicts with the blockchain's idea of decentralization.

A zero-knowledge proof needs to satisfy three basic properties:

1. **Completeness**
2. **Soundness**
3. **Zero-knowledge**

Completeness ensures that an honest prover will be able to convince an honest verifier that he owns the information, if and only if the prover finds the statement as being true.

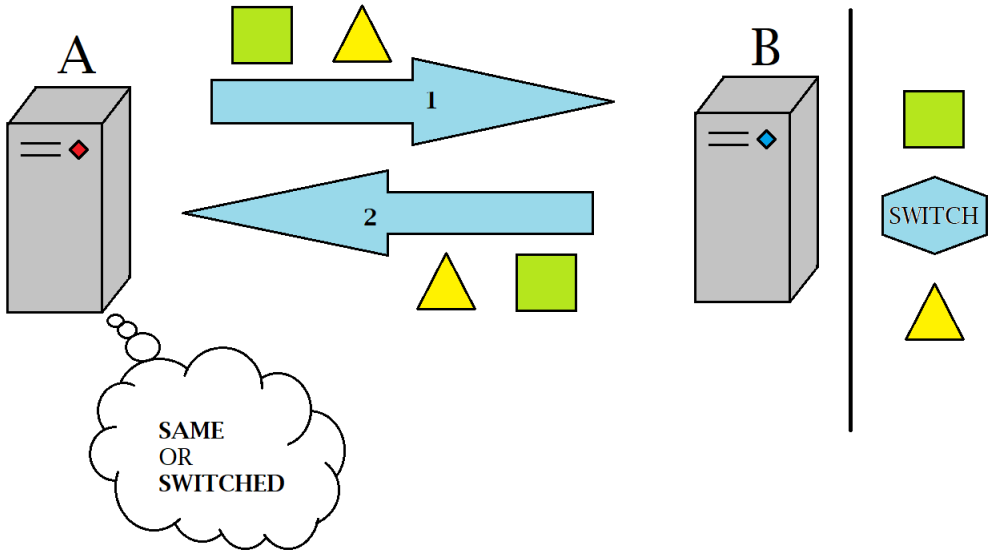
Soundness means that it is impossible, except a small error margin, for a cheating prover to convince a verifier that he possesses the knowledge.

Zero-knowledge implies that no verifier is able to learn anything more than the statement itself.

7.3.1 Practical Example

Let's imagine a scenario where a zero-knowledge protocol might be applied in order to better understand its specific properties.

Suppose we have two computers: A and B. The computer A is capable of generating 2 objects: a pyramid and a cube; The computer B doesn't know what a pyramid or a cube look like, but is trying to verify if computer A is capable of producing those objects, thus knowing what they look like. How can we describe a zero-knowledge proof protocol that would allow computer B to validate A's knowledge regarding the two objects?



A solution would be for the computer A to provide both objects to B. B secretly permutes the objects in any way it wants and then challenges A to tell if the order of the objects has changed or if it remained the same. This way if A makes a good guess, there is a $\frac{1}{2}$ probability that he really

knows what a cube and a pyramid is and can distinguish the two objects. A $\frac{1}{2}$ probability might not be sufficient so the computer B would perform the test several times in a row, this way the rate of error decreases and A's "luck" becomes more and more irrelevant. After n executions of the test the error margin would be equal to $\frac{1}{2^n}$.

Here is a more formal description of the testing protocol:

Let P = pyramid, C = cube, A = computer A, B = computer B

- Input: $A(P, C) => B()$
- Permutation: $B(P, C) => B(C, P)$ or $B(P, C) => B(P, C)$
- Knowledge check: $B(P, C)?\, A(P, C)$ or $B(C, P)?\, A(P, C)$
- Repeat if necessary

Now let's see how good of a solution this one is:

First of all, a good Zero-Knowledge algorithm should satisfy the three basic properties: Completeness, Soundness and Zero Knowledge.

Secondly, the Zero-knowledge aspect of the algorithm implies that a recording of the current validation process must be able to prove the statement to another verifier.

Let's start with the **completeness** property: suppose the computer A indeed owns the information regarding the differences between the objects, then after every test it will 100% of the times tell the right answer to the B's challenge, regardless of the permutation. In other words, A will always know if B has switched the two objects or not. If that is the case, then B, regardless of how many tests have been performed, will conclude that A indeed possesses the knowledge.

The **soundness** is achieved by drastically increasing the number of tests performed by B. For instance, if B is performing only 2 consecutive tests on A, then the probability of A not possessing the information but still randomly managing to give the right answers is a considerable $\frac{1}{4}$. As this value is too large to be acceptable, the test must be performed several times. For example if B interrogates A for 20 iterations then the probability of A giving the right answer for every and each of the tests, even though it doesn't know what are the differences between a pyramid and a cube, and is simply choosing randomly, is a negligible 1/1048576. So the soundness is achieved by increasing the number of executed verifications.

The **Zero Knowledge** property is guaranteed by the low number of objects and options for both A and B. To be more specific, considering that A is passing to B only two objects and the only action that B can perform is to permute (swap the order) of any of the objects, then any one permutation over one of the objects will definitely affect the other one, hence B is not able to set a constraint in order to delimit the Cube from the Pyramid. So there is no way B can get any information about what is the difference between a cube and a pyramid throughout the validation protocol and informational security remains intact.

By extending B's possibilities, we can create a security breach. For example, suppose B may not only permute the objects, but also show different ones instead. This way let's imagine B has a predefined set of a pyramid and a cube, for simplicity we will refer to them as prototypes. The prototypes look exactly like a regular pyramid and cube, but have a label only known to B that tells it what type of object is that. Remember that B can not distinguish the two objects by shape. After A gives the pair of a cube and a pyramid to B: (O_1, O_2), B then shows A the prototypes instead (P_1, C_1). To A they look exactly the same so it tells whether they are switched or not. If A's answer is positive, then B just needs to link the first object to the second prototype and the second object to the first prototype. This way B gains the knowledge about which of A's objects is a cube and which is a pyramid.

7.3.2 Applications of Zero Knowledge Proofs

Besides its importance to blockchain systems, zero-knowledge proofs are also applied to various different fields in science and technology. The most notable application areas are the following:

- Blockchains
- Nuclear Disarmament
- Authentication Systems
- Ethical Behavior

As we can see, due to the zero-knowledge proof protocols' special property of not disclosing any information while assuring its validity, offers a considerable and important benefit. Starting with Ethical Behavior, which refers to enforcement of honest and rightful behavior while preserving the anonymity and ending with the Nuclear Disarmament, where the zero-knowledge proof protocols help with confirming whether an object is a nuclear device or not, without sharing any information about the internal structure and functioning details.

8 Secure Multi-Party Computation - Erhan Adrian 3A6

The secure multi-party computation is a security definition that tries to model in real life a protocol that would simulate an ideal functionality. The main idea is as follows: there should exist a trusted third party (TTP) that would be able to get the input from any user and securely compute it and return the result. The communication would be done via private channels and every user would receive their own expected result.

The real world implementation replaces the TTP with a cryptographic protocol that aims at protecting input-output privacy of players and the correctness of the computation.

At its core Multi-Party Computation focuses on ensuring the fact that the real world communication and the various scenarios that might happen are as secure as the idealistic model. This is achieved by showing that any real world behavior can be replicated in the ideal world, but since the ideal world guarantees security, the real world is secure as well.

8.1 Formalized statement

In the secure computation there are N participants $p_1, p_2, p_3, \ldots, p_n$. Every participant has some secret input data $d_1, d_2, d_3, \ldots, d_n$. All the participants' goal is to compute the value of a function $F(d_1, d_2, d_3, \ldots, d_n)$ that is known to everyone of them. It is presumed that among the participants the are some "dishonest" players, they follow all the right instructions of the protocol, but in the process are trying to obtain additional information.

8.2 Security requirements

There are three main security demands regarding the Secure Multi-Party Computation protocols:

- Confidentiality
- Correctness
- Guaranty of information reception

Confidentiality means that no participant to the computation should be able to obtain any additional information from the protocol, than they already owned.

Correctness refers to the guarantee that every participant will receive the correct data.

Guaranty of information reception implies that no participant should be able to interfere and block the data transmission to any other participant.

8.3 Application in blockchains

The major impact of Multi-Party Computation in blockchain technology is related to the issue of long common reference strings required for zk-SNARKs.

Due to the fact that the generation of the common reference string is fundamental for the whole scheme, a violation in its generation may compromise the security of the whole system. Thus any doubtful behaviors must be intercepted. This is why nobody should rely on a "trusted" player to generate the common reference string since this would once again centralize the trust, which contradicts with blockchain's philosophy.

Multi-Party Computation procedures come to fix this issue. It offers a way to allow many volunteers to join the computations in order to generate the common reference strings. The only requirement for the strings to remain secure is that at least one participant needs to correctly compute his respective steps in the protocol.

9 Conclusion - Erhan Adrian 3A6

The blockchain technology is one of the most distinguishable and appealing innovations of our time. The idea of decentralizing trust between two or multiple agents has found a lot of implementation cases in different areas of human activity. It has enforced security and confidentiality properties like authentication, integrity and non repudiation, which further increased the appeal of this technology.

Although the blockchain is powerful and offers a lot of possibilities, it implied a need in an increased level of security, due the confidential aspect and the vast amount of information that is transported via the blockchain. This need required a lot of different techniques and protocols of Informational Security to be applied. Starting with hash functions and merkel trees, ending with zero-knowledge proofs and ring signatures.

As we could see, throught the years, the security protocols involving blockchain technology have evolved, the algorithms became more efficient and the level of security was constantly increasing.

The application of cryptographic techniques is very diversified and in most cases different protocols and properties are used together in order to ensure a desirable and necessary level of digital security.

The flow of information, its confidentiality, the anonymity of the users, the guarantee of a complete and valid result, all of these aspects lay at the foundation of the cryptographic principles that are used in blockchain technologies.

In conclusion, after covering the main aspects of cryptographic tools for blockchain technologies, we can truly appreciate the power of those security measures, while at the same time understand the shortcomings and the areas of improvement.

10 Bibliography

- Privacy-Enhancing Signatures: "Principles of Blockchain Systems" - Antonio Fernandez Anta, Chryssis Georgiou, Maurice Herlihy, Maria Potop-Butucaru
- Ring signatures: `https://en.wikipedia.org/wiki/Ring_signature`, `https://link.springer.com/chapter/10.1007/11681878_4`
- "But how does bitcoin actually work?" `https://www.youtube.com/watch?v=bBC-nXj3Ng4`
- "Data corruption and Merkle trees" `https://www.youtube.com/watch?v=rsx1nt2bxf8&list=LL&index=3`
- "Wikipedia Blockchain" `https://en.wikipedia.org/wiki/Blockchain`

- Zero-knowledge proof: https://en.wikipedia.org/wiki/Zero-knowledge_proof, https://link.springer.com/article/10.1007/BF02351717, https://www.youtube.com/watchv=5qzNe1hk0oY&t=227s&ab_channel=ComputationalThinking
- "Zero-Knowledge Proofs: STARKs vs SNARKs": https://consensys.net/blog/blockchain-explained/zero-knowledge-proofs-starks-vs-snarks/
- "zk-SNARKs: A High-Level Primer": https://medium.com/clearmatics/zk-snarks-a-high-level-primer-7cc
- "Fiat-Shamir transformation": https://www.zkdocs.com/docs/zkdocs/protocol-primitives/fiat-shamir/
- "Commitment schemes": https://www.youtube.com/watch?v=4w_b8Msxy14