Cryptographic tools for blockchain

Corina Dimitriu, Bianca Buzilă, Leonard Rumeghea

Contribution of authors:

- *Corina Dimitriu* contributed to the entire sections 1, 2, 3 and 4-Introduction, 4.1 and 4.2 with research, information selection, writing and own explanatory schemes
- *Bianca Buzilă* contributed to the entire sections 4.3, 4.4, 4.5 and 5, 6, 7 with research, information selection and writing
- *Leonard Rumeghea* contributed to the entire sections 8, 9 and 10 with research, information selection and writing, added research to section 4.2 and Power Point design ideas

1 Introduction to blockchain

1.1 General description of blockchain technologies

Blockchain technology came into being through the financial domain as a means to transfer digital resources between two endpoints without intervention from a central unit. In the financial field, these transfers incontestably refer to money, while in other work areas, such as medicine, education (and, easily to infer, in medical education) and IoT, they usually translate into efficient, secure, tamper-proof and faithful exchanges of undisclosed information (between doctors and patients, students and professors etc.). With the help of peer-to-peer networks and consensus algorithms, blockchain technology provides transparency and speed in gathering worldwide distributed information for the purpose of (statistical) valid data analysis.

The blockchain network is made up of virtually interconnected (by contract) endpoints with access to a distributed, decentralized database. According to its basic architectural principles, blockchain does not have an almighty owner or controller, which makes all the nodes in the network both activity monitors and activity executors: since the same copy of data is stored and updated in all the constituent nodes, data mining and transactions are allowed for all endpoints, as well as validating each other's requests and responses. From those previously mentioned, one can easily notice the influence of the basic principle of peer-to-peer networks: nodes act as clients and servers at the same time. Therefore, decision making may be considered "inclusive" (the majority of endpoints bears responsibility for it) and the probability for the system to be broken by attackers is minimized. Needless to say, the blockchain architecture requires powerful computers, with large storage capacity, since this is the only method to handle the situation in which the database will be accessed by several nodes at the same moment in time.

If we were to go back in time, bitcoin was the firs and the first and the most well-known project implemented on blockchain. It was invented as a solution to the inherent problems of the traditional banking system, namely

Figure 1: Network architectures

additional costs payed by customers to the banks for acting as central authorities between parties in transactions, long waiting times for international payments, slowness and lack of transparency in information delivery services, which are inevitably delayed by the authentication and manual authorization protocols. According to the bitcoin mining policy, users (associated to computers) posses private keys in order to identify themselves when announcing in the network the exact amount of bitcoins they aim to send. The process of sharing transaction details with the other endpoints may be perceived as an integrity guarantee. In addition, the confidence of participants guarantee is given by the unchangeable and irreversible nature of transactions (*Treleaven, Gendal Brown and Yang (2017)*).The only delay that applies to the transaction is given by the time required by the other nodes to update their data copies accordingly.

To get into structural details, the blockchain architecture which was generally described above, comprises more than one layer. The number of layers depends on the system's practical area of usage, but having the next four of them is usually considered common practice:

1. Data layer: comprises blocks of transactions, more specifically takes care of the arrangement of transactions in blocks; this is where the hash functions and merkle trees come into play to provide the account information being shared with immutability;

2. Network layer: checks for the validity of the transactions being shared in the network; also takes care of distributing and updating data in all nodes through corresponding communication protocols, by using routing and addressing mechanisms;

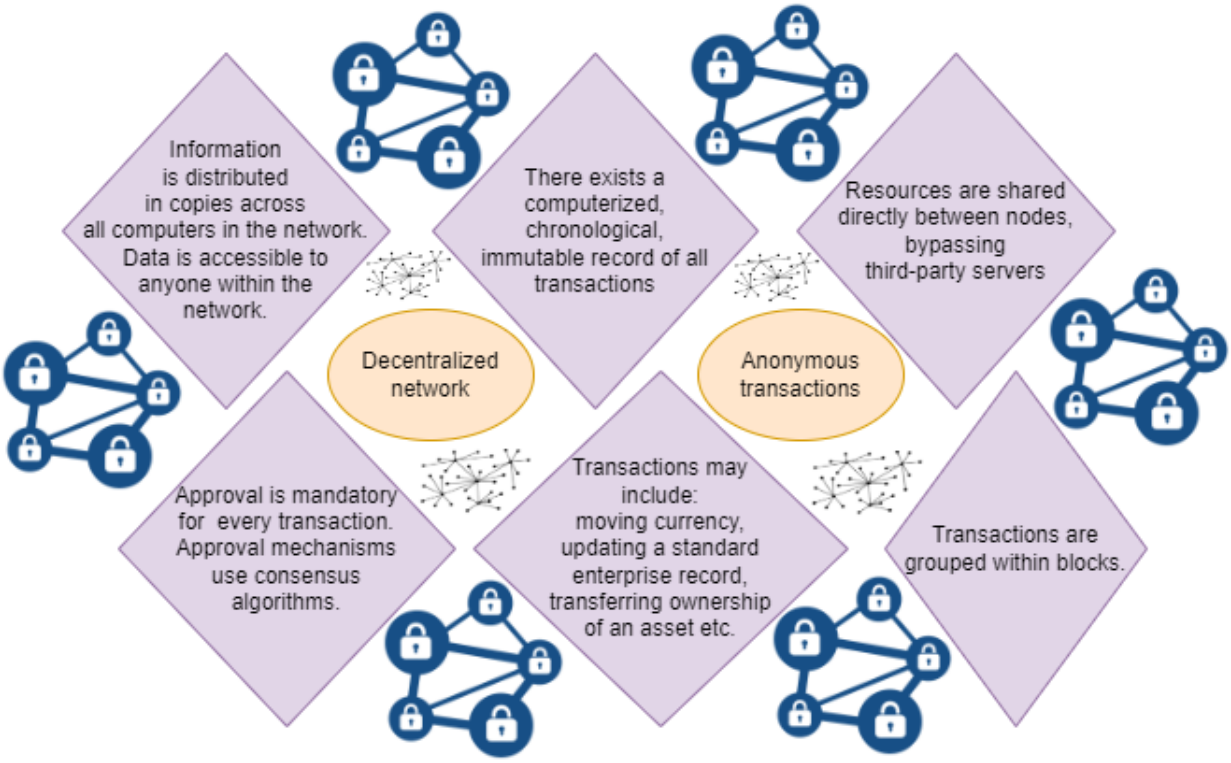
3. Consensus layer: refers to the consensus algorithms being used (will be described later in this paper);

4. Application layer: takes care of the usability and interface with the user;

It is worth mentioning that some studies support adding two more layers between the consensus and the application layer, namely the incentive layer and the contract layer. The incentive handles planning rewards to the participants in the network when meeting the system's desired objectives, while the contract layer comprises scripts for implementing communication between the user interface and the actual network.

Iuon-Chang Lin and Tzu-Chun Liao, 2017 define a more security-oriented perspective over the blockchain architecture, describing six elements which form the foundation of blockchain:

Figure 2: Synthesis - key-concepts in blockchain technology



* Decentralization: refers to the peer-to-peer network already detailed above
* Immutability: editing is not allowed once the information is set and sent, since the transactions are chronologically journaled in the huge ledger that blockchain defines itself through; the hashing process of a new block includes metadata from the previous block's hash output, which makes the chain of transactions immutable
* Transparency: nodes in the network have equal statuses as far monitoring is concerned
* Anonymity: transactions are anonymous, except for the user's blockchain address which is shared for reliability reasons
* Distributed ledgers: the majority of the endpoints must check every new transaction/block for validity
* Consensus: intervenes in the decision masking processes, based on majority vote

1.2 Blockchain in terms of information security

Blockchain system's security is usually discussed with respect to the CIA (Confidentiality, Integrity, Availability) triad, in the context of some specific concerns regarding the blockchain structure. The proper functioning of blockchain technology relies on its potential to mantain data privacy and starts from the premise that all members of the network adopt an honest behaviour. However, it is worth knowing that the layered blockchain structure can lead to vulnerabilities which have been already succesfully exploited by attackers.

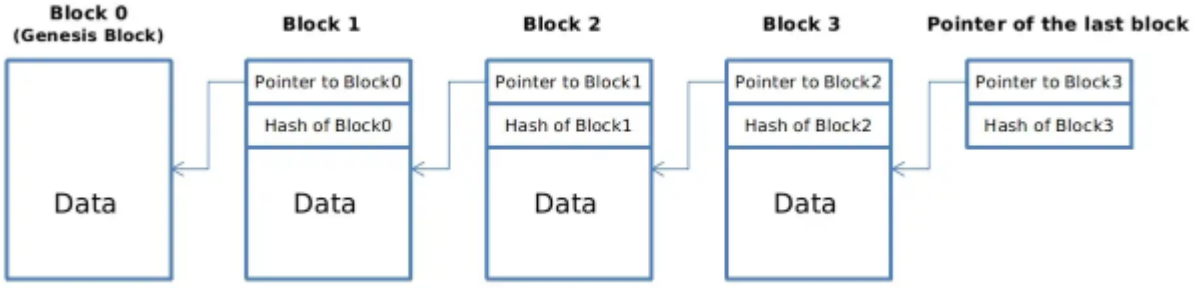
Most implementation policies stand out for ensuring mainly security and privacy within the use of blockchain technologies. However, these two characteristics also involve a couple of adjacent relevant features, such as integrity and authorization. The validation operation, performed before a transaction is written to the database, is based on hash functions and digital signatures, in the presence of public and private keys. Signatures are used for the purpose of identifying documents by the node they belong to / are sent from. The system is therefore prevented from being vulnerable to Denial of service attacks. Hash functions play a significant role in chaining transaction blocks sequentially, in order to preserve immutability and authenticity of data. Almost obviously, they intervene in implementing the consensus algorithms as well. The detailed mechanisms which make use of the above mentioned cryptographic tools, altogether with verifiable random functions and non-interactive zero-knowledge proofs, are described in the next sections.

2 Hash functions in blockchain

By definition, hash functions are functions that map arbitrarily large bitstrings to fixed-length bitstrings (ussually considered "short" and consisting, for example, of 128, 256 or 512 bits). Only collision-resistant hash functions are of interest are far as the content of the present paper is concerned, i.e. those functions H for which, given the key (if defined by the hash algorithm) and the output length, no adversary can compute a pair $(m0, m1)$, with $m0 \neq m1$, such that $H(m0) = H(m1)$ (or $H(K, m0) = H(K, m1)$ in case the hash computing algorithm requires the use of a (pseudo-) randomly generated key) with a higher than negligible probability. For simplicity reasons, in case studies which do not have a focus on hash functions, cryptanalysts often associate hash functions with perfectly random functions, even though the associations is not entirely accurate or rigorous. This technique is known as the *random oracle model*.

In the context of blockchain technologies, chaining the transaction blocks in order to form the distributed ledger requires the use of hash functions, as the hashing process of a new block involves the possession of (metadata from) the previous block's hash output. The next paragraph goes into further detail regarding the validation operation, carried out by verifying correctness of block sequences, which gives an insight into the reason why blocks are actually chained. Moreover, Bitcoin and other cryptocurrencies uniquely identify coin transfer participants through hashes of their public keys, which are conceptually transformed into endpoint addresses. The uniqueness of the address is guaranteed by the collision-resistance property.

Figure 3: The linked list called blockchain (pointer refers to "hash pointer")



Filling a transaction block is immediately followed by chaining it into the ledger of the targeted system.

A transaction block contains a header and a body, the latter being represented by a list of transactions. A nonce is added to the head of the block in order to increase the difficulty of finding connections to block hashes in the sequence from outside the system. The nonce serves as a decision instrument to the unbiased selection of the "validators" from the other nodes in the network: miners are supposed to make a "collective effort" to guess the nonce (the guess is valid if adding it to (one of) the previous block's header and hashing it results in the hash of the target block). The first to guess is the one to be rewarded and the one who adds the block he has just discovered to the linear chain. Individual guesswork is considered to be unfeasible due to computational power reasons.

Figure 4: Anatomy of a transaction block

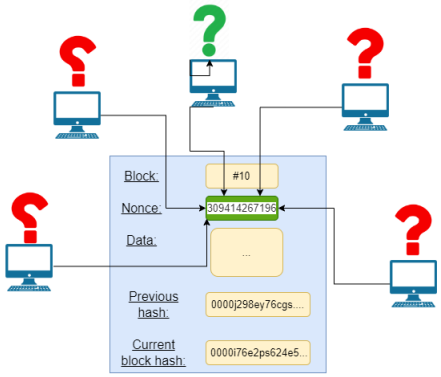
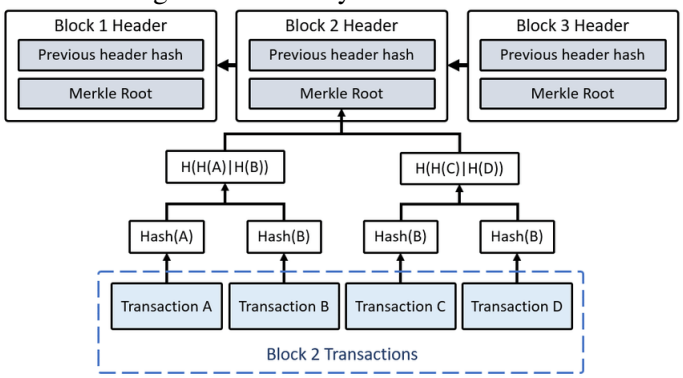


Figure 5: Anatomy of a Merkle Tree



The header mainly contains data and the hash of the previous block, both part of the validation process. Data refers to transaction *metadata* and comprises information which is useful in identifying the sender, the receiver and other particular details depending on the targeted work area (such as the amount of bitcoins to send). These information stored in the data field is used by the other nodes to detect old transactions concerning the same issuer at the validation step (and check, for example, if he possesses the amount of money he pretended to have when issued the current transaction). The validation step therefore results in participants indirectly validating the entire chain. The fact that each block contains the hash of the previous block makes adding a block depend on all the previous ones. Also, it is computationally impossible to edit a previous block without reversing the chained hashing process, which is in turn impossible due to the one-wayness property of good hash functions; this is beneficial for the proper functioning of the system since altering an already existing block would break the trust in all functionalities performed in the chain. The header also includes the root hash of the *merkle tree*, which is the subject of the next important highlight.

3 Merkle trees in blockchain

3.1 About Merkle trees

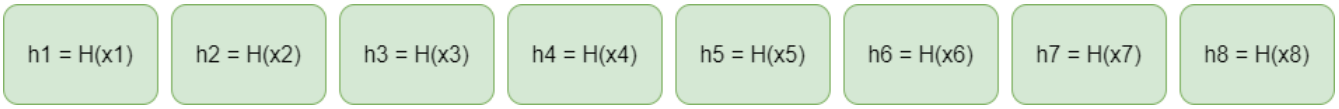
Merkle trees were introduced in the early 80's by the public-key cryptography specialist Ralph Merkle. A Merkle tree might be seen as a (hash) function that requires n parameters as input and outputs a *Merkle tree root hash*. The tree is usually denoted by **MHT**, while the root hash is usually written as $\mathbf{h} = MHT(x_1, x_2, ..., x_n)$. Considered hash functions themselves, Merkle Trees make use of a collision-resistant hash function (in the classical sense) in their construction, which will be further noted as $H(x,y)$ or $H(x|y)$. Both notations refer to the hash of

the concatenation (in the sense of joining the characters) of two previous hashes.

As very eloquently depicted by Alin Tomescu in his study, *What is a Merkle Tree?*, the example below represents the way a Merkle tree which takes 8 files/data containers as input is built:

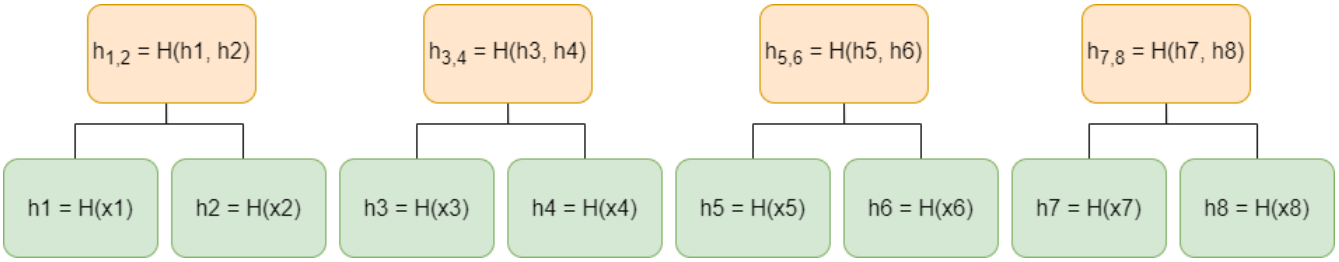
Firstly, each item (file/data container) is hashed and these results are obtained:

Figure 6: First stage of building Merkle trees



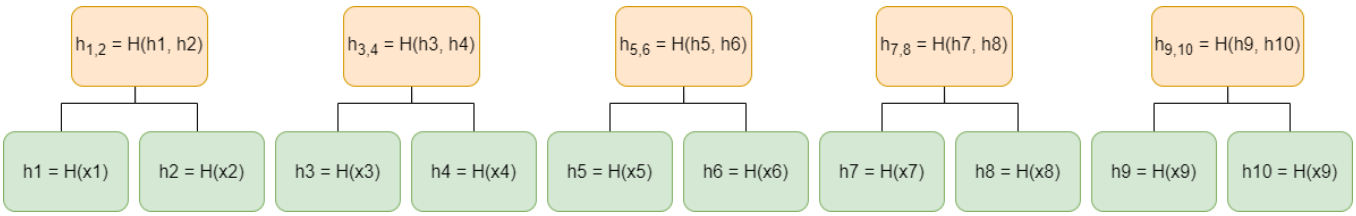
Secondly, pairs of hashes are formed by grouping every two adjacent previously obtained hashes, starting from left to right. If there is an odd number of hashes on the previous level (on any level of the tree except the root), the rightmost previously obtained hash is duplicated. Hashes in pairs are concatenated and the hash function is then again applied on the concatenation result. The next construction shows the effect of applying this operation:

Figure 7: Second stage of building Merkle trees



If we would have had an odd number of file inputs, this would have been the result of applying this step:

Figure 8: Second stage of building Merkle trees with odd number of inputs



The previously described step is repeated until a single hash is obtained on the top of the tree: this hash is called *the Merkle root* or the *root hash*:

Figure 9: Complete Merkle tree

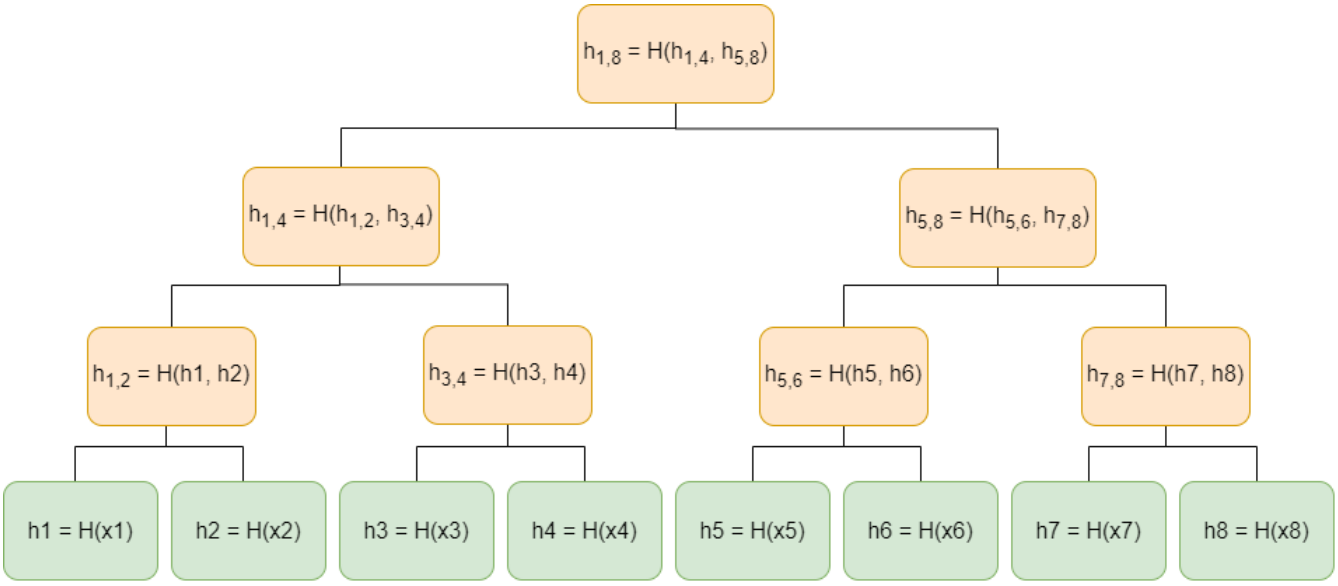
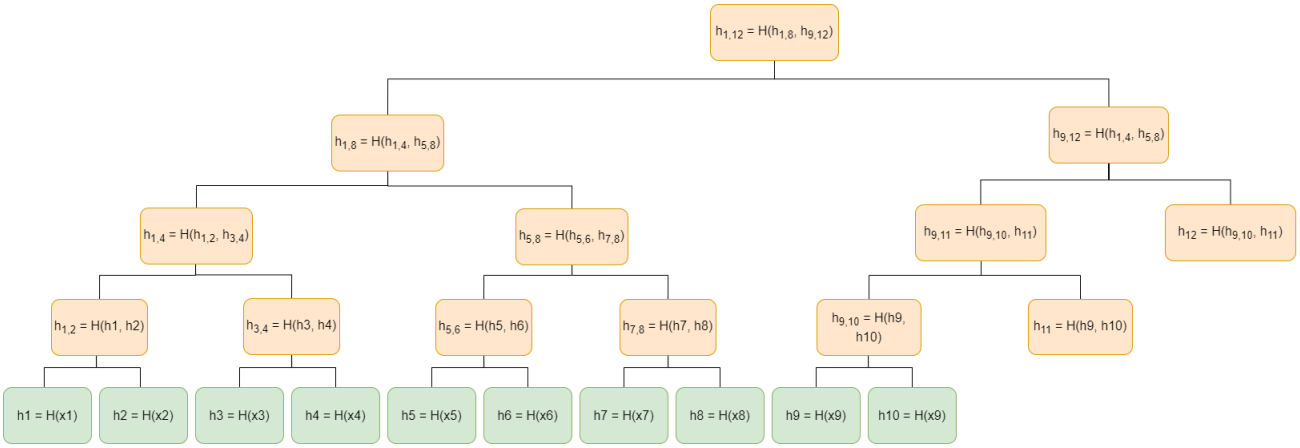


Figure 10: Complete Merkle tree with odd number of inputs



The readers of this paper might now be able to analogously construct their own Merkle tree starting from an arbitrary number **n** of data containers seen as variables $x_1, x_2, ..., x_n$, considering H function given and the simple computing algorithm described above.

3.2 About Merkle proofs

The basic and most significant use of Merkle trees refers to the file integrity problem. A practical perspective regarding what is going on behind when the average user uploads a couple of files on Dropbox (or other storage-oriented platforms) reveals that they also indirectly help solving the storage problem. More precisely, the scope of the user (and the system) involves, in most of the cases, to delete the files after the upload is successfully finished. But the *root hash* is worth being kept, in order to check integrity when downloading is performed. The reader of this paper might wonder why keeping the hash of the files and checking equality at download is not

a suitable approach, since the hash function is collision-resistant, as mentioned from the very beginning of this section. Besides comparison, this last solution only requires rehashing at download and is, probably, more easy to piece together, but can't be enforced in the context of millions or billions of files stored. Note that the height of the tree is included in $O(log(n))$. Eventually, it all boils down to searching in a Merkle tree and this is how it is done:

The *root hash* is somehow made public so as it is known by both the personal computer and the platform. After download, the receiver (the computer, in our case) asks for a *Merkle proof*. A **Merkle proof** is defined as an incomplete representation of the Merkle tree associated to the uploaded files. Taking the same example as in the previous subsection, this Merkle proof could have the following structure, assuming that the third file/data container is being questioned: $h1$, $h_{1,2}$, $h_{5,8}$ are already known and form the smallest number of nodes needed to be passed on to the user in order for him to calculate the root hash. This minimum length set of already given nodes is called the *Merkle path* for the leaf $x3$. The remaining hashes can be computed in this order:

* $h3 = H(x3)$
* $h_{3,4} = H(h3, h4)$ ($h4$ is already known, $h3$ was previously computed)
* $h1,4 = H(h_{1,2}, h_{3,4})$ ($h_{1,2}$ is already known, $h_{3,4}$ was previously computed)
* $h1,8 = H(h_{1,4}, h_{5,8})$ ($h_{5,8}$ is already known, $h_{1,4}$ was previously computed)

Figure 11: Merkle proof

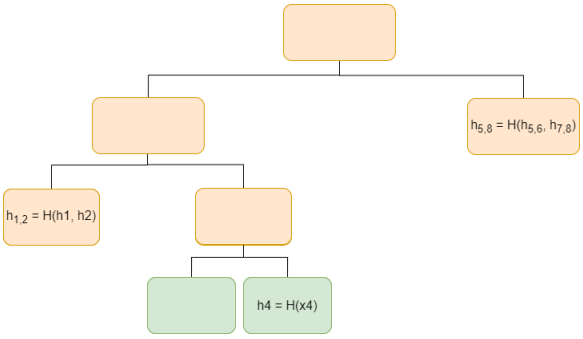
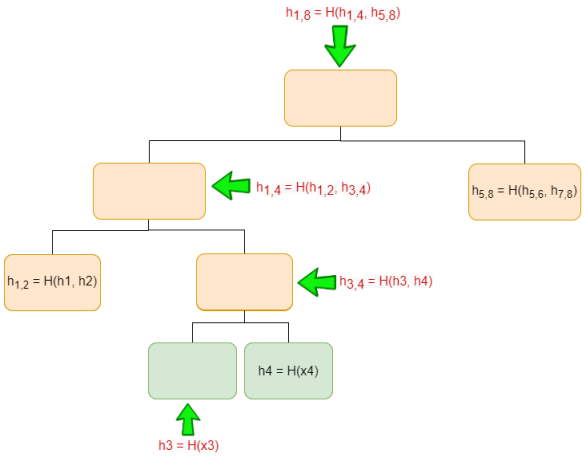


Figure 12: Merkle proof after 'filling in the gaps'



The key in the Merkle proof's structure is given by the fact that the information they provide suffices to re-compute all the missing hashes, all the way up to the *root hash*, which is therefore compared to the pre-existing *Merkle root*, which was made public during the upload process, as stated before.

Obviously, checking equality between the roots ultimately decides the file's integrity.

Moreover, if the Merkle proof validates that $x3$ was indeed the third input for the **MHT** function, no attacker can pretend there exists a Merkle proof which validates $x_j, j \neq 3$ as the third input of the **MHT** function (with higher than negligible probability), due to the collision-resistance property of the **H** function, chained multiple times to result in the **MHT** function (which is therefore collision-resistant as well).

As expected, Dropbox and other storage platforms do not constitute the only use-case of Merkle tree proofs. Efficient validation of integrity is inherent in many kinds of (complex) application architectures (with possibly millions of users and file resources), which causes Merkle proofs to spread from GitHub - where local storage synchronization with the platform is done through commits requiring speed in data transfers - to blockchain technologies - where daily stock trading activities are thoroughly tracked and recorded.

3.3 Merkle trees and blockchain

In the context of blockchain technologies, proving that a transaction belongs to the Merkle root which is stated in its header is achieved through the use of Merkle proofs. In other words, the leaves are authenticated by the Merkle root of the Merkle tree they belong to. The leaves are represented by transaction specific data, such as coins (as in the case of the Zerocash protocol). There are several viable techniques that Merkle trees make available to blockchain technologies so as to fulfill their security, performance and operational constraints:

* A change in one transaction in the record (and, thus, in that transaction's hash) would trigger changes in all the hashes of the above tree nodes, including the root. This would lead to invalid data in a whole chain of tree nodes, which is why modifying previous transactions is an impossible operation with respect to the immutability property of blockchain systems.

* Applying Merkle proofs reduces the amount of memory space needed, by storing hashes for verification purposes instead of entire data blocks and, also, the amount of computing power, by validating transactions in logarithmic time with respect to the number of data blocks. Otherwise, without enforcing Merkle trees, comparing entries in the data blocks character by character would have been necessary to ensure the register in the endpoint's database matches with the network ledger, since the data itself and its proof would not have been separated by implementation anymore.

* Double-spending [1] as forgery attempt cannot pass unnoticed through the network as long as the hash of the falsified transaction in that endpoint does not match the rest of the network hashes.

All in all, Merkle trees succeed in creating a digital fingerprint for the entire collection of transactions, so that all the nodes (as endpoints/miners) in the network, both full and lightweight [2], stay up-to-date with consistent information.

4 Digital signatures

Digital signatures create an honest communication framework around the documents or messages that "travel" from the sender to the receiver, as they play the role of an authenticity ticket for the transmitted data. Taking into

[1] Double spending occurs when somebody tries to insert or alter a transaction in order to regain the cryptocurrency within it (adapted from `https://corporatefinanceinstitute.com/resources/cryptocurrency/double-spending/`)

[2] Full nodes are the ones holding all blocks in the blockchain history of a particular system, in their entirety, while the lightweight nodes only keep their headers and the Merkle root in order to validate transactions within that block (adapted from `https://medium.com/techskill-brew/merkle-tree-in-blockchain-part-5-blockchain-basics-4e25b61179a2`)

account an environment which requires the participation of two entities: the *signer* and the *verifier*, the former signs the message with his private key and the latter verifies its authenticity using the public key; under these circumstances, the sender cannot deny having sent the document (so the non-repudiation property is preserved).

As far as blockchain technologies are concerned, digital signatures are called into action exactly due to their ability to indicate non-repudiation and authentication regarding the ownership of the message. The hash of the message is signed before being sent in the network, from where the signature acts as a digital fingerprint for that particular transaction filled with data.

4.1 Plain digital signatures

Digital signatures fulfill their desired role in terms of information security and, particulary, blockchain systems, through:

- the interaction of three algorithms, namely:
  - key generation, which provides the secret (signing) key and the public (verification) key
  - signature generation, which provides the signature given the message and the secret key
  - signature verification, which is a deterministic algorithm proving the signature right or wrong depending on the message and the public key
- the assurance of two essential security properties, namely:
  - EUF-CMA: Existential Unforgeability under Chosen Message Attack

    Given the following scheme,

    Training phase:

    Attacker $\rightarrow$ *Oracle*: asks the oracle for signatures for a set of n messages of his choosing, denoted by $M = \{m_i \in \{0,1\}^* | i \in \{1,...,n\}\}$

    Oracle $\rightarrow$ *Attacker*: returns $S = \{s_i \in \{0,1\}^k | i \in \{1,...,n\}\}$, representing the corresponding signatures for the n messages

    Therefore, a pair set can be written as follows: $P = \{(m_i, s_i), m_i \in \{0,1\}^* and s_i \in \{0,1\}^k | i \in \{1,...,n\}\}$

    Challenge phase:

    Attacker $\rightarrow$ *Oracle*: provides a forgery, meaning a pair (m_j, s_j) such that s_j is the result of the signature generation algorithm if m_j is given as input and $m_j \notin M$, the set of messages queried before

    A digital signature is said to be **EUF-CMA** if no attacker can generate a forgery in the above conditions with higher than negligible probability.
  - SUF-CMA: Strong Existential Unforgeability under Chosen Message Attack

    Given the following scheme,

    Training phase:

    The same as the training phase for the *EUF-CMA* property.

    Challenge phase:

Attacker $\rightarrow$ *Oracle*: provides a forgery, meaning a pair (m_j, s_j) such that s_j is the result of the signature generation algorithm if m_j is given as input and $(m_j, s_j) \notin P$, the set of pairs queried before (the significant difference towards the *EUF-CMA* scheme is that m_j can be part of the set of messages queried before)

A digital signature is said to be **SUF-CMA** if no attacker can generate a forgery in the above conditions with higher than negligible probability.

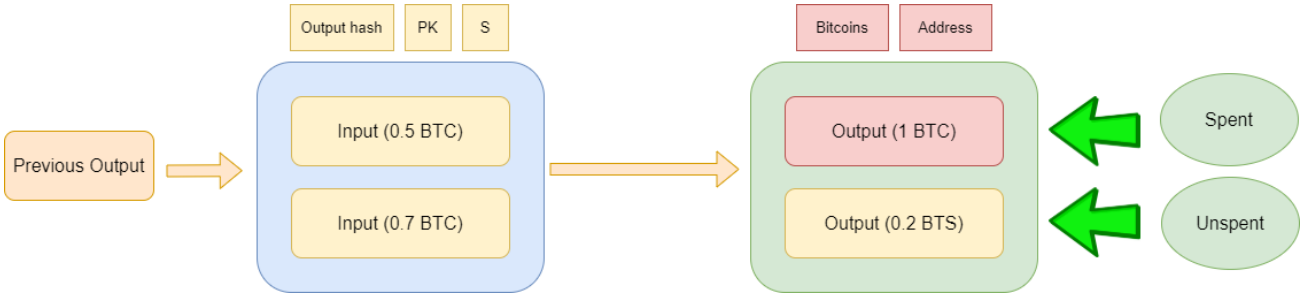
The most widely spread signature schemes are the ones based on the assumption that factoring is a difficult problem, so the RSA-based schemes. One of the preffered schemes in blockchains, the ECDSA, which is a variant of DSA (Digital Signature Algorithm), but on the basis of elliptic curves cryptography, was developed from the assumption that discrete logarithm computation is a difficult problem as well; this scheme is the one chosen by Bitcoin, whose example was followed by those from Ethereum. Among other schemes, the **BLS** (Boneh-Lynn-Shacham) scheme deserves special attention and is briefly presented in the next paragraph.

First of all, the concept of *Gap Diffie-Hellman group* introduces the contrast between the problem of computing g^{xy} given only g^x and g^y (the so-called Computational Diffie-Hellman problem, which is difficult) and the fairly simple problem which asks for validating or invalidating the equality between a given $\widetilde{g}$ and the product of g^x and g^y. Derived from here, the concept of *co-Gap Diffie-Hellman assumption* focuses on the contrast between the hardness of computing h^a given $h \in \langle g_1 \rangle, g_2$ and $g_2^a \in \langle g_2 \rangle$ and the ease of deciding whether $h^a = h^b$ given the same *co-Diffie-Hellman triple* (without going into more details, this is performed using bilinearity). The BLS scheme takes $x \in Z_q$, where q is the groups' order (the groups are $\langle g1 \rangle$ and $\langle g2 \rangle$), as the private key and g_2^x as the public key. The hash of the message is signed by raising it to the power of x. Along with g_2^x and g_2, the signed hash of the message forms a *co-Diffie-Hellman triple*. The secure character of the scheme comes from the fact that g_2^x and the signed hash can be used for signature verification, but re-computing the signature is difficult due to the *co-Diffie-Hellman assumption*.

Use of plain digital signatures in blockchains

The Bitcoin **UTXO** model (unspent transaction output) explains its name by each transaction being depicted as an input-output pair, in which the input represents the spent amount of coins and the output covers the target sum and the BTC that will be spent in future transactions, yet unspent. The input which traverses the transaction and exceeds the target sum turns into unspent output until entering other transactions in the system; the input possesses a reference to the output of the transaction it resulted from, denoted as *OutputHash* in the figure below. The input also includes the ECDSA signature (denoted as *S*) for the current transaction and the public key (denoted as *PK*) predestined to signature verification, while the output comprises the amount of bitcoins (which will be further referenced by some future input) and the hashed public key's address (included in the input) for signature verification by the targeted endpoint. More complex schemes make use of multi-input and multi-output transactions, which is the case of the figure below:

Figure 13: The Bitcoin UTXO model



Bitcoin gives its users the flexibility to decide the future of the remaining coins from the current transaction: for instance, purchasing an object which values 1 BTC with 1.2 BTC you have "in stock" results in two outputs: one of 1 BTC, which goes to the seller, and one of 0.2 BTC, which qualifies as unspent output and has to be be reused (becoming input) in one of the three following ways:

* it can be sent back to the current owner's account
* it can be converted into the transaction fee; placing a transaction fee as reward is compulsory in order for miners to place the 1 BTC transaction to a block in the system
* it can be sent to somebody else

Taking one of the above decisions causes the current transaction to end.

4.2 Aggregate signatures

Aggregate signatures enforce the combination of multiple signatures, on multiple messages, signed by different users, into a single representation. The first scheme was designed by *Boneh et al.* in 2003. In order to achieve that single, compact signature, the BLS scheme might be used. In addition to the previous subsection, a bilinear map $e : \langle g1 \rangle X \langle g_2 \rangle \rightarrow G$ [3], where G is a newly formed group, so as to dive into more details. Signing is done in the same way as previously described, but for each user in particular at first, and verification is executed by checking this equality: $e(s_i, g_2) = e(H(m_i), v_i)$. Considering n users, each one has a signing (secret) key s_i and a verification (public) key $v_i = g_2^{s_i}$. The aggregation step consists of computing the previously mentioned single representation as the product of the individual signatures. According to mathematical notation, $S \leftarrow \prod_{i=1}^{n} s_i$. Verification is carried out by checking if the equality

$$e(S, g_2) = e(\prod_{i=1}^{n} H((m_i)^{x_i}, g_2) = \prod_{i=1}^{n} e(H(m_i)^{x_i}, g_2) = \prod_{i=1}^{n} e(H(m_i), g_2^{x_i})$$

holds. However, there is a restriction that has to be checked before performing the above calculations: all messages must be different from each other, since the existence of equal messages might result in one user computing his public key at the expense of the other user having the same message (meaning, starting from the second one's public key he has already computed) and induce attack risks.

[3] Bilinear maps from $G_1 X G_2$ to G are functions such that for $\forall u \in G_1, v \in G_2, a, b \in \mathbb{Z}, e(u^a, v^b) = e(u, v)^{a \cdot b}$

If order constraints are enforced over the aggregation of the individual signing keys, then the signature specialises into a so-called *sequential aggregate signature*.

Use of aggregate signatures in blockchains

When it comes to data saved in multiple copies, aggregate signatures lead to significant storage space savings, mainly generated by the aggregation of all transaction signatures from each block into a compact signature.

4.3 Multi-signatures

Multi-signature is a specific type of digital signature that makes it possible for two or more users to sign documents as a group. A basic multi-signature scheme combines the multiple unique signatures into the final signature. The downside of such a signature is that its size increases linearly with the number of signers.

The security definitions for multi-signature schemes are based on the work of Micali et al from 2001. Thier scheme does not satisfy the requirements for today's applications in blockchains because it requires an interactive setup phase during which all possible signers interact, meaning that no new parties can join once the setup is complete.

There are many multi-signature schemes that allow non-interactive signing, meaning that each signer can independently generate their contribution to the signature any party can combine the contributions into the final multi-signature. Each contribution is generated like in a standard digital signature scheme and an additional algorithm is used to merge all contributions into the multi-signature.

Boneh et al. introduced in 2018 a new scheme which uses the key aggregation technique, where the verifier needs a short aggregation of all signers' public keys instead of all keys in order to verify the signature. They replaced the Schnorr signature scheme by BLS signatures. The scheme needs an efficiently computable non-degenerate pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_t$ in groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t$ of prime order q, with g_1 and g_2 the generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, and the hash functions $H_0 : \{0,1\}^* \to \mathbb{G}_2$ and $H_1 : \{0,1\}^* \to \mathbb{Z}_q$. The key generation algorithm chooses a random $sk \leftarrow \mathbb{Z}_q$ and output (pk, sk) where $pk \leftarrow g_2^{sk} \in \mathbb{G}_2$. The keys are then aggregated as

$$apk \leftarrow \prod_{i=1}^{n} pk_i^{H_1(pk_i, \{pk_1, \ldots, pk_n\})}.$$

For a given message m, each participant computes their share $s_i = H_0(m)^{a_i \cdot sk_i}$, where $a_i = H_1(pk_i, \{pk_1, ..., pk_n\})$. The final multi-signature is computed as $\sigma \leftarrow \prod_{i=1}^{n} s_i$. The verification is now checking if $e(\sigma, g_2) = e(H_0(m), apk)$.

Use of multi-signatures in blockchains

In the context of cryptocurrencies, the technology was first applied to Bitcoin addresses in 2012, which eventually led to the creation of multisig wallets, one year later.

By using a multi-signature wallet, users are able to avoid the issues caused by the loss or theft of a private key, so their money is therefore secure even if one of the keys is lost.

Multi-signatures can be used for creating two-factor authentication mechanisms. A user could have one private key stored in his laptop and the other one in his mobile device (or even on a piece of paper). This would ensure that only someone who has access to both keys is able to make a transaction or access the funds.

4.4 Threshold signatures

Threshold signatures are a specific type of multi-signatures. They were introduced by Yvo Desmedt and Yair Frankel in 1989.

A threshold signature scheme (TSS) is a method for generating a single digital signature from multiple signers. It is created with multiple private key shares, which are distributed such that no single person controls the private key entirely. A TSS (n,t), where n is the total number of participants t is a previously decided upon threshold, cannot generate the signature unless any t of the n participants work together.

A common threshold cryptographic scheme is Shamir's secret sharing scheme (SSSS) developed in 1979, which is based on Lagrange interpolation. A secret s can be shared between n parties so that any t parties, where $t \leq n$, can reconstruct the secret, but it remains hidden to any $t-1$ parties. To implement this scheme, a random polynomial f of degree $t-1$ is chosen such that $f(0) = s$. Each party $i = \overline{1,n}$ will be given a secret $f(i)$. Any t parties can compute $f(0)$, i.e. reconstruct the polynomial, using Lagrange interpolation.

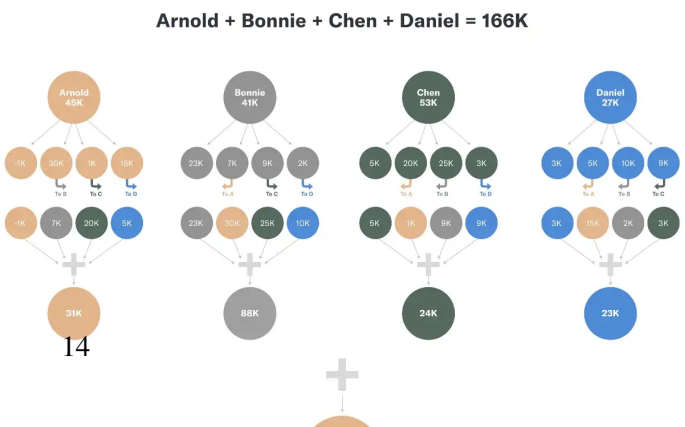
In the TSS, the shared secret is a signing key $x \in \mathbb{Z}_q$ of a BLS signature scheme, and each participant i receives a share $x_i \in \mathbb{Z}_q$. The initial distribution of keys can be achieved by using the distributed key generation protocol developed by Gennaro et al. Each participant i can compute $s_i \leftarrow H(m)^{x_i}$, i.e. a BLS signature with their key x_i, which represents their contribution to the signature. All the shares $s_{i_1}, ..., s_{i_t}$ are used to obtain the signature $s = H(m)^x$ by computing the exponent of $H(m)$, using multiplication and scalar exponentiation instead of addition and scalar multiplication.

There are two differences between TSS and SSSS:

1. **Key generation**:
   - In SSSS, there is one person responsible for creating the private key secret shares called "the dealer." It means that the private key is generated at a single location and is later distributed to different locations by the dealer.
   - In TSS, there is no dealer since the role is distributed so that the full private key is never at a single location.
2. **Signing**
   - In SSSS, the parties must reconstruct the full private key in order to sign, which results in a single point of failure each time a signature is needed.
   - In TSS, the signing is done in a distributed way and the secret shares are never reconstructed.

Example

An example that helps understand TSS was developed by BitGo [9], in which the purpose is to find the average salary of a group of people without revealing an individual's salary. There are 4 parties: Arnold,



Bonnie, Chen, and Daniel. Arnold is making $45,000, Bonnie is making $41,000, Chen is making $53,000, and Daniel is making $27,000 — adding up to $166,000 in total.

Each party partitions their salary into 4 random numbers that sum up to their individual salary like so:

- Arnold shares $30,000 with Bonnie, $1,000 with Chen, and $15,000 with Daniel, and remains with $-1,000.
- Bonnie shares $7,000 with Arnold, $9,000 with Chen, and $2,000 with Daniel, and remains with $23,000.
- Chen shares $20,000 with Arnold, $25,000 with Bonnie, and $3,000 with Daniel, and remains with $5,000.
- Daniel shares $5,000 with Arnold, $10,000 with Bonnie, and $9,000 with Chen, and remains with $3,000.

Next, they each assemble the shares they have been given by the others:

- Arnold sums up all the received numbers from each party with $-1,000, which adds up to $31,000.
- Bonnie sums up all the received numbers from each party with $23,000, which adds up to $88,000.
- Chen sums up all the received numbers from each party with $5,000, which adds up to $24,000.
- Daniel sums up all the received numbers from each party with $3,000, which adds up to $23,000.

Finally, all the parties come together and sum the final values they calculated, which add up to $166,000. They then divide it by 4 to find the average salary: $41,500.

Use of TSS in blockchains

In blockchains, a TSS is used to create digital signatures for cryptocurrency transactions. Some blockchains that use threshold signatures schemes are Bitcoin, Ethereum, Binance Coin, Solana.

Some advantages of choosing to use a TSS are:

- The private key is no longer held by a single point of failure.
- TSS transactions contain the same amount of data as a normal single signature transaction, so they are faster and cheaper to verify with lower transaction fees.
- Many different distributed key share combinations that represent the same private key can be easily generated, without the need to change it.
- The private key can be extended to new members who join the signing group without the need to expose any part of it.

4.5 Forward-secure signatures

The security of digital signatures is based on the fact that the private key is secure. Since a verifier cannot determine whether a signature was issued before or after the private key was exposed, all signatures generated by that key are considered invalid. If an attacker obtains an honest user's signing key, they can sign any messages they choose under that user's name. Until the user's certificate is revoked or expires, the attacker is able to access messages or resources on their behalf using the conventional public-key infrastructure (PKI)-based Internet protocols.

In blockchain systems, especially ones based on proof of stake, the problem is more complicated because:

1. the signature keys are used for two different purposes: spending the tokens and participating in consensus, and

2. the validity of the keys is not determined by external means but rather by the consensus itself.

If an honest user spent all tokens and stopped protecting the signature key, an attacker could be able to discover his key. The attacker can still use the now seemingly worthless key to produce messages that are related to consensus on previous blocks. This is often called a *long-range attack*.

Forward-secure signature schemes, first proposed by Anderson in 1997 and formalized by Bellare and Miner in 1999, are intended to overcome this key exposure problem. The goal of a forward-secure signature scheme is to maintain the validity of past signatures even if the current signing key has been compromised. This is achieved by dividing the total time during which the public key PK is valid into T time periods, and using a different secret key SK_i in each time period. The public key remains fixed in any of the T time period. A key update algorithm is used to compute each next secret key from the current one. The user generates signatures using a separate signing key for each period: SK_1 for period 1, SK_2 for period 2, and so on. The secret key in period i is derived as a function of the one in the previous period $i-1$. Moreover, at the beginning of period i the user applies a public one-way function h to SK_{i-1} to obtain SK_i, and deletes SK_{i-1} as well. This way, an attacker breaking in during period i will certainly get the key SK_i, but not the previous keys $SK_0, ..., SK_{i-1}$, since they have been deleted. Furthermore, since SK_i was a one-way function of the previous key, the attacker is unable to retrieve the old keys from it.

A signature will include the value of the time period during which it was generated. The verification algorithm takes the fixed public key PK, a message and the candidate signature, and verifies that the signature is valid in the sense that it was generated by the honest user during the time period specified in the signature.

Use of forward-secure signatures in blockchains

Forward-secure signature schemes are used by proof-of-stake-based systems such as Algorand or the Ouroboros protocols that build the foundation of the Cardano blockchain. The main idea is that after each time the key is used during the protocol, such as during consensus, participants in the protocol must evolve their signature keys. If an attacker obtains the participant's key after the transaction, he won't be able to create messages related to this or any previous consensus round because the "old" signature key has been deleted.

5 Verifiable random functions

The concept of verifiable random function was introduced by Silvio Micali, Michael Rabin, and Salil Vadhan in 1999. A verifiable random function (VRF) receives a string as input, generates a pair consisting of a public key and a private key, and outputs a pair (r, π) where:

- r is a pseudo-random number that is indistinguishable from a random string to anyone that does not have access to the private key which is necessary in the process of evaluating the function
- π is a string, also known as the proof, that allows anyone with access to the public key to verify the correctness of r

VRFs play an important part in blockchains since they are used in consensus protocols based on honest majority of stake. Stake is defined as the wealth someone possesses in a blockchain network, represented by coins or tokens of the cryptocurrency associated with the blockchain. In stake-based consensus protocols parties try to find consensus on the next block to add to a blockchain, meaning that each party evaluates a VRF to check if it is eligible to propose a block. Usually eligibility is obtained by checking that r corresponds to a very small number.

The proof π is essential in a way that it prevents a situation where any party could claim to be eligible by choosing a forged r. The fact that r is unpredictable guarantees that eligible parties remain hidden until they announce a block.

Use of VRFs in blockchains

Some functional blockchains that use VRFs are: Algorand, Cardano Ouroboros PoS, Polkadot, Chainlink.

Algorand

The Algorand's source code for the implementation of their VRF is public. A pseudorandom output is generated by the VRF using a secret key and a value, along with a proof that anyone may use to verify the result. The VRF is used to choose leaders to propose a block and committee members to vote on a block. It works similarly to a lottery as follows: the output produced when executed for an account is used to sample from a binomial distribution to simulate a call for every algo in a user's account. So the more algos in an account, the greater chance the account has to be selected.

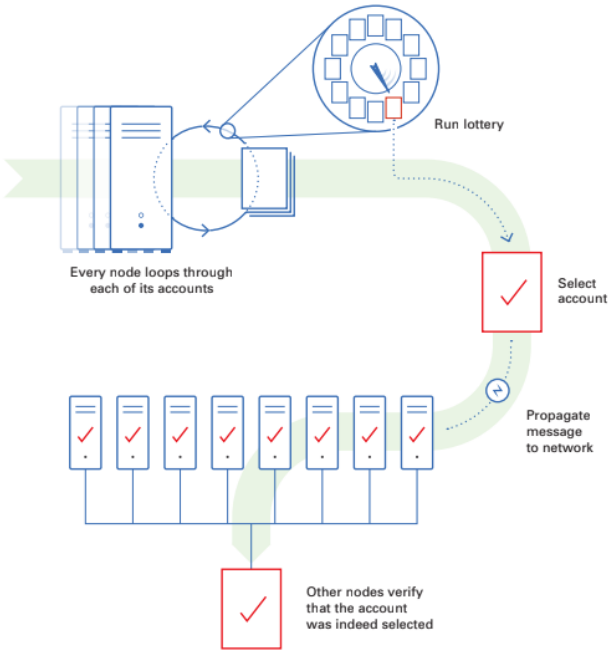


Figure 15: Algorand block proposal

6 Commitment schemes

A commitment scheme allows a sender to encode a message (s)he commits to while keeping it hidden and to reveal the committed message at a certain desired time in the future. Commitments are used to bind a party to a message, in order to prevent them from benefiting in an inappropriate way by switching to other messages.

A commitment scheme has three important properties:

- **hiding** - the message remains hidden from the receiver until the sender decides to reveal it.
- **binding** - the sender can successfully reveal only one message, namely the one (s)he committed to originally.
- **correctness** - the receiver can check the correctness of the revealed message.

Figure 16: Example for use of commitment scheme

There are applications of blockchains where maintaining the privacy of data is as important as verifying in a public manner how that data is being processed. And this is where commitment schemes play a significant part because the private desired information to be uploaded on the blockchain is hidden through their encoding. The privacy of the information is maintained by the hiding property of the commitment scheme. Complications related to modifying the information being processed cannot occur since the binding property ensures that the sent information cannot be changed. The correctness property allows the information to be revealed on-chain because privacy is not an issue anymore, or off-chain because the information remains private to the public and at the same time can be revealed to a certain party.

Pedersen commitment scheme

The Pedersen commitment scheme is based on the discrete logarithm assumption. It uses:

- p and q - are large primes such that q divides $p-1$.
- G_q - is the unique subgroup of $\mathbb{Z}_p^*$ of order q.
- g and h - are elements of G_q, such that nobody knows $\log_g h$. They can be chosen by the receiver or by some external trusted party.

The sender commits to a message $m \in \mathbb{Z}_q$, chooses a random $r \in \mathbb{Z}_q$ and computes the commitment $c = g^m \cdot h^r$. The commitment can be revealed when the sender sends the pair (m,r). The receiver knows the values of g and h and can verify the commitment by recomputing it and checking the result is equal to c.

This scheme is re-randomizable, meaning that another commitment of m is $c' = c \cdot h^s$, where s is random.

This scheme has a homomorphic property that says that given two commitments c and c' of two messages m and m', then $c'' = c \cdot c'$ is a commitment of the message $m'' = m + m'$.

Use of commitment schemes in blockchains

Blockchains like Monero, Algorand, Zerocash use commitment schemes to maintain privacy when transfering coins. Constructing "confidential transactions" using Pedersen commitments was proposed by Adam Back in 2013 in order to add privacy to Bitcoin transactions.

7 Non-interactive proofs

A proof is a logical way in which mathematicians demonstrate that a theorem is true. Classical proofs are non-interactive and since anyone can access them they are also publicly verifiable.

A proof can contain a private key or sensitive information that a prover might not want to reveal when sending the proof. This problem was resolved when Goldwasser, Micali and Rackoff introduced the concept of interactive zero-knowledge proofs that mentain the privacy of data. In such proofs a user (the prover) manages to convince another user (the verifier) that some theorems are true, without revealing any side information or giving an explanation. It is interactive because the prover and the verifier talk back and forth. Even if the prover's input remains private, only the verifier who interacts with the prover is convinced of the theorem's validity, which means that interactive zero-knowledge proofs are not publicly verifiable.

Non-interactive zero-knowledge proof

Since most applications need to have only one prover being able to convince many verifiers about the validity of some theorems, non-interactive zero-knowledge (NIZK) proofs have been explored and discovered by Blum, Feldman and Micali, where once the proof is created, it can be verified by anybody and in this way the public verifiability property is achieved.

NIZK proofs need both the prover and the verifier to have access to the same, short, random string or to the same random oracle model. The last one is used in the Fiat-Shamir transform that was invented in 1986. It changes interactive zero-knowledge proof to non-interactive zero-knowledge proof even with reference to unlimited provers, as long as they only send a polynomial number of queries to the random oracle.

The difference between proofs and arguments:

- in a proof system even an unlimited adversarial prover cannot convince the verifier of a false claim with non-negligible probability.
- in an argument system the security of the verifier is limited to polynomial-time adversarial provers.

SNARGs and SNARKs

A succinct non-interactive argument (SNARG) is a non-interactive argument system where the proof computed by the prover is short in length and fast to verify. A SNARG for a language L is a triple of algorithms (G, P, V) where:

- G (the generator) receives a security parameter λ and samples a reference string α and a corresponding verification state τ . G can be thought to be run during an offline phase by the verifier or by someone the verifier trusts.
- P (the prover) receives the parameters: α (obtained by G), $x \in L$ a statement and w a witness (the information that allows the prover to convince the verifier), and produces a proof π.
- V (the verifier) receives the parameters τ, x, π and verifies the validity of π.

The most common definitions that formalize what "short" and "fast" should actually mean require the length of the proof and the running time of the verifier to be bounded by $p(\lambda + |x|)$, where p is a fixed polynomial, meaning that it does not depend on the language.

When a SNARG must also be an argument of knowledge, then the name SNARK is used. Groth and Lipmaa introduced clever techniques for constructing preprocessing SNARKs by leveraging knowledge-of-exponent assumptions in bilinear groups. At high level, Groth considered a simple reduction from circuit satisfaction problems to an algebraic satisfaction problem of quadratic equations, and then constructed a set of specific cryptographic tools to succinctly check satisfiability of this problem. Gennaro et al. made a first step to better separate the "information-theoretic ingredient" from the "cryptographic ingredient" in preprocessing SNARKs. They formulated a new type of algebraic satisfaction problems, called Quadratic Span Programs (QSPs), which are expressive enough to allow for much simpler, and more efficient, cryptographic checking, essentially under the same assumptions used by Groth.

STARKs

Scalable and transparent arguments of knowledge (STARKs) have more recently been developed to overcome some of the limitations of SNARKs. Micali and Kilian proposed in 1994 to exclude trusted parameters, so a zero-knowledge STARK does not need a common reference string and security is demonstrated in the random oracle model. Another interesting aspect of STARKs is that they are currently immune to any direct attacks, including those utilizing quantum computing. A disadvantage of using STARKs is that they are large in size, being able to reach a few hundred kilobytes or even some megabytes.

Use of zk-SNARKs in blockchains

Zerocash is known as one of the first blockchain protocols to design a zk-SNARK, which can hide vital blockchain transaction data.

8 Privacy-Enhancing Signatures

8.1 Blockchain Privacy Protection Concerns

Blockchain technology's key features are decentralization, transparency, auditability, and persistence. These qualities appear appealing, but they have specific associated side effects that provide several difficulties. Some of these include address tracking and transaction linkability, and identity in a decentralized and permanent ecosystem. Managing keys or wallets is also a major issue with blockchain technology. Key creation and exchange referred to as key managerial tasks, are historically difficult for businesses to handle. IT companies face challenges such as scalability, security, and availability when trying to control and manage encryption keys. The Public Key Scheme and the Ring Signature Scheme offer secure keys by randomly mixing them to provide anonymity for the user. Changing the user's address can also be risky. Even though Blockchain is gaining popularity due to its innovative characteristics, being a distributed system with decentralized control necessitates a close examination of the vital security elements and authentication procedures. Every layer of the Blockchain architecture has to undergo a security examination. Despite several publications discussing three, four, or five-layered network designs for Blockchain, there is unfortunately no common model architecture.

8.2 Term Introduction

8.2.1 Digital Signature

The procedure of digital signature allows the sender to confirm the document's integrity and validity. In simple terms, it's confirmed that a document was transmitted by the expected sender and wasn't altered by a network intruder. The message is encrypted using the sender's private key. And the communication may be decrypted using the sender's matching public key, which was previously supplied to the recipient. The sender prepares a hash of the document that has to be signed to construct a digital signature system. Then transmit it after encrypting it with his private key. Utilizing the sender's public key, the recipient decrypts the sent file. That guarantees it is sent by the expected sender.

8.2.2 Group Signature

A group signature may be thought of as a regular digital signature with the addition that anybody can confirm that a signature is created by an authorized group member while still allowing a chosen trusted party called the group manager to identify who the actual signer is. The majority of group signature methods operate on the following fundamental tenet: A group member (Alice) must create authorization proof to establish that she has a valid membership certificate and an ownership proof to exhibit knowledge of the secret associated with the membership certificate to sign a message. A group signature method, on the other hand, necessitates an initialization procedure for identifying a group, which in certain cases may not be feasible or desirable.

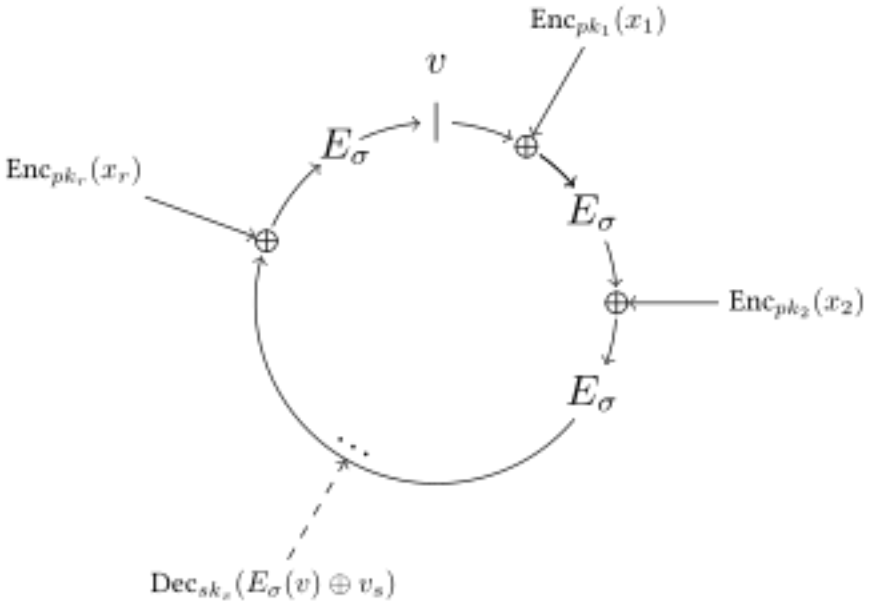
8.3 Mechanisms for Privacy Enhancement

8.3.1 Privacy Enhancement Techniques

The employment of **P**rivacy **E**nhancement **T**echniques (**PET**s) can help to protect a user's online transactions, data, and identity. PETs are used to protect the security and privacy of data and include both hard [4] and soft [5] privacy solutions. Using aggregate signature and group signature algorithms from a collection of mixing peers, a mixing transaction privacy improvement strategy may be created. Additionally, there are methods for enhancing network secrecy, such as onion and garlic routing.

8.3.2 Ring signatures

A digital signature known as a "ring signature" allows a signer to affix their mark to a message so that a group of users may validate it. That is a selection of users made by the signer at random during the signature creation process. Regardless of prior knowledge, approval, or support, all members must be aware of each other's public keys to join the ring. Ring signatures provide for some anonymity because they only reveal to the verifier that the signer is one of the members of the ring and not which specific user signed. Ron Rivest, Adi Shamir, and Yael Tauman introduced ring signatures in 2001.



8.3.3 Threshold Ring Signature Schemes

Ring signatures are exciting to use with blockchains since they have excellent security criteria. Examples include threshold ring signature techniques, which unite the threshold signing and anonymity guarantees of ring signatures. Threshold ring signatures are appropriate for decentralized situations where the number of active participants is unknown and parties can join and leave the system on demand. They let parties support a statement

[4] Hard privacy technologies enable online users to protect their privacy through various services and applications. With the ability to hide or interact secretly, the objective is to minimize the quantity of data gathered and decrease dependency on third parties.

[5] Soft privacy technology strives to safeguard data and allows services to process it while maintaining complete control over its usage.

while maintaining their anonymity. Threshold ring signatures, for instance, can be used in a trusted blockchain to guarantee that specific transactions are only added to the blockchain if at least n trustees accept the operation, without disclosing the trustees' identities. When a quorum is required to approve a statement but the participants desire to maintain their anonymity, threshold ring signatures can be used.

8.3.4 Accountable Ring Signatures

The following is guaranteed by an accountable ring signature: anyone can confirm that the signature was created by a user who is a member of a set of potential signers that can be selected at any time, but the actual signer can still be identified by a designated trusted entity, a system participant independent of any potential set of users.

For usage in the real world, accountable ring signatures would be preferable to ring signatures because:

1. A communication with an accountable ring signature would be more trustworthy than one with a more common unaccountable ring signature.
2. Traditional ring signature methods' information-theoretic anonymity would prevent their use in the real world; a similar situation has occurred with anonymous e-cash techniques.

If an **A**ccountable **R**ing **S**ignature (**RSA**) system satisfies the qualities listed below, it is considered secure:

1. Correctness. An honest user's ring signature is always recognized as being legitimate concerning the given ring.
2. Unforgeability. Any user must be unable to create a legitimate ring signature for a ring to which he does not belong, except for a very small chance.
3. Anonymity. No verifier should be able to determine the identity of the real signer with a probability of greater than $1/z+\varepsilon$ where z is the ring size and ε is a negligible function.
4. Unlinkability. Even if the user's identity is still unknown to the attacker, no verifier can connect two ring signatures created by the same user. 2
5. Revocability. There is an authority that may revoke anonymity and reveal who signed a certain ring signature.
6. No-misattribution. The anonymity revocation authority will not be able to persuade a trustworthy verifier that a particular ring signature was issued by a user who is not the signer.
7. Coalition resistance. A subset of users who are cooperating (or even all users) cannot produce a ring signature that the authority for revoking anonymity cannot link to at least one of the colluding users.

The compiler's fundamental idea is to incorporate tamper-resistant smart cards that are handed to users who wish to create accountable ring signatures as well as an anonymity revocation authority (ARA) that may identify the true signer of a particular accountable ring signature when necessary.

Regarding utilizing smart cards, explained how to create a group signature system based on tamper-resistant smart cards. The fundamental notion is as follows: A group manager selects two sets of public and private keys,

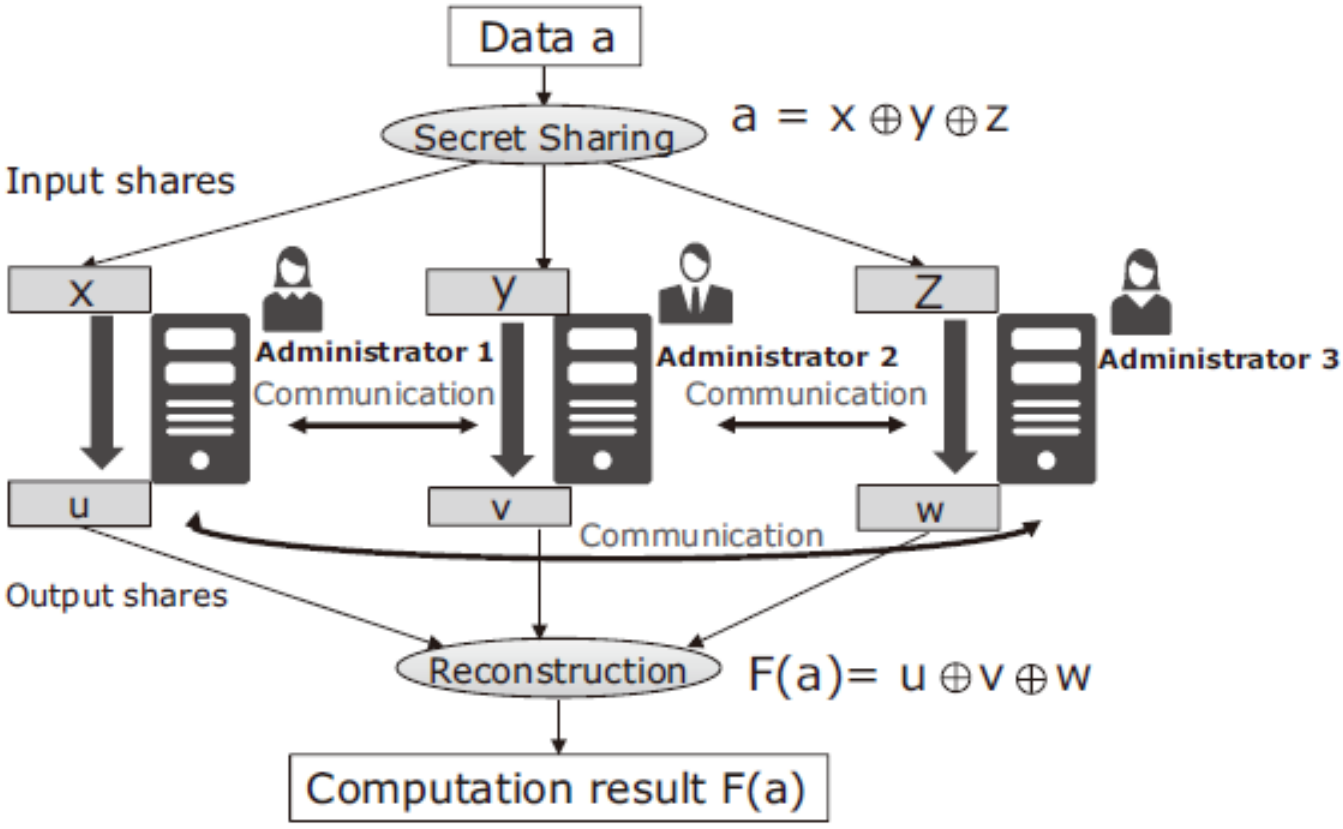
(pk_1, sk_1) for public key cryptosystems, and (pk_2, sk_2) for the digital signature systems, where both pk_1 and pk_2 are known to the public. Each group member is identified by their identity U and owns a tamper-proof smart card that has a pk_1 and sk_2. A group signature is a type of digital signature that is created by applying sk_2 on the concatenation of a message m and the ciphertext c produced by U's encryption with public key pk_1. As a result, the group manager can conceal who signed a certain group signature.

9 Secure Multi-Party Computation

Yao first proposed Secure Multi-party Computation (MPC) in 1982, offering decentralization as a solution to "The Millionaires' Problem". In their 1987 work, Goldreich et al. expanded the two-party computation to include more parties. A collaborative computing problem that guarantees privacy amongst a group of untrusted individuals is resolved using MPC. In this case, secret input is held by two or more parties who want to jointly compute a function to obtain their own output. The participant in this procedure just receives the anticipated result; no further information is provided. In further detail, $\{P = P_1, ..., P_n\}$ participants each hold a set of private data, $x_1, ..., x_n$, respectively. Participants wish to calculate the value of the public function $f(x_1, ..., x_n) = (r_1, ..., r_n)$ on that private data while keeping their own inputs private. Each participant P_i does not get any information other than (x_i, r_i) and the meaningless intermediate values obtained therefrom at the conclusion of the calculation.

Secure MPC must adhere to the following two requirements:

1. Participant P_i is prohibited from accessing any further input, $x_j \quad i \neq j$
2. Sincere contributors get the same results as $r_1 = r_2 = ... = r_n$.



9.1 Restricted Interaction MPC

It is frequently challenging and, in many instances, outright impossible for the parties to communicate fully. For instance, time constraints could make it difficult for wireless devices to send or receive messages, and physical obstacles might make it impossible for them to communicate with one another directly. A leaner style of communication may also be motivated by efficiency concerns. Think about calculating the majority vote using the data from n parties. If security is not required, this job can be computed using just (n-1) messages, but in typical MPC protocols with complete interaction, $\Omega(n^2)$ point-to-point communications are required to compute the same task safely.

The research of MPC procedures with limited party contact has been driven by such reasons. The first researchers on this issue were Halevi, Lindell, and Pinkas. They looked at an interaction pattern in which each of the n parties sends a single message to a centralized server, which then computes the function's result. Goldwasser et al. and Beimel et al. recently investigated a novel interaction pattern where each party individually sends a single message to the server. In both situations, the security assurance is inherently less strong than the typical MPC simulation-based security.

The SMPC protocol has been adopted in practice somewhat slowly, despite several recommended systems. Several alternatives, including Shamir Secret Sharing, Honest-majority MPC with secret sharing (BGW protocol), Threshold cryptography, and Dishonest-majority MPC, are being investigated to address the issues with secure multi-party communication. In the Honest Majority Assumption method, more than half a percent of the par-

ticipants may be honest, whereas the Dishonest Majority Hypothesis implies that more than half percent of the participants may be dishonest. Yehuda Lindell has continued to work in this field since 2004, and his website includes the articles and projects he has undertaken.

9.2 Blockchains' use of multi-party computation

9.3 Keeping identity wallets safe

The identity of the person or business submitting the transaction is represented by the private keys that end users use to sign blockchain transactions. The theft or loss of the private key might have a significant effect. And that's where MPC can help, by splitting apart the keys and dynamically recreating it using input from all parties. Therefore, even if one party is hacked, it is impossible to complete the blockchain transaction using just that shard.

9.3.1 Transactions

In a number of situations involving transactions, various parties must provide their authorization or approval before the transaction is carried out. To guarantee that the output created from each participant's private input will be necessary to complete the transaction on the blockchain, MPC can be utilized in conjunction with this approval method. By using the "M of N" strategy, where at least M people out of a total of N must submit their own private input, this may be strengthened even more. Using multi-sig (multiple signature addresses), which is supported by a few blockchain protocols, is another option for this strategy. MPC, on the other hand, is a completely software-based solution that is platform-independent.

9.3.2 Transaction privacy and confidentiality

To obtain consensus and disseminate copies of the ledger, blockchain solutions frequently rely on broadcasting transactions to all relevant nodes. In some circumstances requiring confidential information or calculations, using this paradigm could be challenging. The transaction receipt is stored as proof on the blockchain and is available for confirmation at any time. Such transactions can be handled via MPC and offloaded from the blockchain.

10 Conclusions

In order for the modern digital world to function without physical interaction, this article established the technological needs for data privacy and protection in general for online transactions. The significance of user privacy is emphasized in both the current and prospective data protection rules of many nations, which urge that users should have the fundamental right to erase and forget if they so choose. In the digital age, it is a fundamental necessity that systems be protected from assaults and that countermeasures be put in place to lessen their susceptibility. The applicability of blockchain technology to satisfy security and privacy criteria was investigated in this context, and the issues that need more research and study to make blockchain technology future-proof have been thoroughly analyzed. The application of privacy enhancing techniques has been thoroughly described. It has been

underlined that strong crypto-privacy algorithms are required in order to improve Blockchain technology's efficiency and enable the construction of quantum-resistant ledgers. These actions are anticipated to increase the use of Blockchain in the digital world while completely complying with privacy laws. This paper's suggested design takes into account the numerous drawbacks of blockchain applications. To make blockchain technology resilient, mitigation strategies must be used if and when new restrictions are discovered.

References

[1] B. Tackmann, I. Visconti, *Cryptographic tools for blockchains*, 2021

[2] Timo Bürk, *Blockchain consensus protocols based on stake*, Master Thesis, Faculty of Science at the University of Bern in Switzerland, 2022 - page 6
https://crypto.unibe.ch/archive/theses/2021.msc.timo.buerk.pdf

[3] Algorand
https://developer.algorand.org/docs/get-details/algorand_consensus/?from_query=verifiab#verifiable-random-function

[4] Torben Pryds Pedersen, *Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing*, Computer Science Department, Aarhus University, Denmark
https://link.springer.com/content/pdf/10.1007/3-540-46766-1_9.pdf?pdf=inline%20link

[5] Image for commitment scheme example
https://karl.tech/learning-solidity-part-2-voting/

[6] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, Omer Paneth, *Succinct Non-Interactive Arguments via Linear Interactive Proofs*
https://eprint.iacr.org/2012/718.pdf

[7] Yvo Desmedt, Yair Franked, *Threshold cryptosystems*
https://link.springer.com/content/pdf/10.1007/0-387-34805-0_28.pdf?pdf=inline%20link

[8] TSS explained by Binance
https://academy.binance.com/en/articles/threshold-signatures-explained

[9] TSS explained by BitGo
https://blog.bitgo.com/bitgo-tss-a-technical-deep-dive-754d703271cb

[10] Mihir Bellare, Sara K. Miner, *A Forward-Secure Digital Signature Scheme*
https://link.springer.com/content/pdf/10.1007/3-540-48405-1_28.pdf?pdf=inline%20link

[11] Dan Boneh, Manu Drijvers, Gregory Neven, *Compact Multi-Signatures for Smaller Blockchains*
https://eprint.iacr.org/2018/483.pdf

[12] Usage scenarios for multi-signatures
https://academy.binance.com/en/articles/what-is-a-multisig-wallet

[13] Image for examples of network architectures
https://www.researchgate.net/figure/A-comparison-of-the-topography-of-centralized-decentralized-and-distributed-networks_fig2_333659272

[14] Image for Merkle Tree architecture
https://www.researchgate.net/figure/The-architecture-of-Merkle-tree-in-the-blockchain_fig4_334291891

[15] Yi Xie, Jiayao Zhang, Honglin Wang, Pengran Liu, Songxiang Liu, Tongtong Huo, Yu-Yu Duan, Zhe Dong, Lin Lu, Zhewei Ye, *Applications of Blockchain in the Medical Field: Narrative Review*
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8555946

[16] Jay Gupta and Swaprava Nath, *SkillCheck: An Incentive-based Certification System using Blockchains*, Indian Institute of Technology Kanpur
https://www.cse.iitb.ac.in/~swaprava/papers/skillcheck.pdf

[17] Kevin Doubleday, Brian Platz, *Blockchain Immutability — Why Does it Matter?*, 2018
https://medium.com/fluree/immutability-and-the-enterprise-an-immense-value-proposition-98cd3bf900b1

[18] Huabing Zhao, *Hash functions and Data Structures*
https://zhaohuabing.medium.com/hash-pointers-and-data-structures-f85d5fe91659

[19] Marie Jeanne Tuyisenge, *Blockchain technology security concerns: literature review*, Uppsala University, 2021

[20] Kerala Blockchain Academy, *Merkle Tree: A Beginners Guide*
https://medium.com/blockchain-stories/merkle-tree-a-beginners-guide-5c53a7defeb9

[21] Alin Tomescu, *What is a Merkle Tree?*
https://decentralizedthoughts.github.io/2020-12-22-what-is-a-merkle-tree/

[22] Jeremy Then, *Merkle Tree, a simple explanation and implementation*
https://medium.com/coinmonks/merkle-tree-a-simple-explanation-and-implementation-48903442bc08

[23] Zerocash project
http://zerocash-project.org/

[24] Haitz Sáez de Ocáriz Borde, *An Overview of Trees in Blockchain Technology: Merkle Trees and Merkle Patricia Tries*, University of Oxford
https://www.researchgate.net/publication/358740207_An_Overview_of_Trees_in_Blockchain_Technology_Merkle_Trees_and_Merkle_Patricia_Tries

[25] Cardano Docs, *Understanding the Extended UTXO model*
https://docs.cardano.org/learn/eutxo-explainer

[26] John Bethencourt, *Intro to Bilinear Maps*, Computer Sciences Department, Carnegie Mellon University https://people.csail.mit.edu/alinush/6.857-spring-2015/papers/bilinear-maps.pdf

[27] S. Bansod, L. Ragha, *Challenges in making blockchain privacy compliant for the digital world: some measures*, Sādhanā 47, 168 (2022)
https://doi.org/10.1007/s12046-022-01931-1

[28] Abida Haque, Alessandra Scafuro, *Threshold Ring Signatures: New Definitions and Post-Quantum Security*, North Carolina State University
https://eprint.iacr.org/2020/135.pdf

[29] Jiapeng Zhou, Yuxiang Feng*, Zhenyu Wang and Danyi Guo, *Using Secure Multi-Party Computation to Protect Privacy on a Permissioned Blockchain*, School of Software Engineering, South China University of Technology, Guangzhou 510006, China, 2021, 21(4), 1540
https://doi.org/10.3390/s21041540

[30] Ronald Cramer, Ivan Bjerre Damgard, Jesper Buus Nielsen *Multiparty Computation and Secret Sharing*, CWI & Leiden University, Aahus University, 2015
https://books.google.ro/books?id=HpsZCgAAQBAJ&lpg=PR9&dq=Secure%20Multi%20Party%20Computation&lr&hl=ro&pg=PR4#v=onepage&q&f=false

[31] Shouhuai Xu, Moti Yung, *Accountable Ring Signatures: a Smart Card Approach*
https://link.springer.com/content/pdf/10.1007/1-4020-8147-2_18.pdf?pdf=inline%20link

[32] Zhou J., Feng Y., Wang Z., Guo D. *Using Secure Multi-Party Computation to Protect Privacy on a Permissioned Blockchain*. Sensors 2021, 21, 1540. https://doi.org/10.3390/s21041540