Blockchain and cryptocurrencies

Michele Zanotti

Contents

1. Preliminary concepts at the basis of Blockchain [2],[3]

   1.1. Introduction to the cryptography concepts used in Blockchain [3]

   - Cryptography services (confidentiality, authentication, integrity, non-repudiation)
   - Public and private key cryptography
   - Elliptic curve cryptography
   - Hash functions
   - Elliptic curve digital signature algorithm (ECDSA)

   1.2. Distributed systems and decentralization

   1.3. Consensus & Byzantine generals problem

2. Introduction to Blockchain [2],[3]

   2.1. What is a Blockchain

   2.2. Blockchain features

   2.3. Types of Blockchain (public, consortium, private)

   2.4. Blockchain history (why it was invented)

   2.5. Overview of today Blockchain applications

3. Bitcoin [15],[1]

   3.1. Bitcoin protocol specification

   - Overview of Bitcoin data types (transaction, scripts, adresses, blocks)

- Transactions
- Bitcoin network architecture
- Bitcoin blockchain (blocks structure, Merkle trees, mining, proof of work)

3.2. Bitcoin wallets

4. Bitcoin privacy

   - Considerations on user anonymity in Bitcoin
   - Possible attacks
   - How to enhance privacy in Bitcoin (explanation of mixing services + reference [13],[19])

5. Bitcoin blockchain scalability

   - Considerations on the scalability of the Bitcoin blockchain and possibile solutions [15],[11]

6. Alternatives to Bitcoin

   - Bitcoin limitations
   - Alternatives to proof of work [4]
   - Namecoin
   - Litecoin
   - ZCash

1 Introductory concepts

1.1 Hash functions

A hash function is a fuction that maps an arbitrary long input string to a fixed length output string. Let h refer to an hash function of length n:

$$h\colon \{0,1\}^* \rightarrow \{0,1\}^n$$

m is usually called "the message", while d is usually called "the digest" and it can be seen as a compact representation of m. The length of d is the

Hash functions are usually used to provide data integrity and they're also used to construct other cryptographic primitives such as MACs and digital signatures.

1.1.1 Desired properties

An hash function should ideally meet these properties:

- **Computational efficiency**: given m, it must be easy to compute $d = h(m)$
- **Preimage resistance** (also called **one-way property**): given $d = h(m)$, it must be computationally infeasible computing m (m is the preimage)
- **Weak collision resistance** (also called **2nd preimage resistance**): given m_1 and $d_1 = h(m_1)$, it must be computationally infeasible finding a $m_2 \neq m_1$ so that $h(m_2) = d_1$
- **Strong collision resistance**: it must be computationally infeasible finding pairs of distinct and colliding messages. Two messages $m_1 \neq m_2$ collide when $h(m_1) = h(m_2)$.
- **Avalanche effect**: changing a single bit of m should cause every bit of $d = h(m)$ to change with probability $P = 0.5$

1.1.2 Examples of hash functions

- **MD5**: published in 1991, it's a 128-bit hash function that was used for file integrity checks. Today it's considered insecure and it shouldn't be used anymore.

- **Secure Hash algorithm 1 (SHA-1)**: 160-bit hash function that was used in SSL and TLS implementations. Today is considered insecure and it's deprecated.
- **SHA-2**: family of SHA functions which includes SHA-256, SHA-384 and SHA-512. SHA-256 is currently used in several parts of the Bitcoin network.
- **SHA-3**: latest family of SHA functions, it is a NIST-standardized version of Keccak, which uses a new approach called "sponge construction" instead of the Merkle-Damgard transformation previously used. This family includes SHA3-256, SHA3-384 and SHA3-512.

1.1.3 Design of SHA-256

1.1.4 Message Authentication Codes (MACs)

A MAC is an hash function which uses a key and which can therefore be used to provide both integrity and authentication (proof of origin). Authentication is based on a key pre-shared between the sender and the receiver. The receiver can verify both integrity and authentication of a message by computing the MAC function of the message and comparing it with the one received from the sender: if they are the same then integrity and authentication are confirmed (note that it is assumed that only the sender and the receiver know the key).

MAC functions can be constructed using block ciphers or hash functions:

- in the first approach, block ciphers are used in the Cipher block chaining mode (CBC mode): the MAC of a message will be the output of the last round of the CBC operation. The length of MAC, in this case, is the same as the block length of the block cipher used to generate it.
- In the second approach, they key is hashed with the message using a certain construction scheme. The most simple ones are *suffix-only* and *prefix-only*, which however are weak and vulnerable:
  - suffix-only: $d = MAC_k(m) = h(m|k)$, where h is an hash function
  - prefix-only: $d = MAC_k(m) = h(k|m)$, where h is an hash function

1.2 Digital signature

Digital signatures are used to associate a message with the entity from which the message has been originated. They provide the same service as MACs (authentication and non-repudiation) plus the non-repudiation.

Digital signature is based on public key cryptography: Alice can sign a message by encrypting it using its private key. Usually, however, for efficiency and security reasons, Alice doesn't encrypt the message but its digest (hash of the message). Figure 1 shows how a generical digital signature function works.

An example of digital signature algorithms are RSA and ECDSA.

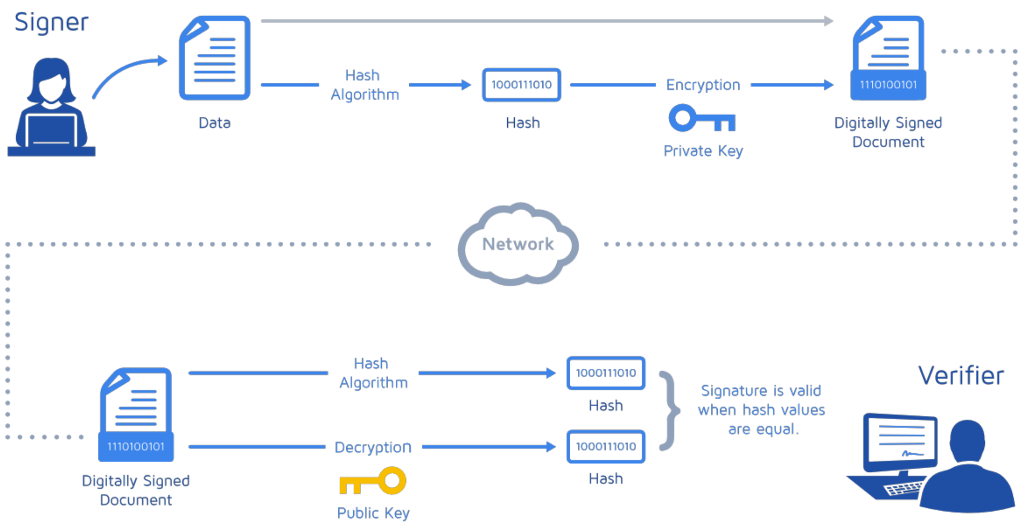


Figure 1: digital signature signing and verification scheme

1.3 Elliptic Curve Digital Signature Algorithm (ECDSA)

ECDSA is a variant of the Digital Signature Algorithm (DSA) which uses elliptic curve cryptography.

1.3.1 Key pair generation

1. Define an elliptic curve E with modulus P, coefficients a and b and a generator point A that forms a cyclic group of order p, with p prime

2. Choose a random integer d so that $0 < d < q$

3. Compute the public key B so that $B = dA$

The public key is the sextuple $K_{pb} = (p, a, b, q, A, B)$, while the private key is the value of d randomly chosen in Step 2: $K_{pr} = d$

1.3.2 Signing a message

1. Choose an ephemeral key K_e, where $0 < K_e < q$. It should be ensured that K_e is truly random and no two signatures have the same key because otherwise the private key can be calculated
2. Compute $R = K_e A$
3. Initialize a variable r with the x coordinate value of the point R
4. The signature on the message m can be calculated as follow:

   $$S = (h(m) + dr)K_e^{-1} \bmod q$$

   where $h(m)$ is the hash of the message m. The signature is the pair (S, r).

1.3.3 Signature verification

A signature can be verified as follow:

1. Compute $w = S^{-1} \bmod q$
2. Compute $u_1 = wh(m) \bmod q$
3. Compute $u_2 = wr \bmod q$
4. Calculate the point $P = u_1 A + u_2 B$
5. The signature (S, r) is accepted as a valid signature only if:

   $$X_P = r \bmod q$$

   where X_P is the x-coordinate of the point P calculated in Step 4

1.4 Blind signature

Blind signatures were introduced by David Chaum in 1982 [10] and refer to a cryptographic primitive that allows an entity to digitally sign a message without knowing or being able to read the message that it signs. The following analogy introduced by Chaum himself clearly explains what blind signatures are:

"*Assume an envelope with both a piece of paper (e.g. a contract) and carbon paper inside it. The envelope is sealed and sent to the signer. The signer cannot see what is inside the envelope without breaking the seal. The signer signs the envelope, and thanks to the carbon paper, the contract inside the envelope gets signed too. The signer returns the envelope to the sender, who opens it and extracts the carbon-signed contract.*"

Blind signature can be implemented using different schemes. In this section it will be briefly discussed the RSA scheme proposed by Chaum. Another scheme based on the Diffie-Hellman problem will be discussed in section 5.2.

1.4.1 RSA signature scheme

In RSA the public parameters are $n = pq$ and a chosen e relatively prime to $\varphi(n)$, while the private parameters are the primes p, q, $\varphi(n)$ and $d = e^{-1} \bmod \varphi(n)$. The signer private key is d and its public key is e.

The signer can sign a message m by computing the signature s:

$$s = m^d \bmod n$$

Anyone can verify the signature using the signer public key and verify if the message is equal to the result of the computation below:

$$s^e = m^{ed} \bmod n = m \bmod n$$

1.4.2 RSA blind signature scheme

1. Generation of a blinding factor $b = r^e \bmod n$, with r random number
2. The message m is blinded with the blinding factor b previously calculated: $m_* = b \cdot m \bmod n$
3. The blinded message m_* is sent to the signer. The signer signs it by computing s_*:

$$s_* = m_*^d \bmod n = b^d \cdot m^d \bmod n = r^{e \cdot d} \cdot m^d \bmod n = r \cdot m^d \bmod n$$

4. The user can divide s_* by r for retrieving $s = m^d \bmod n$, namely the RSA signature of the message m

1.5 Distributed systems

1.5.1 What is a distributed system

Blockchain at its core is basically a distributed system, therefore it is essential to understand distributed systems before understanding Blockchain.

A distributed system is a network that consists of autonomous nodes, connected using a distribution middleware, which acts in a coordinated way (passing messages to each other) in order to achieve a common outcome and that can be seen by the user as a single logical platform.

A node is basically a computer that can be seen as an individual player inside the distributed system and it can be honest, faulty or malicious. Nodes that have an arbitrary behavior (which can be malicious) are called *Byzantine nodes*.

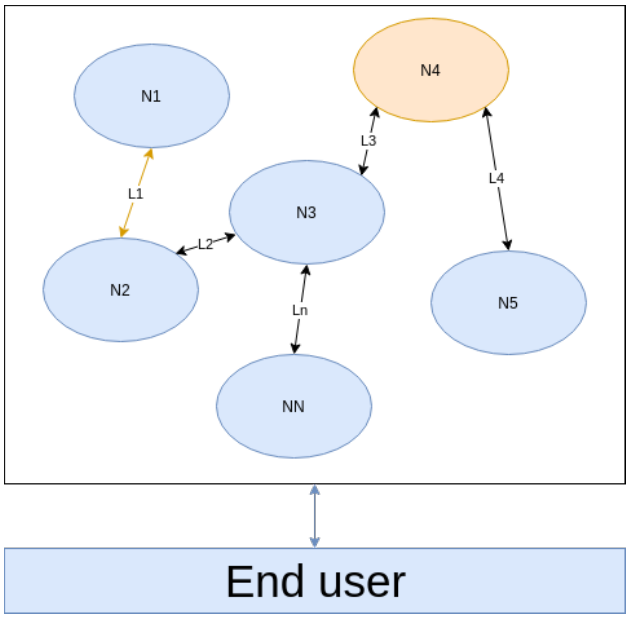


Figure 2: design of a distributed system. N4 is a Byzantine node while L1 is a broken/slow network link

The main challenge in a distributed system is the fault tolerance: even if some of the nodes fault or links break, the system should tolerate this and should continue to work correctly. There are essentially two types of fault: a simple node crash or the exhibition of malicious or inconsistent behavior arbitrarily. The second case is the most difficult to deal with and it's called *Byzantine fault*. In order to achieve fault tolerance, replication is usually used.

Desired properties of a distributed system are the following:

- **Consistency**: all the nodes have the same lates available copy of the data. It is usually achieved through consensus algorithms which ensure that all nodes have the same copy of the data
- **Availability**: the system is always working and responding to the input requests without any failures
- **Partition tolerance**: if a group of nodes fails the distributed system still continues to operate correctly

There is however a theorem, the *CAP theorem*, which states (and proves) that a distributed system cannot have all these three properties at the same time. In particular, the theorem states that in the presence of a network partition (due for example to a link failure) one has to choose between consistency and availability.

1.5.2 Consensus

Consensus is the process of agreement between untrusted nodes on a data value. When the involved nodes are only two it's really easy to achieve consensus, while in a distributed system with more than two nodes it is really hard (in this case the process of achieving consensus is called *distributed consensus*). The data value agreed is the majority value, therefore the value proposed by 51% of the nodes.

A consensus mechanism must meet these requirements:

- **Agreement**: all the correct (non-faulty/malicious) nodes must agree on the same value
- **Termination**: the execution of the consensus process must come to an end and the nodes have to reach a decision
- **Validity**: the agreed value must have been proposed by at least one honest node

- **Fault tolerance**: the consensus algorithm must be able to run even in the presence of one or more Byzantine (faulty or malicious) nodes
- **Integrity**: the nodes make decisions only once in a single consensus cycle (in a single cycle a node cannot make the decision more than once).

1.5.3 The Byzantine Generals Problem (BGP)

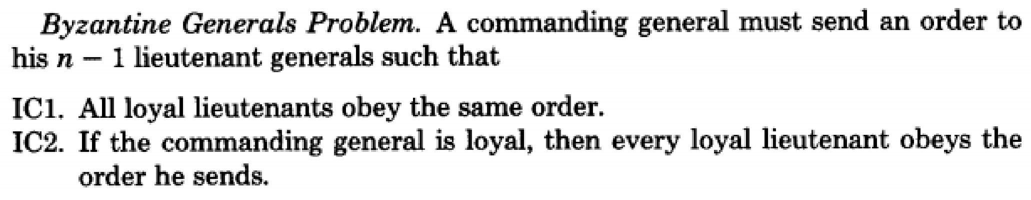
The Byzantine Generals Problem (BGP) is a problem described by Leslie Lamport [17] in which a group of generals are surrounding a city and they have to formulate a plan for attacking it (simplifying, they have to decide whether to attack or retreat from the city). Their only communication way is the messenger and they have to agree on a common decision. The issue is that some of the generals may be traitors trying to prevent the loyal generals from reaching an agreement by communicating a misleading message. The generals need an algorithm to guarantee that all the loyal generals agree on the same plan (attack or retreat) regardless of what traitors generals do. Loyal generals will always do what the algorithm says they should, while the traitors may do anything they wish.

As an analogy with distributed systems:

- generals can be considered as nodes
- traitors can be considered Byzantine nodes
- the messenger can be seen as the channels of communication between the generals.

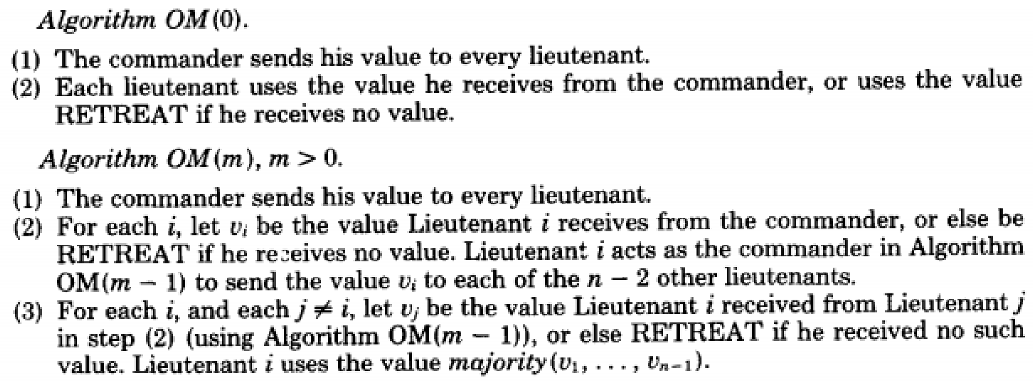
The problem can be see in term of generals-lieutenants: a General makes the decision to attack or retreat, and must communicate the decision to his lieutenants. Both the lieutenants and the general can be traitors: they cannot be relied upon to properly communicate orders (traitor generals) and they may actively alter messages in an attempt to subvert the process (traitor lieutenants).

To solve this problem, Lamport proposed an algorithm for reaching consensus that assumes that there are m traitors and $3m$ actors. This implies that the algorithm can reach consensus only if $2/3$ of the actors are honest: if the traitors are more than $1/3$, consensus cannot be reached. The goal is to make the majority of the lieutenants choose the same decision (not a specific one). The original algorithm proposed by Lamport is shown in figure 4.


Byzantine Generals Problem. A commanding general must send an order to his $n - 1$ lieutenant generals such that

IC1. All loyal lieutenants obey the same order.
IC2. If the commanding general is loyal, then every loyal lieutenant obeys the order he sends.

Figure 3: page 3 of the original Lamport's paper [17]


Algorithm OM(0).

(1) The commander sends his value to every lieutenant.
(2) Each lieutenant uses the value he receives from the commander, or uses the value RETREAT if he receives no value.

Algorithm OM(m), $m > 0$.

(1) The commander sends his value to every lieutenant.
(2) For each i, let v_i be the value Lieutenant i receives from the commander, or else be RETREAT if he receives no value. Lieutenant i acts as the commander in Algorithm OM($m - 1$) to send the value v_i to each of the $n - 2$ other lieutenants.
(3) For each i, and each $j \neq i$, let v_j be the value Lieutenant i received from Lieutenant j in step (2) (using Algorithm OM($m - 1$)), or else RETREAT if he received no such value. Lieutenant i uses the value $majority(v_1, \ldots, v_{n-1})$.

Figure 4: Lamport's algorithm for reaching consensus

1.5.4 Byzantine Fault Tolerance (BFT)

A distributed system is said to be Byzantine Fault Tolerant when it tolerates a the class of failures that belong to the Byzantine Generals' Problem [16]. In other words, a Byzantine Failure is a fault that presents different symptoms to different observers and for this reason BFT is really difficult to achieve.

For example, a Byzantine Fault could be a node acting as a "traitors" and generating arbitrary data during the process of reaching consensus.

2 Introduction to Blockchain

2.1 What is Blockchain

From a technical point of view, Blockchain is a distributed ledger that is cryptographically secure, append-only, immutable (extremely hard to change), and updateable only via consensus among nodes.

From a business point of view, a blockchain can be defined as a platform whereby peers can exchange values without the need for a central trusted party by using transactions which are stored inside the blockchain in a verifiable and permanent way.

2.2 Blockchain features

Decentralization

This is the core feature of Blockchain. Thanks to decentralization there's no need of a central trusted entity which stores the data and validates the transaction, since the same copy of the Blockchain is stored by every node and the validation of transaction is achieved through consensus.

Distributed consensus

Blockchain have a high Byzantine Fault Tolerance[1] and allows to achieve distributed consensus, therefore allows to have a single version of a data value agreed by all parties without requiring a central authority.

High availability

Blockchain is based on a peer-to-peer network of thousands of nodes and data is replicated on each node, therefore the whole system is highly available since even if one or more nodes fail the whole network can continue to work correctly.

[1]without BFT, a peer would able to transmit and post false transactions

Immutability

All the data stored in a blockchain is immutable: once a block has been added to the blockchain, it is considered pratically impossible to change it (changing it is computationally infeasible since it would require an unaffordable amount of computing resources).

Transparency

Blockchain is shared between the nodes and everyone can see what is in the blockchain, thus allowing the system to be transparent and trusted.

Security

Blockchain ensures the integrity and the availability of the data. Since private keys and digital signatures are used, it also provide authentication and non-repudiation. It doesn't provide confidentiality, due to it's transparency feature (privacy is however required in certain scenarios, thus research in this area is being carried out).

Blockchain security is due especially to its distributed nature, since for an attacker would be a lot easier to tamper with data if it was stored on a single central entity.

Uniqueness

In Blockchain every transaction is unique and has not been spent already. This is especially usefull in cryptocurrencies applications of Blockchain, where avoidance of double spending is a key requirement.

2.3 Blockchain structure

As shown in figure 5, a blockchain consists of linked list of ordered fixed-length blocks, each of which includes a set of transactions. In this section, the generic elements of a blockchain will be presented.

2.3 Blockchain structure

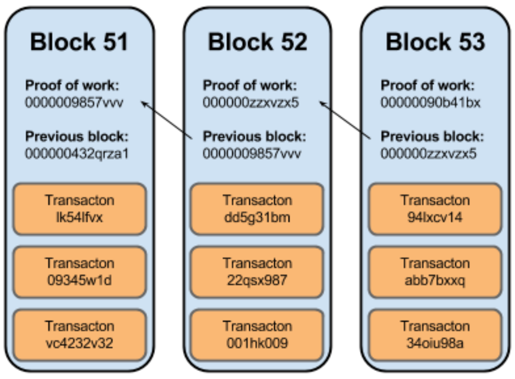


Figure 5: basic blockchain schema

Blocks

A block groups transactions in order to organize them logically and its size depends on the blockchain implementation. Generally, a block is composed of:

- a set of transactions
- a hash which identifies the block
- a pointer to the previous block hash (unless it's the genesis block)
- a nonce
- a timestamp

The *genesis block* it's simply the first block in the blockchain and therefore it can't contain any reference to the previous block.

Addresses

Addresses are unique identifiers which identify the parties involved in a transaction. An address is usually a public key or it's derived from a public key.

Transactions

A transaction is a tranfer of value from an address to another.

Peer-to-peer network

Transaction scripts

Transaction scripts are predefined sets of commands for nodes to transfer values from one address to another and perform various other functions.

Programming language and Virtual machine

A Turing-complete programming language is an extension of transaction scripts and it allows the peers to define the operations that has to be performed on a transaction, without the limitations of a non-Turing-complete transaction script. Programs encapsulate the business logic and can for example transfer a value from one address to another only if some conditions are met.

A virtual machine allows Turing-complete code to be run on a Blockchain as smart contract (e.g. Ethereum virtual machine).

Not every Blockchain supports Turing-complete programming languages and virtual machines (e.g. Bitcoin is not Turing-complete[2]).

Nodes

A node is an active entity which stores a copy of the blockchain and can perform and/or valide transactions (following a consensus protocol, e.g. the Proof of Work).

2.4 Consensus in Blockchain

Consensus in Blockchain is required to establish wheter the ledger itself or a piece of information submitted to it are valid or not. In analogy with the Byzantine Generals Problem, the "generals/lieutenants" are the nodes partecipating in the blockchain, the messangers are the network used by the nodes for communicating and the "traitors" are the nodes which try to tamper with the data by submitting for example false data or by modifying the existing blocks.

In today Blockchain implementations are used four main consensus mechanisms: the Pratical Byzantine Fault Tolerance (PBFT), the Proof of Work (PoW), the Proof of Stake (PoS) and the Delegated Proof of Stake (DPoS).

[2]It however supports smart contracts

2.4 Consensus in Blockchain

2.4.1 Practical Byzantine Fault Tolerance Algorithm (PBFT)

The PBFT is an algorithm proposed by M. Castro and B. Liskov as an optimized solution to the Byzantine Generals Problem (more in general, it is an efficiet replication algorithm that is able to tolerate Byzantine faults [8]).

Simplifying, the algorithm works as follows [12], [8]: each "general" maintains an internal state and when he receives a message, he uses the message in conjunction with his internal state to run a computation, which tells to the general what to think about the message in question. After reaching his individual decision about the message, the general shares that decision with all the other "generals" in the system. A consensus decision is determined based on the total decisions submitted by all generals.

The advantage of this method is that is very efficient and allows to establish consensus with less effort than other methods. The main disadvantage is that it precludes the anonimity of users on the system.

Two example of Blockchains which use PBFT are Hyperledger and Ripple.

2.4.2 Proof of Work (PoW)

Contrary to the PBFT, Proof of Work doesn't require all nodes to submit their individual conclusions in order for a consensus to be reached. Instead, this mechanism relies on proof that enough computational resources have been spent before proposing a value for acceptance by the network: only a single node (the first one) announces its conclusions about the submitted information and those conclusions can then be independently verified by all other nodes in the system.

This is the consensus scheme used by Bitcoin (see chapter 3).

2.4.3 Proof of Stake (PoS)

This consensus mechanism is similar to the PoW but in this case the network selects an individual to confirm the validity of new information submitted to the ledger based on the nodes' stake in the network. Therefore, instead of any individual attempting to carry out an intensive computation in order to propose a value, the network itself runs a lottery based on the nodes' stake to decide who will announce the results: the more stake one node has, the higher the probability to be chosen is.

The main idea behind the PoS mechanism is that if a node that has enough stake in the system it means that it has invested enough in the system so that any malicious attempt would outweigh the benefits of performing an attack on the system.

The main problem of this approach is that the system rewards moore those who already are most deeply involved in the network leading consequently to an increasingly centralized system.

This mechanism has been adopted by Peercoin.

2.4.4 Delegated Proof of Stake (DPoS)

This method is an evolution of the PoS whereby each node that has stake in the system can choose an entity to represent their portion of stake in the system by voting. The more stake one node has, the higher is the weight of is vote. The entity with most votes (weighted) becomes a delegate which validates transactions (and collects rewards for doing so).

This method is adopted by Bitshares.

2.5 Types of Blockchain

Blockchain can be distinguished into three different types, each one charaterized by a certain set of attributes.

Public Blockchain

Public Blockchains are blockchains open to the public in which everyone can join the network, mantain the shared ledger and partecipate in the consensus process. The ledger is therefore owned by noone and is publicly accessible by everyone.

These type of Blockchain typically have an incentivizing mechanism to encourage more participants to join the network. Bitcoin for example, one the largest public Blockchain, reward with cryptocurrency miners who join the network.

Public Blockchains have two main disadvantages: the substantial amount of computational power required to maintain a distributed ledger at a large scale and the lack of privacy for the transactions stored inside the blockchain.

Private Blockchain

Private blockchains are private and open only to an organization or a group of individuals. Participants need to obtain an invitation or permission to join the Blockchain and mantain the ledger. Usually the network is permissioned: there are restrictions on who is allowed to participate in the network, and only in certain transactions.

An example of private blockchain with permissioned network is the Linux Foundation's Hyperledger Fabric [14].

Consortium Blockchain

Consortium blockchains are blockchains where the consensus process is controlled by a preselected set of nodes (e.g. a consortium of organization, each of which operates a node). The right to read the blockchain might be public or permissioned. An example of consortium blockchain is R3 [18], which is based on the platform Corda.

3 Bitcoin

3.1 Introduction

Bitcoin it's the first fully decentralized cryptocurrency. It was invented by Satoshi Nakamoto in 2008 and it was the first real implementation of Blockchain. Bitcoin can be either defined as a protocol, a digital currency and a platform.

Bitcoin can be seen as a combination of

- a decentralized peer-to-peer-network (the Bitcoin protocol)
- a public transaction ledger (the blockchain)
- a set of rules for validating transactions (consensus rules)
- a mechanism for reaching distributed consensus on the blockchain (distributed consensus algorithm)

that allows the usage of the digital currency named bitcoin.

From now on, Bitcoin with the capital *B* will refer to the Bitcoin protocol while bitcoin with the lowercase *b* will refer to the bitcoin currency.

Bitcoin is a distributed peer-to-peer system in which users can exchange currency over the network just as it can be done with conventional currency. However, unlike traditional currencies, bitcoins are enterely virtual and thus there are no physical coins. In particular, there are not even virtual coins since they are implied in the transactions that send value from a sender to a receiver: users have private keys which allow them to prove the ownership of bitcoins and sign transactions in order to unlock the value and transfer it to another user. These keys are the only requirement for spending bitcoins and therefore they are protected in wallets stored in the user's devices.

The reference implementation

Bitcoin is an open source project and is developed by a community of volunteers. The first implementation was released by Satoshi Nakamato in 2008 (the only member of the development community at the time). That implementation during the years has been heavily modified and improved evolving into what is known as *Bitcoin Core*, which is now the reference implementation of the Bitcoin system. This implementation is considered the authoritative one and it specifies how each part of the system has to be implemented.

3.2 Scripts

Bitcoin uses a simple stack-based programming language called “Script” for describing how bitcoins can be spent and transferred in order to extend flexibility and support different types of transactions. Essentially, a Bitcoin script a list of instructions recorded with each transaction that describe how the next person wanting to spend the Bitcoins being transferred can gain access to them [20].

Script is a very simple language and it’s not Turing complete. The language has been deliberately designed limiting its operators (it doesn’t have loop operators and complex control flow different than conditional control flow) in order to avoid abuses of the scripts for conducting denial of service attacks, since the transaction scripts has to be executed on each node of the network.

Script supports a number of function called “Opcodes”, uses a reverse polish notation in which every operand is followed by its operators and it’s evaluated from the left to the right using a LIFO stack. Table 1 shows the most common Opcodes while figure 9 shows an example of Script program.

Opcode	**Description**
OP_CHECKSIG	This takes a public key and signature and validates the signature of the hash of the transaction. If it matches, then TRUE is pushed onto the stack; otherwise, FALSE is pushed.
OP_EQUAL	This returns 1 if the inputs are exactly equal; otherwise, 0 is returned.
OP_DUP	This duplicates the top item in the stack.
OP_HASH160	The input is hashed twice, first with SHA-256 and then with RIPEMD-160.
OP_VERIFY	This marks the transaction as invalid if the top stack value is not true.
OP_EQUALVERIFY	This is the same as OP_EQUAL, but it runs OP_VERIFY afterwards.
OP_CHECKMULTISIG	This takes the first signature and compares it against each public key until a match is found and repeats this process until all signatures are checked. If all signatures turn out to be valid, then a value of 1 is returned as a result; otherwise, 0 is returned.

Table 1: Most commonly used Opcodes. Taken from the bitcoin developer’s guide [5].

3.3 Keys and Addresses

As mentioned in this chapter's introduction, ownership of bitcoin is established through digital keys, bitcoin addresses, and digital signatures.

In order to be included in the Bitcoin blockchain, transactions require a valid signature which can be generated only with a private (secret) key. The private key therefore proves the ownership of bitcoins by signing transactions and transferrig value from a user to another. Keys come in pairs consisting of a private (secret) key and a public key and they are generated through Elliptic Curve Cryptography. In analogy with the traditional banking,the public key can be seen as the bank account number while the private key as the secret PIN (or the signature on a check) which provides control over the account by allowing to unlock the value and transferring it to other people.

3.3.1 Adresses

An address is unique string of digits and characters which identify the originator and/or the destination of a transaction. Addresses are derived from public keys through one-way cryptographic hashing in order to obtain the public key fingerprint. In particular, a Bitcoin address is derived by hashing the user's public key it twice, first with the SHA-256 algorithm and then with RIPEMD160. This produces a 160-bit hash, which is then prefixed with a version number and finally encoded using Base58Check encoding. The final result is a 26-35 chatacters string which begins with "1" (public key address) or "3" (pay-to-script-hash address) and it looks like the the string below:

```
1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy
```

The generation process scheme is shown in figure 6.

Base58 and Base58Check Base58 is an encoding scheme which allows to represent long numbers as alphanumeric strings. It is a subset of Base64, which represent numbers using 26 lowercase letters, 26 capital letters, 10 numerals, and 2 more "special" characters and it's usually used to encode email attachments. In particular, Base58 is Base64 without all that characters that are frequently mistaken for one another, namely it is Base64 without the 0 (number zero), O (capital o), l (lower L), I (capital i) and the two special characters. Base58Check is a Base58 encoding with an additional checksum of four bytes added to the end of the data that is being encoded which prevents a mistyped bitcoin address from being accepted by the wallet software as a valid destination.

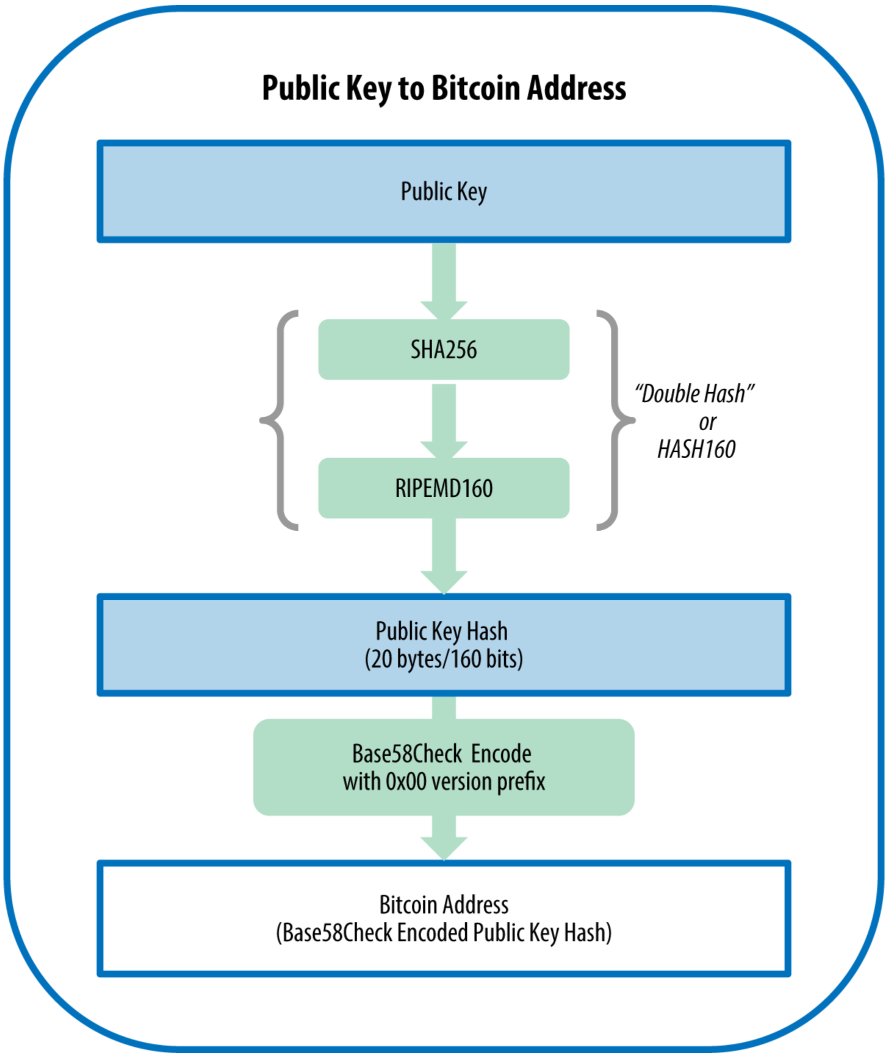

Public Key to Bitcoin Address
Public Key
SHA256
RIPEMD160
"Double Hash" or HASH160
Public Key Hash (20 bytes/160 bits)
Base58Check Encode with 0x00 version prefix
Bitcoin Address (Base58Check Encoded Public Key Hash)

Figure 6: Bitcoin address generation scheme

P2SH and P2PKH As already mentioned before, Bitcoin addresses that begin with the number "3" are pay-to-script hash (P2SH) addresses. Unlike the address which start with "1", also known as pay-to-public-key-hash (P2PKH), which are associated to a public key owned by a user, the P2SH addresses designate the beneficiary of a Bitcoin transaction as the hash of a script. When a user send a bitcoin to a P2PKH address, that bitcoin can only be spent by the receiver by presenting the corresponding private key signature and public key hash associated to its address. When instead the bitcoin is sent to a P2SH address, namely to the hash of a script, the requirements for spending that bitcoin are defined by the script and are usually more restrictive (for example it could be required more than one signature to prove the ownership). A P2SH address is derived from a transaction script in the same way a P2PKH address is derived from a public key (double hashing + Base58Check encoding).

3.3.2 Keys

Public and private keys in Bitcoin are generated through ECC and they can be represented in different formats. All the possible representations, even if they look different, correspond to the same number. This has been done in order to facilitate people to read and transcribe the keys without introducing errors.

Private keys Private keys are simply a 256-bit random number. For generating it, Bitcoin software uses the underlying operating system's random number generators which usually is initialized by a human source of randomness, like for example the elapsed time between the pression of the keys of the keyboard.

Private key formats The private key can be represented in different formats (shown in table 2), each one corresponding to the same 256-bit number. Different formats are used in different circumstances: for example Hexadecimal and raw binary formats are used internally in software while WIF is used by users.

Type	Prefix	Description
Raw	None	32 bytes
Hex	None	64 hexadecimal digits
WIF	5	Base58Check encoding
WIF-compressed	K or L	As above, with added suffix 0x01 before encoding

Table 2: Private key representation formats [1]

Public key generation Public keys are generated starting from the private keys using elliptic curve multiplication, which is a so-called "trap door" function: it is easy to do in one direction (multiplication) and impossible to do in the reverse direction (division). Bitcoin uses the elliptic curve and the set of constants specified by the secp256k1 standard, defined by the NIST. The elliptic curve used is defined by the following equation:

$$y^2 = (x^3 + 7) \text{ over } (\mathbb{F}_p) \tag{1}$$

or, equivalently:

$$y^2 \bmod p = (x^3 + 7) \bmod p \tag{2}$$

where $p = 2^{256}–2^{32}–2^9–2^8–2^7–2^6–2^4–1$ is a very large prime number. Starting from the private key k, the public key K is calculated multiplying it by a predetermined point on the curve called the generator point G (defined by the secp256k1 standard) in order to produce another point somewhere else on the curve, which will correspond to the public key K:

$$K = k * G$$

Since the generator point G is always the same for all bitcoin users, a private key k multiplied with G will always result in the same public key K. The relationship between k and K is fixed and known but it can only be calculated in one direction (from k to K), so it's impossible to derive from an address (derived from K) the corresponding user's private key.

Public key formats In Bitcoin, since ECC is used, a public key in the uncompressed format is a point on an elliptic curve consisting of the coordinates pair (x, y). Uncompressed public keys are presented with the prefix `04` followed by two 256-bit numbers, one for each coordinate, and therefore they are 65 Bytes long. The compress format instead includes only the x-coordiante since the y one can be derived from it and by solving the equation (1) it uses the prefixes `03`, if the y-coordinate is an odd number, or `02`, if it is an even number. The length of a compressed public key is therefore 33 Bytes. Compressed public keys were introduced in order to reduce the size of the transactions, since the most of them also include the public key. The reason why two different prefixes are required for compressed keys is that the left side of the equation (1) is y^2 and therefore the solution for y is a square root, which can have a "positive" or "negative value": graphically, this means that the y-coordiante can either be above or below the x-axis and therefore two different points can be identied since the curve is symmetric. Actually since we are in fhe field $\mathbb{F}_p$ it doesn't make sense talking about positive and negative values: the y-coordinate can in fact be *even* or *odd* (which correspond to the positive/negative terms used before).

Note that a a public key in both compressed and uncompressed formats always corresponds to the same private key, even if the two formats have a different representation. The address derived from the compressed public key however is different from the address derived from the uncompressed one. To solve this issue, compressed private keys have been introduced: a compressed private key is a “private key from which only compressed public keys should be derived”, while uncompressed private keys are “private keys from which only uncompressed public keys should be derived” [1].

3.4 Transactions

Transactions are data structures that encode the transfer of value between participants in the bitcoin system. In important to point out that they are not encrypted and are publicly visible in the blockchain. Blockchain blocks are made up of transactions and these can be viewed using any online blockchain explorer.

3.4.1 Transaction inputs and outputs

A transaction includes at least one input and output: inputs can be seen as coins being spent that the user has created in a previous transaction while outputs as coins being created.

Outputs and UXTO In particular, outputs are discrete and indivisible units of bitcoin measured in *Satoshi*[3], recorded on the blockchain and recognized as valid by the network. All the available and spendable outpus are stored in the blockchain and they are called *unspent transaction outputs* or *UTXO*. The balance shown by Wallets application is nothing more than the aggregated value all the UTXOs the user can spend with the keys it controls. Note that a UXTO can only be spent in its entirety by a transaction, consequently, if an UTXO is larger than the desired value of a transaction, it must still be consumed in its entirety and change must be generated in the transaction (most of the bitcoin transactions generate change). Transaction outputs consist of two parts: an amount of bitcoin (expressed in Satoshis) and a cryptographic puzzle that determines the conditions required to spend the output. This puzzle is also known as a *locking script* and it consists of a digital signature and public key proving the ownership of the UXTO.

[3]One Satoshi = 10^{-8}bitcoins

Inputs Transaction inputs consists of which UTXO will be consumed (can be more than one single UXTO) and a proof of ownership through the unlocking script to unlock the selected UXTO.

3.4.2 Transactions structure

Field	Size	Description
Version Number	4 bytes	Used to specify rules to be used by the miners and nodes for transaction processing.
Input counter	1-9 bytes	The number of inputs included in the transaction.
List of inputs	variable	Each input is composed of several fields, including Previous transaction hash, Previous Txout-index, Txin-script length, Txin-script, and optional sequence number. The first transaction in a block is also called a coinbase transaction. It specifies one or more transaction inputs.
Output counter	1-9 bytes	A positive integer representing the number of outputs.
List of Outputs	variable	Outputs included in the transaction.
lock_time	4 bytes	This defines the earliest time when a transaction becomes valid. It is either a Unix timestamp or a block number.

Table 3: Structure of Bitcoin transactions

3.4.3 Transactions life cycle

This is the typical life cycle of a transaction:

1. A sender sends a transaction (using a wallet application)

2. The wallet signs the transaction using the sender's private key in order to proof the ownership of the value being transferred

3. The transaction is broadcasted to the Bitcoin network using a flooding algorithm.

4. Mining nodes include this transaction in the next block to be mined.

5. Once a mining node solves the Proof of Work problem it broadcasts the newly mined block to the network and the confirmation process starts: each nodes verify the block and propagate it further

6. The receiver start to receive confirmations. After approximately six confirmations, the transaction is considered finalized and confirmed.

3.4.4 Transaction fees

Most transactions include transaction fees. These fees have to purposes: compensate the bitcoin miners and act as a security mechanism by making economically infeasible for attackers to flood the network with transactions. The value of the fees dependens on the size of the transaction since it's calculated by subtracting the sum of the outputs to the sum of the inputs:

$$Fees = Sum(Inputs) – Sum(Outputs)$$

Fees also act as an incentive for miners to encourage them to include a user transaction in the block the miners are creating. Each miner chooses from a memory pool which transactions include in the block he will propose based on their priority: a transaction with a higher fee will be picked up sooner by the miners since it's more profitable.

3.4.5 Coinbase transactions

A particual kind of transaction is the *coinbase transaction*, which is created by the "winning" a miner and is the first transaction in a block. This transactions create brand-new bitcoins that the miner can spend as a reward for mining and do not consume UTXO, instead, they have a special type of input called the *coinbase*.

3.5 The Bitcoin Blockchain

The Bitcoin blockchain is a linked list of blocks of transactions, each one identified by a SHA-256 hash. Each block references the previous one (the *parent block*) by embedding its hash in the header. This chains of hashas goes back all the way to the first block ever created, known as the *genesis block*.

Although a block can have only one single parent, it can temporarily have multiple childrens. This happen during a *blockchain fork*, a temporary situation which occurs when miners solve the proof of work of their block almost simultaneously. Eventually however the forks are resolved and only one child block becomes part of the blockchain.

Modifying a block causes it hash to change. Consequently, since each block contains in its header the hash of its parent block, changing a block causes the child's hash to change, which also requires a change in its child block hash and so on. This cascade effect ensures that once a block has many generations following it, it cannot be changed without forcing a recalculation of all subsequent blocks: since this recalculation requires a huge computation, the blockchain history is pratically immutable. This is a key feature of the Bitcoin security.

3.5.1 The block structure

Table 4 summarize the structure of a block of the blockchain, while table **??** shows the structure of the block header.

Field	**Size**	**Description**
Block size	4 bytes	The size of the block, in bytes.
Block header	80 bytes	Several fields form the block header.
Transactions counter	1-9 bytes	How many transactions the block contains.
Transactions	Variable	The transactions recorded in the block

Table 4: Structure of a Bitcoin block

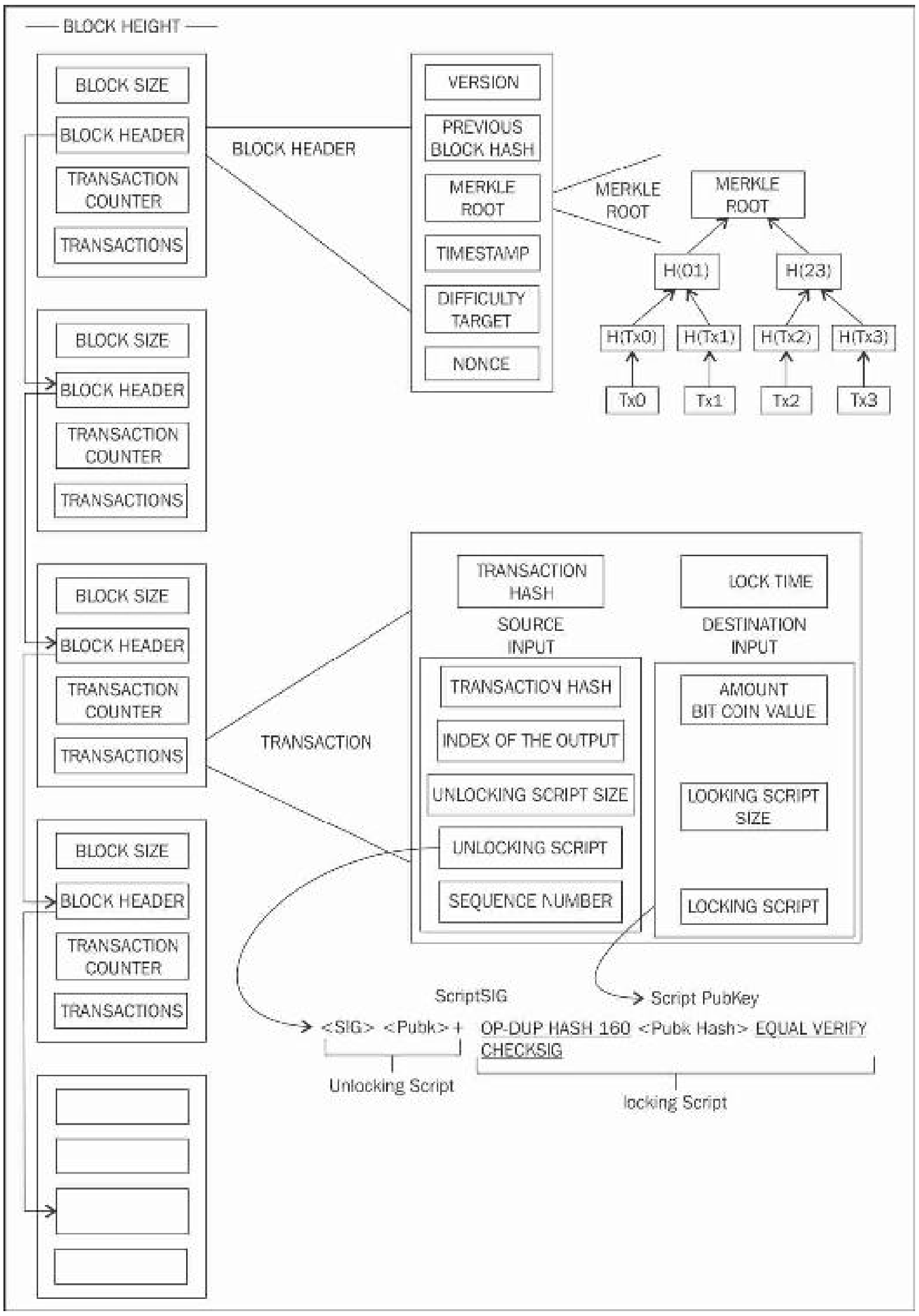


Figure 7: Bitcoin blockchain structure scheme

Field	**Size**	**Description**
Version	4 bytes	A version number to track software/protocol upgrades.
Previous block hash	32 bytes	A reference to the hash of the previous (parent) block in the chain.
Merkle root	32 bytes	A hash of the root of the merkle tree of this block's transactions.
Timestamp	4 bytes	The approximate creation time of this block (seconds from Unix Epoch).
Difficulty target	4 bytes	The Proof-of-Work algorithm difficulty target for this block.
Nonce	4 bytes	A counter used for the Proof-of-Work algorithm.

Table 5: Structure of a Bitcoin block header

3.5.2 Merkle trees

Each block summmarize all the transactions it contains using a Merkle tree, which is a data structure used for efficiently summarizing and verifying the integrity of large sets of data. A Merkle tree is a binary tree containing hashes and it produces an overall digital fingerprint of the entire set of transactions, providing a very efficient method to verify whether a transaction is included in a block. The hash algorithm used in bitcoin's merkle trees is double-SHA256 (SHA256 applied twice).

In the Bitcoin blocks headers only the 32-byte hash corresponding to the tree root is stored, which summarizes all the transactions and allows a node to check whether a specific transaction is included in the block by computing the $log_2(N)$ hashes which make up a *merkle path* connecting the transaction to the root of the tree, with N number of transactions of the block. Figure 8 shows an example of merkle path, while table 6 compares the size of a block to the size of a merkle path.

Thanks to merkle trees, a node can download just the block headers (80 bytes per block) and still be able verify whether a transaction is included in a block by retrieving a small merkle path from a full node (which stores the complete blockchain) instead of storing or retrieving the full block, which is a lot more efficient as pointed out by table 6.

The nodes that do not maintain a full copy of the blockchain are *called simplified payment verification* (SPV) nodes and they use merkle paths to verify transactions without downloading full blocks.

Number of transactions	**Approx size of block**	**Path size**	**Path size**
16 transactions	4 kilobytes	4 hashes	128 bytes
512 transactions	128 kilobytes	9 hashes	288 bytes
2048 transactions	512 kilobytes	11 hashes	352 bytes
65535 transactions	16 megabytes	16 hashes	512 bytes

Table 6: Merkle tree efficiency

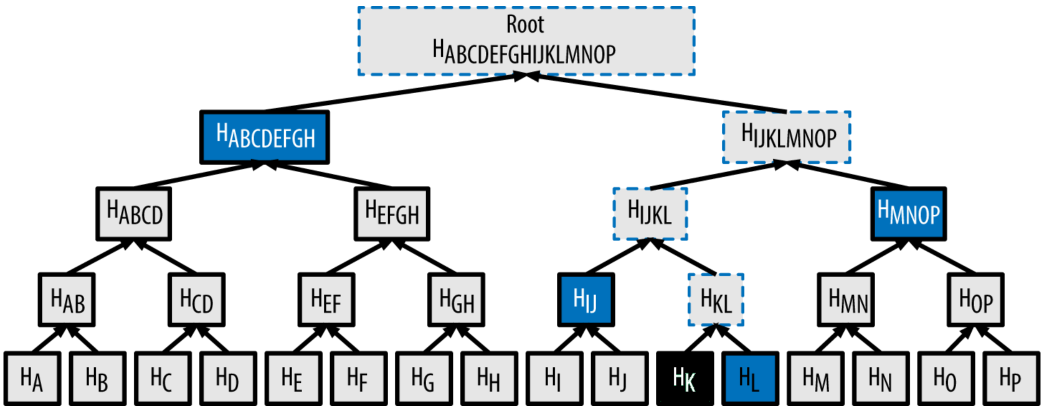


Figure 8: Example of a Merkle path. The path consists of the four hashes with the blue background and with these hashes any node can prove that H_K is included in the merkle root by computing four additional pair-wise hashes outlined in a dashed line.

3.6 Mining and Proof of Work

Mining is a resource-intensive process by which transactions are validated and new blocks are added to the blockchain. Transactions that become part of a block and added to the blockchain are considered confirmed, which means that the receivers of the transactions can spend the value they received.

Roughly one new block is created (*mined*) every 10 minute and Miners after mining a block are rewarded with two types of rewards: new coins created with each new block (a basecoin transaction) and transaction fees from all the transactions included in the block.

Proof of Work In order to earn the reward, miners compete with each other to solve an hard problem based on a cryptographic hash algorithm. The solution to the problem, called the Proof-of-Work, is included in the new mined block and acts as proof that the miner expended significant computing effort. The proof of work requirement is given by the following equation:

$$H(N||Prev\_hash||Tx||Tx||...Tx) < Target \tag{3}$$

where H is the SHA256 hash function, N is the nonce contained in the block header, $Prev\_hash$ is the hash of the previous block, Tx are the transactions cointained in the block, $Target$ is the difficulty value and || is the concatenate operator. For example, if the target is $0x10000000000000$ then finding a hash less than the target means finding a hash that stats with a zero. Consequently, the difficulty level of the proof of work can be seen as the number of zeros that the hash of the block has to start with. The only way for finding a valid hash therefore is to use a the brute force method, changing the nonce value for every hash calculation in order to get different hashes until a valid one in found (any specific hash input to one and only one hash value). Once the miner met the correct number of zeros, the block is immediately broadcasted and accepted by other miners. The difficulty of this work is always adjusted (increased) so as to limit the rate at which new blocks can be generated by the network to one every 10 minutes.

The algorithm for mining a block can be summarized in the following steps:

1. Retrieve the hash of the previous block from the Bitcoin network
2. Choose wich transaction include in the block (according to their priority)
3. Compute the double SHA256 hash of the block header
4. Check whether the resultant hash is lower than the current difficulty level (target). If so, then stop the process, otherwise change the nonce (usually it is increased by 1) and go back to step 3.

3.7 Consensus

Mining is a key feature of Bitcoin which secures the bitcoin system and allows to have network-wide consensus without a central authority. In particular, in Bitcoin consensus is not achieved explicitly since there is no election or fixed moment when consensus occurs. Instead, consensus is an emergent artifact of the asynchronous interaction of thousands of independent nodes. For this reason, in Bitcoin the consensus process is called *emergent consensus*. Bitcoin's decentralized consensus emerges from four processes that occur independently on nodes across the network:

- Independent verification of each transaction by every full node
- Independent aggregation of verified transactions into new blocks by mining nodes and inclusion of the proof of work
- Independent verification of the new blocks by every node and assembly into the chain: each node performs a series of tests for validating it before propagating it to its peers and inserting it into the blockchain. This ensures that only valid blocks are propagated on the network: block which are tampered with will thus be rejected. Thanks to this verification, dishonestly miner (for example miners who write themselves a transaction for an arbitrary amount of bitcoin instead of the correct rewardhave) have their blocks rejected and not only lose the reward, but also waste the effort expended to find a Proof-of-Work solution.
- Independent selection, by every node, of the chain with the most cumulative computation demonstrated through Proof-of-Work

The 51% attack This consensus mechanism is vulnerable to the so-called 51% attack, which can be carried out by a group of miners controlling more than 50% of the total network hashing power. In this situation the attackers would be able to prevent new transactions from gaining confirmations, allowing them to halt payments between some or all users. The attackers would also be able to reverse transactions that were completed while they were in control of the network, meaning they could double-spend coins. This attack is however hypothetical in Bitcoin and even if it was carried out the attacker wouldn't be able to create new coins or alter old blocks.

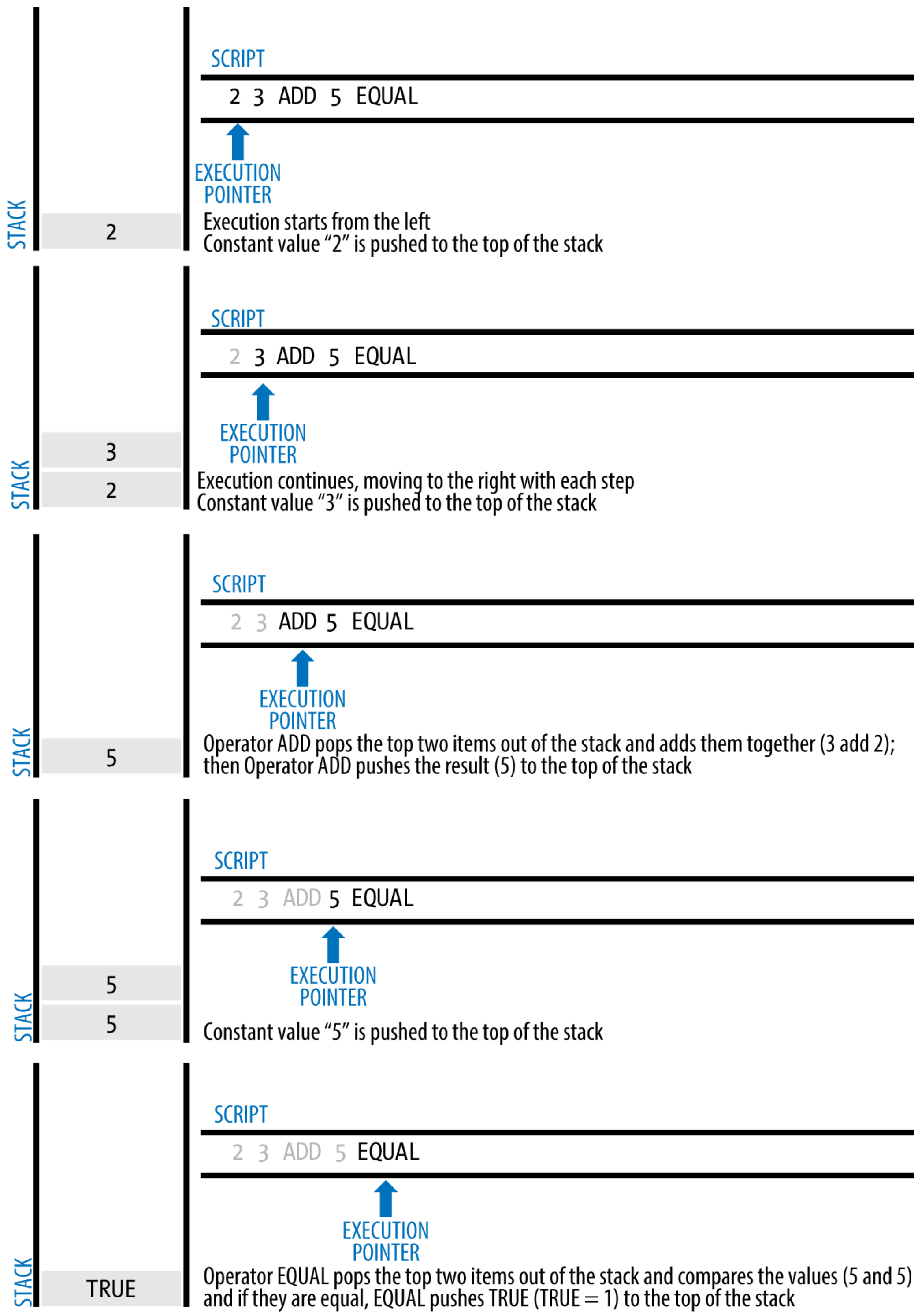

STACK
2
SCRIPT
2 3 ADD 5 EQUAL
EXECUTION POINTER
Execution starts from the left
Constant value "2" is pushed to the top of the stack
STACK
3
2
SCRIPT
2 3 ADD 5 EQUAL
EXECUTION POINTER
Execution continues, moving to the right with each step
Constant value "3" is pushed to the top of the stack
STACK
5
SCRIPT
2 3 ADD 5 EQUAL
EXECUTION POINTER
Operator ADD pops the top two items out of the stack and adds them together (3 add 2); then Operator ADD pushes the result (5) to the top of the stack
STACK
5
5
SCRIPT
2 3 ADD 5 EQUAL
EXECUTION POINTER
Constant value "5" is pushed to the top of the stack
STACK
TRUE
SCRIPT
2 3 ADD 5 EQUAL
EXECUTION POINTER
Operator EQUAL pops the top two items out of the stack and compares the values (5 and 5) and if they are equal, EQUAL pushes TRUE (TRUE = 1) to the top of the stack

Figure 9: Example of a Script program. Image taken from reference [1]

4 Bitcoin anonimity issues

In bitcoin the transactions are exchanged between adresses, which, as explained in the previous chapters, are basically hashes of public keys. The purpose of these adresses is to serve as pseudonyms and provide some anonymity. However, since all Bitcoin transactions are stored in a publicly available ledger and they basically consist of a chain of digital signatures which provide cryptographic proofs of funds transfer, Bitcoin privacy concerns were raised and in the last few years researchers have shown that Bitcoin anonymity is much weaker than was initially expected. Users' transactions in fact can often be easily linked together and if any one of those transactions is linked to the user's identity, then all of its transactions may be exposed. For these reasons Bitcoin is sometimes compared to a bank making bank statements publicly available online but blanking out the names.

Tainted bitcoins The main consequence of this lack of anonimity is that the history of each bitcoin can be traced and therefore some funds (for example stolen bitcoins or bitcoins known to have been used for illegal purposes) can be marked as *tainted*, deflating consequently their value. This can be done for example by warning users to don't accept coins that comes from a given address using alert messages or by coding a list of banned Bitcoin addresses within the official Bitcoin client releases. Also users are less likely to accept coins that are tainted since owning them can be a risk and they can ask the payer to use different coins as payment.

Clients privacy measures Besides addresses, Bitcoin clients adopts some more privacy measures. These measures consist of allowing users to have more than one address and encouraging them to frequently change their adresses by transferring some of their bitcoins to the newly created addresses. Moreover, for each user a new address is automatically created and used for collecting the change resulting from any transaction of the user. These addresses are called *shadow addresses*.

4.1 Compromise of privacy examples

In the following section it will be shown some examples of how user privacy can be compromised by exploiting the existing Bitcoin client implementations and carrying out a behaviour-based analysis of the public ledger [15]. It's important to point out that there are also other kind of attacks which operates at the network layer and which allow the attacker to obtain user information from the Bitcoin peer to peer network [15], which however will not be discussed in this book.

Exploiting multi input transactions The first method for obtain users information consists of observing the multi-input transactions. A multi-input transaction, as discussed in chapter 3.4, is a transaction which accept more UXTO as input. If in a transaction these UXTOs are owned by different addresses, then it is straightforward to conclude that the input addresses belong to the same user.

Exploiting shadow addresses Another method is to exploit the shadow addresses generated by the Bitcoin clients. When a Bitcoin transaction has n output addresses $\{a_1, \dots, a_n\}$ (transaction with multiple recipients) and only one address is a new address (namely the address has never appeared in the ledger before) it is then possible to assume that the newly appearing address is a shadow address for the user that sent the transaction.

Behavior-based analysis Besides exploiting Bitcoin client implementations, an attacker can also use behavior-based clustering algorithms like K-Means (KMC) and Hierarchical Agglomerative Clustering (HAC) for profiling Bitcoin users. Without going into the datails of these techniques, in reference **??** these algorithms has been tested in a simulated Bitcoin system and the achieved results shows that given 200 simulated user profiles, almost 42% of the users have their profiles captured with 80% accuracy and the profile leakage in Bitcoin is larger when users participate in a large number of transactions, while decreases as the number of transactions performed by the user decreases.

5 Enhancing Bitcoin privacy

There are basically two approaches for enhancing users privacy in Bitcoin:

- Mixing services: they achieve users privacy generally without degrading the performance of the system. However, they require absolute trust in a third party.
- Cryptograpic extensions of Bitcoin: extensions of the Bitcoin protocol which eliminate the need for trusted third parties but tend to be less efficient in terms of performance.

5.1 Mixing services

A bitcoin mixing service act as mediators and provides anonymity by transferring payments from an input set of bitcoin addresses to an output set of bitcoin addresses, such that is it hard to trace which input address paid which output address, as schematized in figure 10. Examples of this kind of mixing services are Mixcoin and CoinParty. The former relies on a third party can violate users privacy and steal users' bitcoins (theft is detected but not prevented), while the second uses more mixing parties and it is considered secure only if 2/3 of the mixing parties are honest.

There is also another kind of mixing services in which the service acts as a "coin hisory resetter". In this case the user sends to the mixer a certain amount of bitcoin and a return address and the mixer sends back to the user (to the specified address) someone else's coins of the same value. Examples of this kind of services are BitLaundry and Bitcoin Fog. The problem of these services is that they do not protect form network-layer attacks since the eventually the user is the one making payments (instead of the mixing service).

5.2 Enhancing privacy through blind signatures

In the following section is presented a mixing service scheme for enhancing Bitcoin privacy through the use of blind signatures. The scheme has been proposed by E. Heilman, F. Baldimtsi and S. Goldberg [13], is based on the scheme used in eCash [9] and, unlike other previous schemes that are efficient but achieve limited security/anonymity or other which provide strong

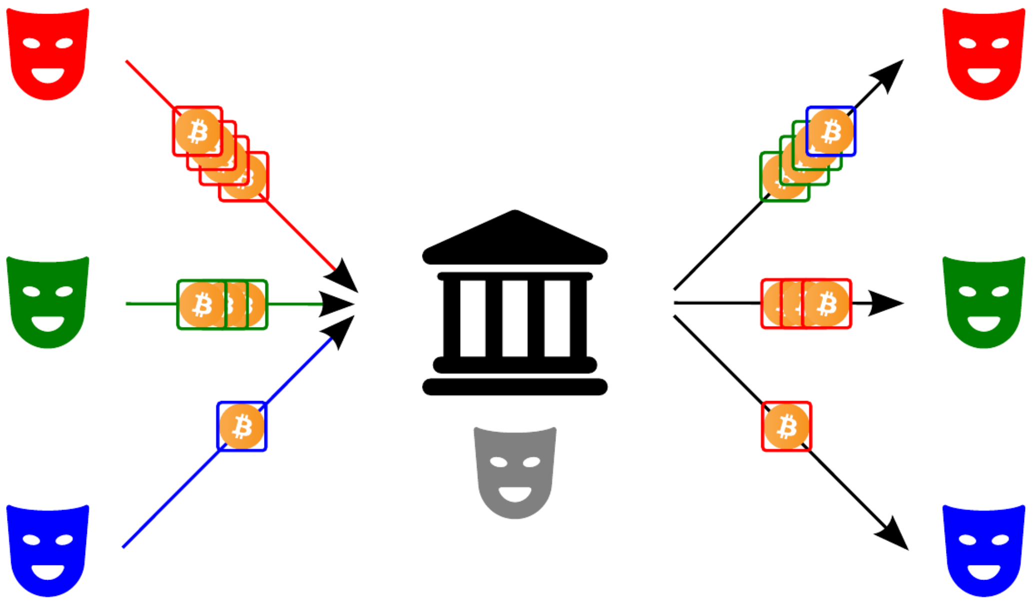


Figure 10: Scheme of how a mixing service works

anonymity but are slow and require large numbers of transactions, it provides anonimity at reasonable speed using an untrusted third party (which can therefore be malicious).

5.2.1 High-level overview of the scheme

Scenario The scenario is the following: A, *the payer*, wants to anonymously send 1 bitcoin to B, *the payee*. If A performed a standard transaction sending 1 BTC from $address_A$ (owned by A) to a fresh ephemeral address $address_B$ (owned by B), there would be a record in the blockchain linking the two addresses. Even if A and B always create a fresh address for each payment they receive, the links between addresses can be used to de-anonymize users when they for example have a transaction with a third party which learns their identify (e.g. their email address). The basic idea is to used a third party I that breaks the link between A and B addresses: A sends coins to I and I sends different coins for the same value to B, acting thus as a mixing service. If other users use I and enough transactions pass through it, it becomes difficult for an attacker to link A and B.

The main problem is that I knows everything about the transactions between A and B.

eCash scheme A possible solution to this issue is the scheme used in eCash for preventing I from knowing who A wants to pay. This scheme is shown in figure 11 and relies on blind signatures. A chooses a random serial number sn, blinds it to $\overline{sn}$ and asks I to compute a blind signature $\overline{\sigma}$ on $\overline{sn}$, which sends back to A. A unblinds these values to obtain $V = (sn, \sigma)$ and then pays B using the voucher V. Finally, B redeems V with I to obtain the bitcoin. With this scheme I does not know who A wants to pay it cannot read the blinded serial number $\overline{sn}$ that it signs and it cannot link a message/signature (sn, σ) pair to its blinded value $(\overline{sn}, \overline{\sigma})$. Blindness therefore ensures that I cannot link a voucher it redeems with a voucher it issues. Blind signatures are also unforgeable, which ensures that a malicious user cannot issue a valid voucher to itself.

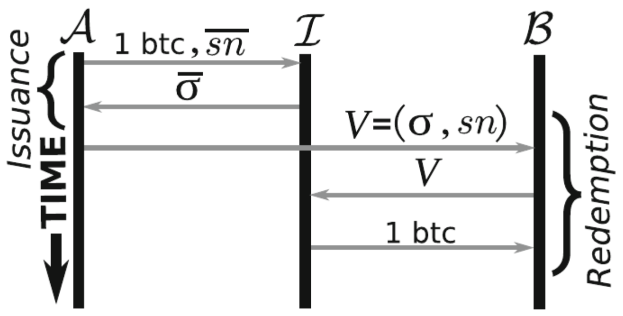


Figure 11: eCash protocol scheme

Heilman scheme The main problem with the eCash approach is that I has to be honest: if I is malicious it could refuse to issue a voucher to A after receiving its bitcoin and thus the scheme fails. To solve this, the scheme proposed in [13] uses Bitcoin *transaction contracts* for achieving blockchain-enforced *fair exchange*. An high level view of the scheme is shown in figure 12. The scheme consists of four blockchain transactions that are confirmed in three blocks on the blockchain and the key idea is that A transfers a bitcoin to I if and only if it receives a valid voucher V in return. The four transactions implement two fair exchanges:

- $V \rightarrow BTC$, which consists of the transactions (1) $T_{offer(V \rightarrow BTC)}$ and (2) $T_{fulfill(V \rightarrow BTC)}$ and it ensures that a malicious I cannot redeem B's voucher without providing B with a bitcoin in return. The exchange stands in for the interaction between B and I. Transaction (1) offers a fair exchange of one bitcoin (from I) for one voucher (from B), while transaction (2) is created by B to meet the offer by I.

- $BTC \to V$, which consists of the two transactions (1) $T_{offer(BTC \to V)}$ and (2) $T_{fulfill(BTC \to V)}$ and it ensures that a malicious I cannot take a bitcoin from A without providing it with a voucher V.

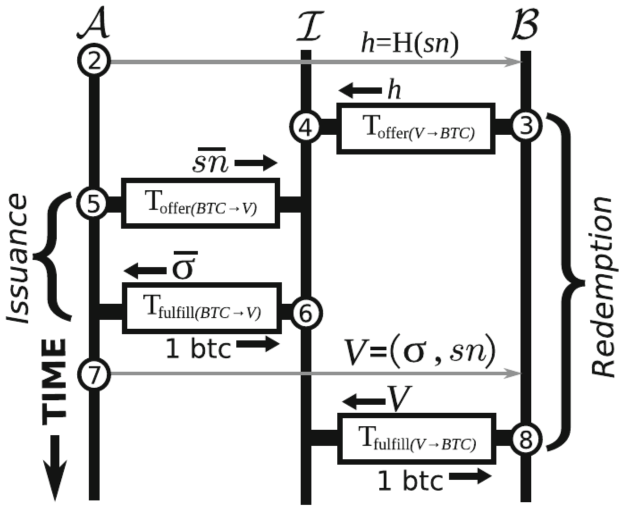


Figure 12: High-level view of the scheme proposed in [13]

5.2.2 Fair exchange implementation

In the following section it will be explained how the transaction contracts $T_{offer(BTC \to V)}$ and (2) $T_{fulfill(BTC \to V)}$ implement the fair exchange $BTC \to V$ used in our protocol scheme shown in figure 12. The implementation for the exchange $V \to BTC$ is analogous.

As already mentioned, the fair exchanges are achieved using transaction contracts. A transaction contract can be implemented using *Script*, a simple programming language provided by Bitcoin which, as discussed in chapter 3.2, allows to associate each transaction with a script which defines the rules for spending the transaction outputs, namely how the next person wanting to spend the bitcoins being transferred can gain access to them.

In this case, the CHECKLOCKTIMEVERIFY feature of Script is used in order to timelock a transaction, so that funds can be reclaimed if a contract has not been spent within a given time window tw. For implementing the $BTC \to V$ fair exchange, A generates the transaction contract $T_{offer(BTC \to V)}$ which says the following:

"A offers bitcoins to I under the condition that I must compute a valid blind signature on the blinded serial number $\overline{sn}$ (provided by A) within time window tw. If this condition is not satisfied, the bitcoin reverts to A."

The output of this transaction therefore can be spent in a future transaction T_f only if one of the following conditions is met:

1. T_f is signed by I and contains a valid blind signature $\overline{\sigma}$ on $\overline{sn}$
2. T_f is signed by A and the time window tw has expired.

For fulfilling the contract and acquiring the bitcoins of the transaction I has therefore to satisfy the condition 2, which means that I has to post a transaction $T_{fulfill(BTC \to V)}$ that contains a valid blind signature $\overline{\sigma}$ on $\overline{sn}$. If I does not fulfill the contract within the time window tw, then the condition 2 is met when A signs and posts a transaction T_f that returns back the offered bitcoins.

5.2.3 Blind signature scheme

The scheme used for the blind signature is the Boldyreva's scheme [7], which requires two rounds of interaction. Since the elliptic curve defined by the standard Secp256k1 used by Bitcoin doesn't support the bilinear pairings required for the adopted signature scheme, for using the scheme it is necessary to slightly modify the Bicoin protocol in order to adopt a different elliptic curve.

Let $\mathbb{G}$ be a cyclic additive group of order p (with p prime) in which the Diffie-Hellman problem is hard and $\mathbb{G}'$ a cyclic multiplicative group of prime order q. Let $e\colon \mathbb{G} \times \mathbb{G} \to \mathbb{G}'$ be the bilinear pairing, g be a generator of the group $\mathbb{G}$ and H be a hash function mapping arbitrary strings to elements of $\mathbb{G} \setminus \{1\}$. The public parameters are (p, g, H) while the signer public/private key pair is $(sk, pk = g^{sk})$. The signature scheme works as follow:

- To blind sn, user A picks random $r \in \mathbb{Z}_p^*$ and sets $\overline{sn} = H(sn)g^r$.
- To sign $\overline{sn}$, signer I computes $\sigma = \overline{sn}^{sk}$.
- To unblind the blind signature $\overline{\sigma}$, user A computes $\sigma = \overline{\sigma} pk^{-r}$.
- To verify the signature σ on sn, anyone holding pk checks that the bilinear pairing $e(pk, H(sn))$ is equal to $e(g, \sigma)$. For verifying the blinded signature $\overline{\sigma}$ on the blinded $\overline{sn}$, anyone holding pk checks if $e(pk, \overline{m}) = e(g, \overline{\sigma})$.

5.2.4 Anonymity considerations

While in the eCash protocol of figure 11 the anonymity level of users depends on the total number of payments using I, in the scheme proposed by Heilman shown in figure 12 the anonimity level depends on the number of payment through I in a given epoch. The protocol in fact runs in epochs and provides set-anonymity within each epoch. An epoch is the three-blocks window in which the four transaction required by the protocol are confirmed and stored, as shown in figure 13.

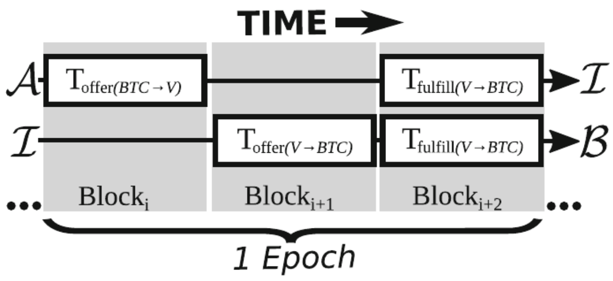


Figure 13

The anonimity considerations about the proposed scheme are based on the following assumptions:

- all the users coordinate on epoch so that the transactions arrangement shown in figure 13 is respected (e.g. they choose the starting block so that its height[4] is multiple of three)
- the payer A and the payee B trust each other. If A or B were malicious they could easily conspire with I revealing the other part identity: for exmple A could comunicate I the serial number of the received voucher so that when I can identify B when he redeems that voucher
- payees B always receive payments in a fresh ephemeral address $address_B$
- payers only make one anonymous payment per epoch. Similarly, payees only accept one payment per epoch.

Epoch set-anonimity As consequence of assumptions (2) and (3), in every epoch there are exactly n addresses making payments (playing the role of payer A) and n receiving addresses (playing the role of B). Anyone looking

[4]height=distance from the genesis block

at the blockchain can therefore see the participating addresses of payers and payees, but the probability of successfully linking any chosen payer A to a payee B should not be more than $1/n$, namely an attacker observing the blockchain can do no better than randomly guessing who paid whom during an epoch.

Transparency of anonimity Users in learn the size of their anonymity set (number of partecipants in their epoch) only after a transaction completes by looking at the blockchain. If particular B feels his anonymity set is too small in one epoch, he can increase the size of it by using the scheme as a mixing service and making a new transaction to another address owned by him. For example: $address_B$ gets paid in an epoch with $n = 4$. B can create a fresh ephemeral address $address'_B$ and have $address_B$ pay $address'_B$ in a subsequent epoch. If the subsequent epoch has a $n = 100$, then B increases the size of his anonymity set.

Intersection attack As pointed out by the autors in [13], there's the possibility for an attacker (or anyone looking at the blockchain) to attempt an intersection attack that de-anonymaze users across different epochs by using frequencial analysis.

6 Bitcoin and Blockchain scalability

6.1 Introduction

As a consequence of the increasing adoption of Blockchain-based cryptocurrencies, their ability to scale has raised concerns and has received a lot of attention in the last few years. In particular, the key concerns are:

- can cryptocurrencies based on decentralized blockchains be scaled up to match the performance of a mainstream payment processor?
- what does it take to get there?

Bitcoin current performance As reference, Bitcoin today requires around 10 minutes to confirm a transaction (a new block is mined every ~ 10 minutes) and achieves a maximum throughput of 7 transaction/sec [6]. Since the transactions are confirmed only after the block they belong to is created and added to the blockchain, the maximum throughput of Bitcoin is effectively capped at maximum block size divided by block interval. In comparison, a payment processor such as Visa credit card processes 2000 transactions/sec on average, with a peak rate of 56,000 transactions/sec [6].

Bitcoin reparametrization A solution for increasing Bitcoin throughput could therefore be to change the block interval time and to increase the size of the block, which is currently 1MB, in order to increase the amount of transactions confirmed every 10 minutes. In the last few years there's been a debate about this topic which splitted the community. People in favour for increasing the size claim that increasing it would allow Bitcoin to easlily reach VISA (and anolog payment systems) numbers, while the opposing ones claim that this would damage decentralization because blocks of big size require a lot of computational power for being mined and this increases the costs of participation, centralizing the miners in a few powerfull nodes. Their proposed solutions therefore consist of spending effort for optimizing the use of the current block space available and offloading certain processing to off-chain networks (off-chain solutions).

As discussed in [], since scalability is not a singular metric and it includes various performance and security metrics, reparametrization can achieve only limited benefits considering the network performance given by Bitcoin's current peer-to-peer network protocol and the willing to maintain its current degree of decentralization. However, it is still an open question wheter reparametrization alone can address the growth of Bitcoin to the same order of magnitude of systems like the previously mentioned VISA. Following the considerations discuessed in [], the next section will explore the reparametrization limitations which shows that likely the scaling problem of Bitcoin (and, more in general, of Blockchain systems) cannot be faced with reparametrization alone.

Interesting point offerend by article ***: their conclusion is that fundamental protocol redesign is needed for blockchains to scale significantly while retaining their decentralization (reparametrization only is not enough, as explained in SECTION 3). They also discuss about new strategies for designing new protocols by addressing blockchain limitations through a partition of the system in different layers, analyzing the bottlenecks and the limitations of each layer (SECTION 4).

References

[1] A.M. Antonopoulos. *Mastering Bitcoin: Programming the Open Blockchain.* O'Reilly Media, 2017. ISBN: 9781491954362. Available at: https://books.google.it/books?id=MpwnDwAAQBAJ.

[2] J.J. Bambara et al. *Blockchain: A Practical Guide to Developing Business, Law, and Technology Solutions.* McGraw-Hill Education, 2018. ISBN: 9781260115864. Available at: https://books.google.it/books?id=z5hIDwAAQBAJ.

[3] I. Bashir. *Mastering Blockchain.* Packt Publishing, 2017. ISBN: 9781787125445. Available at: https://books.google.it/books?id=dMJbMQAACAAJ.

[4] Iddo Bentov, Ariel Gabizon, and Alex Mizrahi. "Cryptocurrencies Without Proof of Work". In: *Financial Cryptography and Data Security.* Ed. by Jeremy Clark et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 142–157. ISBN: 978-3-662-53357-4.

[5] *Bitcoin Developer Guide.* Available at: https://bitcoin.org/en/developer-guide (visited on 04/08/2018).

[6] *Bitcoin scalability problem.* Aug. 2018. Available at: https://en.wikipedia.org/wiki/Bitcoin_scalability_problem (visited on 08/08/2018).

[7] Alexandra Boldyreva. "Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme". In: *International Workshop on Public Key Cryptography.* Springer. 2003, pp. 31–46.

[8] Miguel Castro, Barbara Liskov, et al. "Practical Byzantine fault tolerance". In: *OSDI.* Vol. 99. 1999, pp. 173–186.

[9] David Chaum. "Blind Signature System". In: *Advances in Cryptology: Proceedings of Crypto 83.* Ed. by David Chaum. Boston, MA: Springer US, 1984. ISBN: 978-1-4684-4730-9. DOI: 10.1007/978-1-4684-4730-9_14. Available at: https://doi.org/10.1007/978-1-4684-4730-9_14.

[10] David Chaum. "Blind Signatures for Untraceable Payments". In: *Advances in Cryptology.* Ed. by David Chaum, Ronald L. Rivest, and Alan T. Sherman. Boston, MA: Springer US, 1983, pp. 199–203. ISBN: 978-1-4757-0602-4.

[11] Kyle Croman et al. “On Scaling Decentralized Blockchains”. In: *Financial Cryptography and Data Security*. Ed. by Jeremy Clark et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 106–125. ISBN: 978-3-662-53357-4.

[12] Chris Hammerschmidt. *Consensus in Blockchain Systems. In Short.* 2017. Available at: https://medium.com/@chrshmmmr/consensus-in-blockchain-systems-in-short-691fc7d1fefe (visited on 02/08/2018).

[13] Ethan Heilman, Foteini Baldimtsi, and Sharon Goldberg. “Blindly Signed Contracts: Anonymous On-Blockchain and Off-Blockchain Bitcoin Transactions”. In: *Financial Cryptography and Data Security*. Ed. by Jeremy Clark et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 43–60. ISBN: 978-3-662-53357-4.

[14] *Introduction — hyperledger-fabricdocs master documentation*. Available at: https://hyperledger-fabric.readthedocs.io/en/release-1.2/whatis.html (visited on 02/08/2018).

[15] G. Karame and E. Androulaki. *Bitcoin and Blockchain Security*. Artech House information security and privacy series. Artech House, 2016. ISBN: 9781630810139. Available at: https://books.google.it/books?id=b%5C_nwjwEACAAJ.

[16] G. Konstantopoulos. *Understanding Blockchain Fundamentals, Part 1: Byzantine Fault Tolerance*. 2017. Available at: https://medium.com/loom-network/understanding-blockchain-fundamentals-part-1-byzantine-fault-tolerance-245f46fe8419 (visited on 01/08/2018).

[17] Leslie Lamport, Robert Shostak, and Marshall Pease. “The Byzantine generals problem”. In: *ACM Transactions on Programming Languages and Systems (TOPLAS)* 4.3 (1982), pp. 382–401.

[18] *r3.com*. Available at: https://www.r3.com/ (visited on 02/08/2018).

[19] Amitabh Saxena, Janardan Misra, and Aritra Dhar. “Increasing Anonymity in Bitcoin”. In: *Financial Cryptography and Data Security*. Ed. by Rainer Böhme et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 122–139. ISBN: 978-3-662-44774-1.

[20] *Script*. Available at: https://en.bitcoin.it/wiki/Script (visited on 04/08/2018).