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 [10],[1]

   3.1. Bitcoin protocol specification

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:

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

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 length of the hash. 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 paris 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 unsecure 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 unsecure 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.

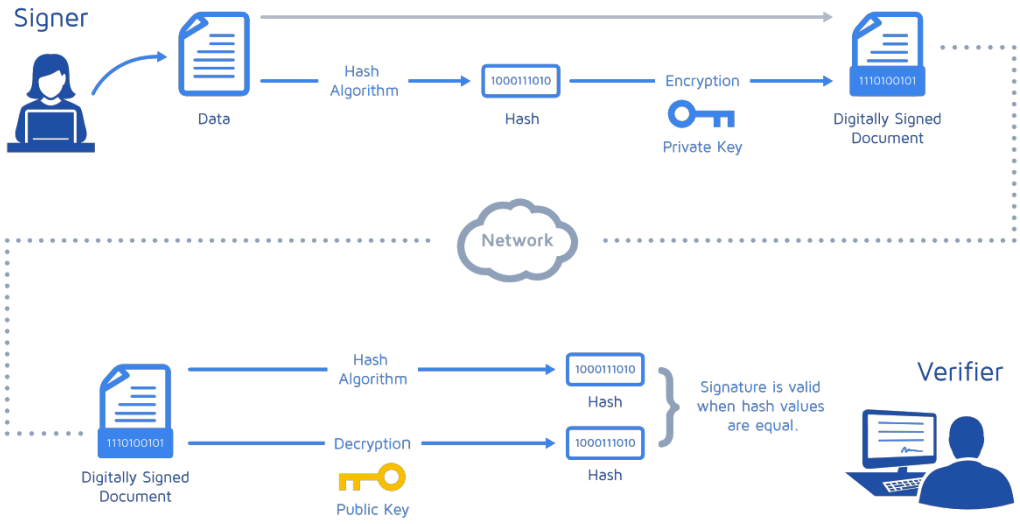

Figure 1: digital signature signing and verification scheme

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 Distributed systems

1.4.1 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 act in a coordinated way (passing message 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*.

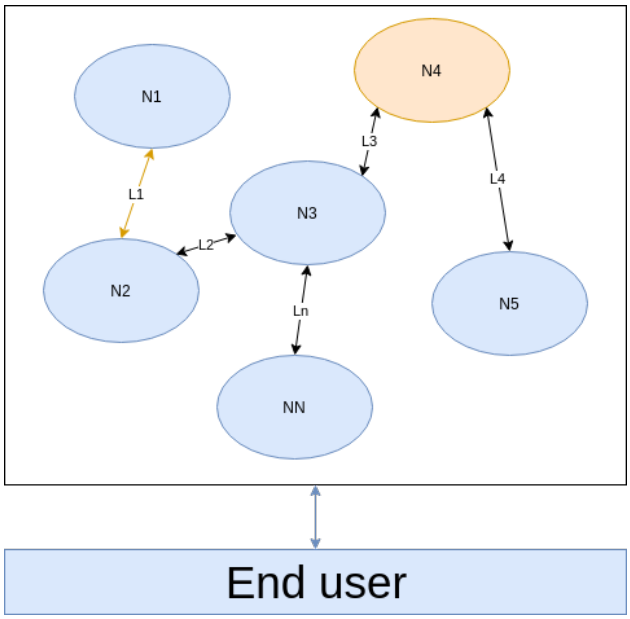

Figure 2: design of a distributed system. N4 is a Byzantine node while L1 is a broken/slow network link

The main challenge in a distributed system is the fault tollerance: even if some of the nodes fault or links break, the system should tollerate 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.

1.4.2 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).

1.4.3 The Byzantine Generals Problem (BGP)

The Byzantine Generals Problem (BGP) is a problem described by Leslie Lamport [12] 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 wheter 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).

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 [12]

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 specic one). The original algorithm proposed by Lamport is shown in figure 4.

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

1.4.4 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 [11]. 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.

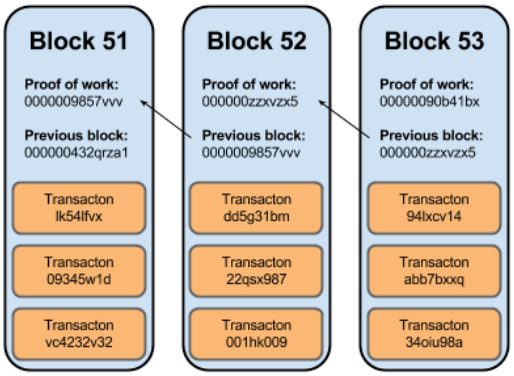

Figure 5: basic blockchain schema

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

2.4.1 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 [5]).

Simplifying, the algorithm works as follows [7], [5]: 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 [9].

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 [13], 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 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.

3.2.1 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.

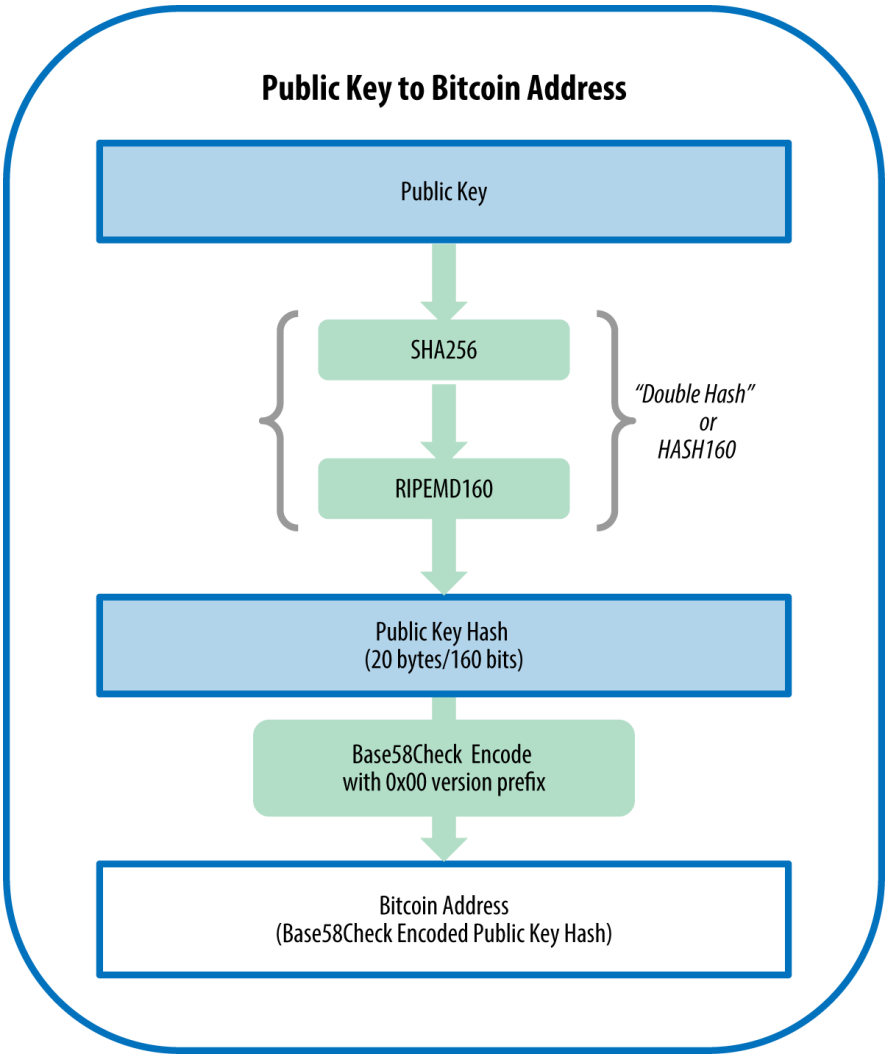

Figure 6: Bitcoin address generation scheme

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 re-

strictive (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).

3.2.2 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 1), 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 1: 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].

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

[6] 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.

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

[8] 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.

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

[10] 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.

[11] 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 08/01/2018).

[12] 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.

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

[14] 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.