Blockchain and cryptocurrencies

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 [7],[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 [6],[9])

5. Bitcoin blockchain scalability

- Considerations on the scalability of the Bitcoin blockchain and possibile solutions [7],[5]

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:

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

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 length of the hash.

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 **2$^{\text{nd}}$ 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.

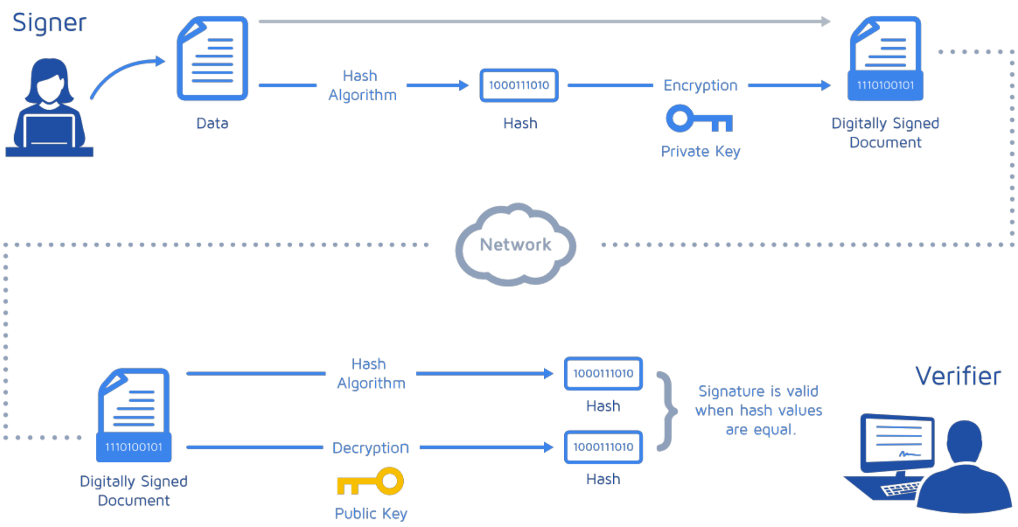

Figure 1: digital signature signing and verification scheme

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*.

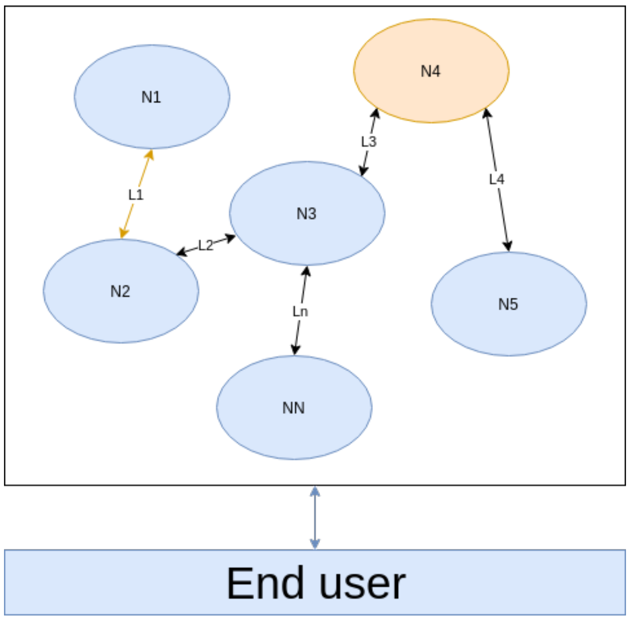

Figure 2: design of a distributed system. N4 is a Byzantine node while L1 is a broken/slow network link

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.

The 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 The Byzantine Generals Problem (BGP)

The Byzantine Generals Problem (BGP) is a problem described by Leslie Lamport [8] in which a group of generals, each one leading a portion on the Byzantine army, 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 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.

1.4.3 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*).

A consensus mechanism must meet these requirements:

- **Agreement**: all the 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).

2 First

asdasd

2.1 test

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] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.