mathematics

MDPI

Review

Analysis of the Cryptographic Tools for Blockchain and Bitcoin

Víctor Gayoso Martínez, Luis Hernández-Álvarez and Luis Hernández Encinas *

Institute of Physical and Information Technologies (ITEFI), Spanish National Research Council (CSIC), Serrano 144, 28034 Madrid, Spain; victor.gayoso@iec.csic.es (V.G.M.); luis.hernandez@csic.es (L.H.-Á.)
* Correspondence: luis@iec.csic.es; Tel.: +34-91-561-88-06

Received: 10 December 2019; Accepted: 11 January 2020; Published: 15 January 2020

Abstract: Blockchain is one of the most interesting emerging technologies nowadays, with applications ranging from cryptocurrencies to smart contracts. This paper presents a review of the cryptographic tools necessary to understand the fundamentals of this technology and the foundations of its security. Among other elements, hash functions, digital signatures, elliptic curves, and Merkle trees are reviewed in the scope of their usage as building blocks of this technology.

Keywords: asymmetric cryptography; bitcoin; blockchain; digital signature; elliptic curves; hash function; merkle tree

MSC: 94A15; 94A60; 94A62

1. Introduction

Blockchain is one of the emerging technologies generating more headlines in the last years. In brief, this technology can be defined as a linked chain of data blocks that allows the creation of transaction records (financial, contractual, etc.) based on a distributed consensus protocol managed by the participants (i.e., the nodes of the network) lacking a central authority. By construction, the chain of records becomes immutable; that is, no single node can modify the content of the blocks that have been previously agreed. In other words, only insertions or aggregations of new transactions are allowed, as it is not possible to eliminate or modify existing ones.

The immutability property is complemented with additional characteristics. Firstly, it must be possible to obtain a summary of the status of the entire chain at any given time, so that, if any block of the chain were manipulated, it must be possible to detect such manipulation. Secondly, it would be desirable to have access to a simple way for verifying whether a transaction has been incorporated to the blockchain or not. Finally, the parties involved in a transaction to be included in any of the blocks should be allowed to do so in a pseudo-anonymous manner.

Possibly, one of the best-known implementations of this technology is bitcoin, the cryptocurrency named that way for the use that it makes of several cryptographic primitives to achieve pseudo-anonymity of the participants, immutability of stored records, and distributed consensus without resorting to a central authority.

In this article, we review the most elementary cryptographic concepts that are necessary to understand the fundamentals of blockchain technology. The main tool used for ensuring the integrity of information are the hash functions; the verification that a piece of information comes from the legitimate sender is carried out through digital signature schemes (some of them using elliptic curves) and Merkle trees. Thus, the concepts of hash functions, digital signatures, elliptic curves, and Merkle trees are presented.

The rest of this article is organized as follows. Section 2 reviews the hash functions that allows verifying data integrity. Section 3 presents the concepts associated to elliptic curves that help to understand some of the digital signature schemes discussed in Section 4. Section 5 focuses on the properties of Merkle trees. The result of applying the cryptographic tools mentioned in the previous sections to blockchain technology is shown in Section 6, with particular attention to bitcoin. Finally, conclusions are presented in Section 7.

2. Hash Functions

2.1. Definition

Hash functions are among the cryptographic primitives that have increased their relevance in recent years. It is important to note that such functions do not encrypt or decrypt messages. However, they are an indispensable tool for verifying data integrity, apart from other applications equally interesting.

Hash functions can be defined as functions that are capable of transforming any block of binary data into another fixed-size binary block [1,2]. The result of such a transformation is called hash or digest.

The first hash functions were proposed to be used in digital signature protocols with the goal to improve their efficiency, as signatures were constructed by using the digest of data elements instead of the whole elements. Being the hash a much shorter element, the protocol was more efficient since the calculations were simpler and less bandwidth was needed when sending the signed data.

In addition to this initial use, hash functions have been applied to other areas related, in general, to the protection of information and, in particular, to its integrity. Thus, they are also used to detect corrupted data, presence of viruses, etc.

From a mathematical point of view, hash functions are created using the concept of Trapdoor One-Way Functions (TOWF), which are functions defined between the sets X and Y

$$f\colon X \to Y, \text{ with } f(x) = y$$

that fulfill the following conditions:

1. f is a unidirectional function, thus, from a computational standpoint, it must be easy to compute $f(x) = y$ for all elements $x \in X$ but, at the same time, it must be very difficult to obtain $x = f^{-1}(y)$ for a given value $y \in Y$.
2. If additional information, known as trapdoor, is learned, then it must be feasible to calculate in polynomial time an element $x \in X$ so that $f(x) = y$.

Thus, a hash function is a unidirectional function that is applied to a message m of variable size, where the message belongs to a certain set of messages, M, and provides a digest of the message with a fixed, predetermined bit size, n. Therefore, a hash function, $\mathfrak{h}$, can be described as follows:

$$\mathfrak{h}\colon M \to \{0,1\}^n, \text{ with } \mathfrak{h}(m) = \widehat{m}.$$

Since hash functions transform a message of any length into a collection of n bits, the number of possible hashes is much smaller than the number of different input messages. Consequently, there will always be different messages whose digests match.

2.2. Properties

Other important properties that these functions must fulfill are the following:

1. *Bit dependency*: The hash of a message, $\mathfrak{h}(m) = \widehat{m}$, must be a complex function dependent on all the bits of the message, so that, if a bit of the message is changed, its hash must change, approximately, half of the bits.

2. *Preimage Resistance*: Given a hash $\widehat{m}$, it must be computationally difficult to get a message m so that $\mathfrak{h}(m) = \widehat{m}$. In other words, from a computational point of view, any hash function must be difficult to reverse.
3. *Resistance to the second preimage*: Given a message m_1, it must be computationally difficult to find another message, m_2, $m_1 \neq m_2$, with the same hash. That is to say, it must not be possible to find another message such that $\mathfrak{h}(m_1) = \mathfrak{h}(m_2)$.
4. *Collision Resistance*: It must be computationally difficult to find two messages m_1 and m_2, $m_1 \neq m_2$, so that $\mathfrak{h}(m_1) = \mathfrak{h}(m_2)$.

Property 1 allows guaranteeing the integrity of information, since, if any number of bits are modified, the result of the function will show a great difference between the original hash and the new hash, thus trial and error attacks are not feasible.

On the other hand, it is worth mentioning that, despite being similar, the last two properties are actually are different. Property 3 considers that one of the messages is known and tries to locate a different message with the same hash. In comparison, in Property 4, no conditions are imposed on the messages. Based on the birthday paradox, collision resistance is a weaker condition than resistance to the second preimage. If a hash function is vulnerable to collision resistance, its use will no longer be recommended.

The compliance with the above properties guarantees, at least initially, that the functions that fulfill them are not vulnerable and prevent that, taking as input the hash of a message, the content of the message can be retrieved and that no one will be able to find another message with the same hash.

2.3. Widely-Used Hash Functions

For quite some time, the most widely-used hash function was MD5 (Message Digest 5), proposed by Rivest [3], which generates hashes of 128 bits. However, the MD5 function stopped being used when some vulnerabilities were published in 2005 [4]. In fact, even though it is still used in non-secure contexts such as checking the integrity of downloaded files, in environments where security is a critical aspect, its use is forbidden.

The SHA-1 (Secure Hash Algorithm-1) function, which generates 160-bit hashes, was adopted by the National Institute of Standards and Technology (NIST) in 1995 [5]. SHA-1 continues to be used quite frequently today, although collisions have been found [6]. As in the case of MD5, its use is absolutely discarded by most international organizations and institutions in the scope of secure applications. Another function that provides 160-bit hashes and is still used today in some scenarios is the RIPEMD-160 (RACE Integrity Primitives Evaluation Message Digest) function [7].

The SHA-2 family of hash functions is the successor of the SHA-1 function. The SHA-2 specification includes the SHA-224, SHA-256, SHA-384, and SHA-512 functions, which provide hashes of 224, 256, 384, and 512 bits, respectively, where SHA-224 and SHA-384 are truncated versions of the SHA-256 and SHA-512 functions, respectively. This range of hash sizes considerably increases security compared to the output lengths of MD5 and SHA-1 [5].

In November 2007, NIST launched the SHA-3 Cryptographic Hash Algorithm Competition in order to choose a standard hash function that met new efficiency and safety requirements. The announcement of the winning function of this competition was made public in October 2012, and the choice fell on Keccak, which was subsequently employed as the kernel of the SHA-3 family of functions [8]. The 256 and 512 versions of Keccak were adopted as the hash function in the Ethereum blockchain.

3. Elliptic Curves

In this section, the reader is presented with the mathematical description of elliptic curves, so it is possible to completely understand how the elliptic curve signature algorithm employed in blockchain works.

3.1. Definition

From a mathematical perspective, an elliptic curve defined over a field $\mathbb{F}$ is a cubic, non-singular curve of genus one with at least one rational point. Most elliptic curves can be described as the set of points $(x, y) \in \mathbb{F} \times \mathbb{F}$ verifying the following equation, known as the non-homogeneous Weierstrass equation:

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6,$$

where $a_1, a_2, a_3, a_4, a_6 \in \mathbb{F}$.

An elliptic curve point is said to be singular if the partial derivatives of the curve equation are equal to zero at that point. Consequently, the curve is singular if it has at least a singular point, while it is non-singular if it does not have any singular point.

There is another way of representing the Weierstrass equation, and that is by using homogeneous variables, as expressed below [9]:

$$Y^2Z + a_1XYZ + a_3YZ^2 = X^3 + a_2X^2Z + a_4XZ^2 + a_6Z^3.$$

This alternative expression is quite useful, as it allows defining a special point, which is called the point at infinity and is typically represented as $\mathcal{O} = [0 : 1 : 0]$. The point at infinity cannot be obtained through the homogeneous form and is paramount when operating with the points of the curve. More specifically, for any point P on the curve, the point of infinity acts as the identity element in the point addition operation, guaranteeing that $P + \mathcal{O} = \mathcal{O} + P = P$. Figure 1 shows graphically how the addition of points P and Q produces point R when using an elliptic curve defined over the field of the real numbers.

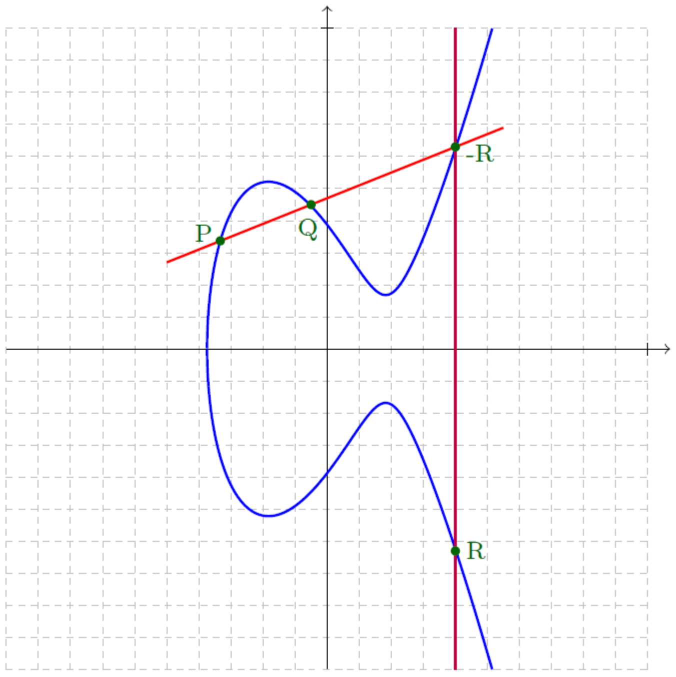

Figure 1. Point addition on an elliptic curve defined over the field $\mathbb{R}$.

3.2. Elliptic Curves over Finite Fields

Most cryptosystems defined over elliptic curves use two special finite fields: prime fields $\mathbb{F}_p$, where p is an odd prime number, and binary fields $\mathbb{F}_{2^m}$, where m can be any positive integer. Due to several reasons, including licence issues and some security weaknesses detected in curves over binary fields, prime fields are nowadays the best option when implementing elliptic curve cryptosystems [10,11]. In this type of curves, the term key length, which is employed as a first approximation for classifying the cryptographic strength of the curves, must be interpreted as the number of bits needed to represent the prime number p.

An important concept that arises in finite fields is that of the order, which can be applied to both an elliptic curve and to its points: the order of an elliptic curve is the the number of points of the curve, while the order of a point P is the value n such that $n \cdot P = \mathcal{O}$, where $n \cdot P$ is the scalar multiplication of the point P by the number n (i.e., $P + P + \cdots + P$, where P appears n times).

A point G is said to be a generator if it is used for producing either all the points of the additive group defined by the elliptic curve or a subgroup of it. For security reasons, only generators whose order is a prime number are used in commercial deployments.

Given a curve and a generator, the term cofactor refers to the result of dividing the number of points of the curve by the order of the generator. Most standards only allow curves whose cofactor is either 1 or a small number, e.g., 2, 3, or 4.

Weierstrass Curves

The peculiarities of prime fields allow simplifying the non-homogeneous Weierstrass equation, obtaining in the process what is called the short Weierstrass form represented as $y^2 = x^3 + ax + b$, where $4a^3 + 27b^2 \not\equiv 0 \pmod{p}$.

As in the case of the general Weierstrass equation, the identity element of the short Weierstrass form is the point at infinity $\mathcal{O}$, while the opposite element of a point $P = (x_P, y_P)$ is the point $-P = (x_P, -y_P)$. Adding two points $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$ such that $P \neq \pm Q$ produces a point $R = (x_R, y_R)$ whose coordinates can be computed as follows [12]:

$$\left.\begin{array}{rcl} x_R & = & \lambda^2 - x_P - x_Q, \\ y_R & = & \lambda\,(x_P - x_R) - y_P, \\ \lambda & = & \dfrac{y_Q - y_P}{x_Q - x_P}. \end{array}\right\}$$

In comparison, when $P = Q$, it is necessary to use and alternative addition formula, thus, in this case, the point $R = 2P$ obtained through the doubling operation has the following coordinates [12]:

$$\left.\begin{array}{rcl} x_R & = & \lambda^2 - 2\,x_P, \\ y_R & = & \lambda\,(x_P - x_R) - y_P, \\ \lambda & = & \dfrac{3\,x_P^2 + a}{2\,y_P}. \end{array}\right\}$$

4. Digital Signatures

4.1. Definition

The digital signature of a document is a procedure that ensures its authorship, similar to how the handwritten signature can guarantee it, but in electronic format. The digital signature was born due to the need to guarantee that the messages or documents received by electronic means effectively came from their legitimate authors and thus avoid possible impersonations.

Nowadays, any user with a digital certificate can digitally sign documents and assure their authorship. This fact has meant that, today, the electronic signature is probably the most used cryptographic protocol [1,2]. In addition, most countries consider digital signature with the same legal validity as the handwritten signature.

To be useful, digital signatures must be easy to calculate and difficult to falsify. In comparison with handwritten signatures, which are almost always the same (with only small graphic differences) whatever the document being signed, the content of digital signatures depends on the signer, the digital document to be signed, and, if the procedure used allows it, on the time at which the action was performed.

Apart from these characteristics, the elaboration of the digital signature and its subsequent verification are executed by means of cryptographic protocols that allow not only proving the authorship of a document but, together with the hash function used, guaranteeing its integrity, i.e., the signed document that is received in a transmission has not been modified during the delivery process.

4.2. Digital Signature Protocol

An asymmetric cryptosystem is a cryptographic procedure consisting of an encryption function, $\mathcal{E}$, and a decryption function, $\mathcal{D}$, so that users can encrypt and decrypt certain information. To carry out these procedures, each user has a public–private key pair. The public key (which is publicly known) allows anyone to encrypt the confidential information for the user that will receive it, while his private key (a secret value only known by the user who possesses it) is what allows the receiver to decrypt the encrypted information.

To represent the phases of any digital signature protocol, let us see how Alice, $\mathcal{A}$, prepares her digital signature for a plain-text message, m, being her public and private keys, A and a, respectively.

1. Alice must make available her public key, A, so anybody willing to verify her signatures can do so.
2. After selecting a hash function $\mathfrak{h}$, she must compute the digest of the document to be signed, m, using the $\mathfrak{h}$ function. The digest is represented as $\mathfrak{h}(m) = \widehat{m}$.
3. Alice must use her private key a in conjunction with the encryption function, $\mathcal{E}$, to the message hash, obtaining the signature for that message s: $\mathcal{E}_a\left(\widehat{m}\right) = s$.
4. Finally, Alice must post the message whose signature has just been calculated together with its corresponding signature: (m, s).

This protocol guarantees that Alice has been the signer of the message since she is the only person who knows her private key. If user Bob, $\mathcal{B}$, wants to verify the validity of the signature of $\mathcal{A}$, s, for the message m, he must follow the following signature verification protocol:

1. Calculate the document hash using the same hashing function that Alice used in order to obtain $\mathfrak{h}(m) = \widehat{m}$.
2. Decipher with the public key of $\mathcal{A}$, A, the signature of the message, s, obtaining:
$$\mathcal{D}_A(s) = \mathcal{D}_A\left(\mathcal{E}_a\left(\widehat{m}\right)\right) = \widehat{m}'.$$
3. Finally, check if the values of $\widehat{m}$ and $\widehat{m}'$ match. If so, the signature is verified. Otherwise, the signature for that message is rejected.

The previous protocol also ensures that the message was not manipulated during transmission because, otherwise, the value $\widehat{m}'$ would not match that of $\widehat{m}$.

Currently, the most widespread electronic signatures are carried out either by means of the asymmetric cryptosystem RSA [13] or by implementations using elliptic curves [14].

4.3. Elliptic Curve Digital Signature Algorithm (ECDSA)

In blockchain technology, the most widely used signature schemes are based on elliptic curves. In this sense, the Elliptic Curve Digital Signature Algorithm (ECDSA) has become a standard [15,16].

The procedure for Alice to perform a digital signature with ECDSA is the following:

1. Alice must compute the hash of the message m in the usual way, $\mathfrak{h}(m) = \widehat{m}$.

2. She must generate a random number k, $1 \leq k \leq n-1$ (where n is the order of the elliptic curve) and compute the point $k \cdot G = (x_1, y_1)$ and the value $r = x_1 \pmod{n}$. If $r = 0$, then Alice must discard those elements and repeat this step.
3. Alice must determine $k^{-1} \pmod{n}$ and calculate $s = k^{-1}(\widehat{m} + u \cdot r) \pmod{n}$.
4. The signature associated to the message m is the pair (r, s).

Once Bob has obtained the message m, the signature (r, s), and Alice's public key A, he must follow these steps if he wants to validate the signature:

1. Bob must compute by himself the hash $\mathfrak{h}(m)$ associated to the message m.
2. Then, he must check that the values r and s belong to the range $[1, n-1]$.
3. Bob must compute the value $s^{-1} \pmod{n}$ so he can calculate $z_1 = \widehat{m} \cdot s^{-1} \pmod{n}$ and $z_2 = r \cdot s^{-1} \pmod{n}$.
4. After that, he must determine the point of the curve $z_1 \cdot G + z_2 \cdot A = (x_0, y_0)$, where G is the generator of the curve.
5. Finally, Bob will accept the signature as valid if and only if $r = x_0 \pmod{n}$.

The validation process is formally correct given that, if we denote as $(P)_x$ the first coordinate of the point $P \in E$, which can be checked by the following:

$$\begin{aligned} x_0 &= (z_1 \cdot G + z_2 \cdot A)_x = \left(\widehat{m} \cdot s^{-1} \cdot G + r \cdot s^{-1} \cdot A\right)_x = \left(\widehat{m} \cdot s^{-1} \cdot G + r \cdot s^{-1} \cdot a \cdot G\right)_x \\ &= \left((\widehat{m} + r \cdot a)\, s^{-1} G\right)_x = (k \cdot G)_x = x_1 \pmod{n} = r. \end{aligned}$$

In the particular case of bitcoin, the elliptic curve that is used is known as the Koblitz curve `secp256k1`, which is defined in the Standards for Efficient Cryptography [17] and whose equation is $y^2 = x^3 + 7$. This curve has

$$115,792,089,237,316,195,423,570,985,008,687,907,852,837,564,279,074,904,382,605,163,141,518,161,494,337$$

points, considering as the base field the one with

$$115,792,089,237,316,195,423,570,985,008,687,907,853,269,984,665,640,564,039,457,584,007,908,834,671,663$$

number of elements, which is a 256-bit integer representing the key size.

In the particular case of the keys used with the `secp256k1` curve, for a user to generate his public–private key pair, he must randomly generate a collection of 256 bits and, following the protocol defined for the elliptic curves, derive the corresponding public key, which, in this case, has a length of 64 bytes [15,16].

The public key is a point belonging to the `secp256k1` curve, and thus it has two coordinates and can be represented as (x, y). Each of these numbers is encoded using the 256-bit big endian format, and the key is written down concatenating its value in hexadecimal format with the prefix `0x04`. As a result, the key is encoded with 65 bytes.

If the so-called compressed format is used to represent the two coordinates of the point that constitute the public key, then only the x coordinate is encoded, since the y coordinate can only take two values, as the curve is symmetrical with respect to the abscissa axis. In this case, the public key would be written with the 32 bytes of the x coordinate together with a one-byte prefix (`0x02` or `0x03`, depending on the result of some computations using that coordinate). As a result, when using the compressed format, the public key is encoded with only 33 bytes.

4.4. Other Digital Signatures

It should be noted that there are several types of signatures, and each of them has its corresponding protocol, specifically designed to consider certain particular situations not included in the general and standard approach of a classic digital signature [18]. In the next paragraphs, some of these types of signatures, which are used in blockchain, are briefly presented.

Multisignature schemes are signature protocols in which a group of users signs a single message and the signature is only valid if all of them jointly participate in the process [19]. In the case of blockchains, several multisignatures with varying degrees of complexity are used, the main motivation of its usage being to increase the security by preventing one of the parties involved in a transaction from deceiving the rest of parties.

Group signatures are signatures in which a single member of a certain group anonymously signs a message on behalf of all the members of the group [20]. When the signature is to be verified, the signature is deemed valid if it is proven that it was made by one of the group members, even if it cannot be determined specifically which party generated it. For example, this situation may occur when an employee of a large company signs a document; for a verifier, it is sufficient to know that the message was signed by an employee of that company, even if the verifier does not get to know which employee completed the process. In group signatures, there is a prominent member of the group of signers who acts as the administrator. In addition to including or excluding members of the group, the administrator's goal is to reveal the actual member who signed a certain document in the event that there is a dispute or controversy about the signature (for instance, if it is necessary to provide that information in court).

Ring signatures [21] are digital signatures that can be produced by any member of a particular group. In this type of signature, each member has their own keys, and it is not possible to calculate, in a computationally efficient way, which of the keys of the group members was the one used to generate a given signature. Ring signatures are similar to group signatures, but they have two differences. The first is that there is no way to revoke the anonymity of an individual signature, and the second is that it is not necessary for the user group to meet any preconditions.

In [21], two applications of ring signatures are described. The first one allows leaking a secret: One of these ring signatures could be used by "a senior government official" to disclose a secret, without revealing which official was the one who signed the disclosure. Ring signatures can be used in this case because there is no way to revoke the anonymity of such a signature and, in addition, because the group can be created at any time since there are no preconditions on the group. A second application is that of deniable signatures. In this case, the sender and the recipient of a message are the only members of the group. Thus, when one of the users signs a message, the signature will be valid for the other participant, but for any other person, it will not be possible to know who the signer is. Therefore, such a signature is valid, but both members of the group can deny being the author of it in case they are forced to do so given the circumstances.

5. Hash Pointers and Merkle Trees

5.1. Hash Pointers

As we have already indicated, blockchains are chains of blocks of information where each block has an associated hash value. Those hashes are used for generating a data index. If each data block contains the hash of the block that has been previously added to the chain, a linked chain of blocks is obtained, where the hashes play the role of pointers.

In this context, it is important to define a method that allows identifying if a certain block of information has been previously included in the chain. However, the direct search for information in lists linked by hashes is a computationally complicated task, which depends on the number of blocks included in the chain. Therefore, it is necessary to use a method that stores and manages hash pointers efficiently, and Merkle trees are one of the tools that help to achieve that goal [22].

5.2. Binary Trees and Merkle Trees

Before defining Merkle trees, it is necessary to define binary trees.

A *binary tree* is a tree-shaped graph that contains a root, internal nodes, and leaves. If a descending order is considered, from the leaves to the root, each leaf has no predecessors and has only one child, each internal node has two parent nodes and a single child node, and the root node has two predecessors and no descendants.

As an example, the following binary tree has eight leaves; two levels of internal nodes with four and two nodes, respectively; and one root.

$$\begin{tikzcd}
\mathit{leaf}: & l_1 \arrow[dr] & & l_2 \arrow[dl] & & l_3 \arrow[dr] & & l_4 \arrow[dl] & & l_5 \arrow[dr] & & l_6 \arrow[dl] & & l_7 \arrow[dr] & & l_8 \arrow[dl] \\
\mathit{nodes}: & & n_9 \arrow[drr] & & & & n_{10} \arrow[dll] & & & & n_{11} \arrow[drr] & & & & n_{12} \arrow[dll] & \\
\mathit{nodes}: & & & & n_{13} \arrow[drrrr] & & & & & & & & n_{14} \arrow[dllll] & & & \\
\mathit{root}: & & & & & & & & r_{15} & & & & & & &
\end{tikzcd}$$

Merkle trees are binary trees in which the leaves contain arbitrary data and the remaining nodes represent the output of a hash function, $\mathfrak{h}$, applied to the concatenation of the two preceding nodes.

As an example, the leaves of the following Merkle tree contain the characters that represent different currencies: euro (€), dollar ($), yen (¥) and bitcoin (₿). For their part, the nodes are determined by calculating the output of the $\mathfrak{h}$ function applied to the two previous nodes.

$$\begin{tikzcd}
l_1 = € \arrow[dr] & & l_2 = \$ \arrow[dl] & & l_3 = ¥ \arrow[dr] & & l_4 = ₿ \arrow[dl] \\
& n_{1,2} = \mathfrak{h}\,(l_1||l_2) \arrow[drr] & & & & n_{3,4} = \mathfrak{h}\,(l_3||l_4) \arrow[dll] & \\
& & & r_{1,4} = \mathfrak{h}\,(n_{1,2}||n_{3,4}) & & &
\end{tikzcd}$$

In this example, the value of the root, $r_{1,4} = \mathfrak{h}\,(n_{1,2}||n_{3,4})$ allows checking if certain data blocks were included in the tree represented by that root.

To verify if such a Merkle tree contains a certain value stored in one of its sheets, for example the bitcoin currency symbol, ₿, it is not necessary to know all the blocks of the tree, but a much smaller amount (approximately the logarithm of the number of leaves).

In fact, to prove that the value ₿ was included in the Merkle tree with root $r_{1,4}$, it is only necessary to know the value of the leaf $l_3 = ¥$ and the value of the node $n_{1,2}$ since the node $n_{3,4}$ is computed taking as input the l_3 and l_4 elements. In the event that the calculated value did not match the root of the Merkle tree, $r_{1,4}$, it can be stated that the value ₿ is not part of the tree. This statement follows directly from the bit dependency property of hash functions (see Section 2).

Thus, it is possible to manage the integrity of the information in a blockchain if the value of the root of the Merkle tree that is associated with such data is included in the data blocks of the chain. It is not necessary, therefore, to add the values of all the data, which reduces the amount of information to be included.

6. The Bitcoin Cryptocurrency

6.1. Digicash

Bitcoin is certainly one of the most well-known cryptocurrencies. However, it should be briefly noted that bitcoin is not the first digital currency that has been proposed. In fact, that honor goes to Chaum, who proposed *Digicast*, a digital currency that used cryptographic tools to carry out economic transactions anonymously [23].

The cryptographic primitives used by Chaum laid the groundwork for the creation of pseudonyms in order to simulate the non-traceability of money. In other words, if we cannot deduce who the owners of any of our real-world bills are, it is logical to demand the same property from electronic or digital currencies.

Digicash was a currency that allowed the money to be transferred blindly, that is, without knowing who the donor of the money was, once it had been received (as in real-world financial operations). However, the lack of success of this currency was due to the fact that it needed the collaboration of a bank as a trusted third party to avoid the problem of double-spending (i.e., to prevent the same digital currency from being used by its owner to pay several times).

6.2. Bitcoin, ₿

It was not until 2008, when Satoshi Nakamoto (a pseudonym used by an unknown author) proposed the bitcoin cryptocurrency, ₿. He did it by publishing its proposal in a mailing list [24], thus avoiding control of the scientific and financial community.

The technology on which bitcoin is based allows the construction of a financial system which stands as an alternative to the traditional money model, replacing the role of the central banking authorities with a consensus protocol between multiple entities (the nodes of a network). The identity protection of the owner of the electronic currency is achieved through cryptography, which is why this new type of currency is called *cryptocurrency*.

In any case, we must not forget that the non-traceability of a cryptocurrency is a problem when examined under the lens of the laws against money laundering. In the real world, the exchange of money in electronic transfers takes place between the owners of bank accounts, which are completely identified by financial institutions. Contrary to this model, in payments with cryptocurrencies, the transfer of value flows from a pseudo-anonymous identity to another pseudo-anonymous identity.

Bitcoin proposes a protocol that allows monetary transactions between users using fiat money based on a limited resource. In the case of bitcoin, the limited resource is the resolution of a computationally difficult mathematical problem. In other words, with bitcoin technology, cryptocurrencies are minted by software computing a hard-to-solve problem, which has a high energy expenditure associated.

Indeed, the protocol associated to bitcoin works through a Peer-to-Peer (P2P) network, so that all nodes can access a copy of the bitcoin blockchain, but only certain users can write on it.

Each data block contains a header and a set of transactions. More specifically, the header contains the version of the bitcoin system, the hash value of the previous block, the root of the Merkle tree that contains all the transactions of the block, the time stamp with the date of creation of the block, and two other parameters: the difficulty and a nonce (a number used only once). These two parameters play a very important role in the bitcoin mining process.

By its own design, the time elapsed between the inclusion of two consecutive blocks in the bitcoin blockchain should be about 10 min. This conservative criterion aims to guarantee the propagation of the new block to the entire bitcoin P2P network. To guarantee this rate of generation of new blocks, every 2016 blocks the difficulty parameter that is included in the header of the blocks is reset according to the formula

$$D_n = \frac{D_o \cdot 2016 \cdot 10}{T},$$

where D_n represents the new difficulty, D_o the old difficulty, and T the time spent mining the last 2016 blocks.

Miners are responsible for solving the computationally difficult problem, which involves finding partial collisions of a given hash function. In Section 2, these functions are defined and it is noted that a collision consists of finding two different messages m_1 and m_2, $m_1 \neq m_2$, such that $\mathfrak{h}(m_1) = \mathfrak{h}(m_2)$. In the case of bitcoin, the selected hash function is SHA-256 [5].

To find a partial collision, each miner considers the values that form a block (header and transactions), then uses the difficulty corresponding to the moment in which he makes his calculations, and tries different nonces until the hash value obtained is less than a given threshold. This threshold value is equivalent to the hash value having a certain number of leading zeros. Such number of zeros is determined by the number of miners in the network (and their computing power) so that the time needed to find a solution is, on average, about 10 min. It is clear that the higher is the number of leading zeros, the greater is the difficulty of solving the problem.

This mathematical problem of finding partial collisions, that is, the PoW (Proof of Work) (sometimes known as a mathematical puzzle) guarantees that it is not possible to make a *double expense*. In other words, when a user pays an amount of cash in exchange for goods, he loses control of that money, which did not happen in the case of Digicash. In this case, the PoW guarantees that once a payment with bitcoins is made, the amount of cryptocurrency used is no longer available to the user who has paid with it, so that it cannot be reused.

The PoW concept was originally proposed by Dwork and Naor in 1992 [25], although it was not associated to that expression back then. Their proposal described a process to reduce the reception of spam and Distributed Denial-of-Service (DDoS) attacks. The process implied that senders of emails had to complete a computational work with little electrical cost, so that the cost was high if the computation had to be done millions of times, which would not compensate its possible benefit to spam creators. Subsequently, Jakobsson [26] named this procedure as Proof of Work.

As a simple example, if the content of the block was the string "ITEFI-CSIC", the miner should try, concatenating the previous block, different values until he achieves the goal of obtaining a predetermined number of leading zeros. The only way to find a good nonce is to try it randomly until finding one that meets the requirements.

```
     ITEFI-CSIC46: 007f55f8bdd1f4969501b7fba49c23ae4880e28726a832ce001ffd7f99eb5c38,
   ITEFI-CSIC2521: 00061ec240f84662314ec825f84c045d12ee2131f2d5f0a7612e3330c942ea70,
  ITEFI-CSIC39266: 000089847656f7abffa7d83f7d93da6da6aa9219ee886306eda19e0ad0afc4f8,
ITEFI-CSIC1014613: 00000c45edbbf730197f8a165f7edc18806e71693da854bb10d978ad951452e1.
```

The following example corresponds to bitcoin's block #545,919, extracted at 20:39:15 on 15 October 2018:

```
000000000000000000d04b0dd03e95f4c0dbb811f778a31731834e662ce4169.
```

Once a miner makes its solution public, the rest of the network can verify that the appropriate parameters have been used and, if there is consensus, the block processed by the miner will be the one that joins the blockchain. The miner will receive his reward in the form of bitcoin currency.

One of the fundamental aspects of this technology is the guarantee of pseudo-anonymity among the participants when making a transaction. This is achieved through the strategy that each participant in the network is identified by a bitcoin account.

To create a bitcoin account, each user needs a public–private key pair for the ECDSA protocol (see Section 4.3). Taking as input the 512 bits of the public key, the user determines its hash value using the SHA-256 function, and then the RIPEMD hash function is applied to the result, thus the output is a 160-bit string. This final result is encoded in Base-58 to convert it into alphanumeric values, which is the personal address of the user corresponding to such a key pair. It should be taken into account that each user can have as many addresses as desired.

Through this scheme, each user is pseudo-anonymously identified by an address associated to a public key. If the user is the recipient of a transaction, the amount that is transferred will increase the balance that the user already had in the wallet associated to that address. On the contrary, if the user is the originator of the transaction, he must prove that he is the owner of the amount involved in it using the private key associated to the public key of the address being used. Once the transaction is completed and the block containing it is added to the blockchain, the recipient can reuse the money received in a new transaction, proving that he is the legitimate owner using his private key.

In this way, the private key constitutes proof of ownership of a specific address and, therefore, of the origin and destination of the transactions. In short, the money linked to an address constitutes the balance in the transactions that have that address as origin or destination.

Finally, in relation to the distributed consensus protocol for the acceptance and inclusion of new blocks in the blockchain, since there is no central node, there must be a coordination between the nodes so that the update of the financial balance created after new transactions is performed consistently.

Consensus systems based on the model introduced by Lamport et al. [27] are known as Byzantine Fault Tolerance (BFT) systems by analogy with the *problem of the Byzantine generals*. In this problem, three or more generals are besieging an enemy city and must agree if they attack or withdraw, taking into account that, if they do not reach an agreement, they will be defeated. Generals communicate through messengers, knowing that messages can be intercepted by the enemy, manipulated by messengers, or include false information intentionally entered by traitorous generals. In [27], it is proven that it is possible to reach a consensus provided there are no more than a third of traitorous generals.

In fact, the consensus proposed by Nakamoto in a network with a high synchronization rate between nodes and where no node achieves a computing capacity close to 50 % of the global, gives rise, in practice, to a system that presents a Nash equilibrium. It is important to recollect that a Nash equilibrium occurs when each player uses the best strategy to win and everyone knows each other's strategies. Therefore, each player earns nothing by modifying their strategy while the others maintain theirs.

7. Conclusions

Throughout this work, the main cryptographic tools related to the security and reliability of blockchain are presented: hash functions, digital signatures, elliptic curves, and Merkle trees.

Digital signatures allow proving the authorship of a document and guaranteeing its integrity. The digital signature algorithm used by many blockchain technologies is ECDSA, a standard that uses elliptic curves. Among all the curves that can be used, `secp256k1` has been selected by many implementations. `secp256k1` is a standard elliptic curve that uses 256-bit keys, a strength level similar to the AES algorithm.

Hash functions appear in several places when dealing with blockchains. On the one hand, they are used as part of the digital signature algorithms. On the other hand, hash functions play an important role in the implementation of Merkle trees, a concept that allows efficiently checking if a certain block is present in a blockchain.

Through the combination of the aforementioned elements, it has been possible to create blockchain, a reliable technology with a promising future.

Author Contributions: The contribution of all authors of this paper was very similar, as all of them have collaborated in the elaboration, validation, reviewing edition, and the location of the used resources. Finally, V.G.M. and L.H.E. are responsible for the funding acquisition. All authors have read and agreed to the published version of the manuscript

Funding: This work was supported in part by the Ministerio de Economía, Industria y Competitividad (MINECO), in part by the Agencia Estatal de Investigación (AEI), in part by the Fondo Europeo de Desarrollo Regional (FEDER, UE) under Project COPCIS, Grant TIN2017-84844-C2-1-R, and in part by the Comunidad de Madrid (Spain) under Project reference P2018/TCS-4566-CM (CYNAMON), also cofunded by European Union FEDER funds. Víctor Gayoso Martínez would like to thank CSIC Project CASP2/201850E114 for its support.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

ANSI	American National Standards Institute
BFT	Byzantine Fault Tolerance
DDoS	Distributed Denial-of-Service
ECC	Elliptic Curve Cryptography
ECDSA	Elliptic Curve Digital Signature Algorithm
MD	Message Digest
NIST	National Institute of Standard and Technology
P2P	Peer-to-Peer
PoW	Proof of Work
RIPEMD	RACE Integrity Primitives Evaluation Message Digest
RSA	Rivest, Shamir, and Adleman
SECG	Standards for Efficient Cryptography Group
SHA	Secure Hash Function
TOWF	Trapdoor One-Way Function

References

1. Menezes, A.J.; van Oorschot, P.C.; Vanstone, S.A. *Handbook of Applied Cryptography*; CRC Press, Inc.: Boca Raton, FL, USA, 1996. [CrossRef]
2. Paar, C.; Pelzl, J. *Understanding Cryptography. A Textbook for Students and Practitioners*; Springer: Heidelberg, Germany, 2010. [CrossRef]
3. Rivest, R. The MD5 message-digest algorithm. *RFC* **1992**, 1321. [CrossRef]
4. Wang, X.; Yu, H. How to break MD5 and other hash functions. In *Advances in Cryptology-Eurocrypt'2005*; Cramer, R., Ed.; Springer: Berlin, Germany, 2005; pp. 19–35. [CrossRef]
5. NIST. Secure Hash Standard (SHS). Federal Information Processing Standard Publication. FIPS 180-4. 2015. Available online: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf (accessed on 15 January 2020).
6. Wang, X.; Yin, Y.L.; Yu, H. Finding collisions in the full SHA-1. In *Advances in Cryptology-Crypto'2005*; Shoup, V., Ed.; Springer, Berlin, Germany, 2005; pp. 17–36. [CrossRef]
7. Dobbertin, H.; Bosselaers, A.; Preneel, B. RIPEMD-160: A strengthened version of RIPEMD. In *Fast Software Encryption*; Gollmann, D.; Ed.; Springer: Berlin, Germany, 1996; pp. 71–82. [CrossRef]
8. NIST. SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. FIPS PUB 202. 2015. Available online: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf (accessed on 15 January 2020).
9. Menezes, A.J. *Handbook of Elliptic and Hyperelliptic Curve Cryptography*; Chapman & Hall/CRC: Boca Raton, FL, USA, 2006.
10. Bernstein, D.J., Lange, T. Curve25519: New Diffie-Hellman speed records. In Proceedings of the 9th International Conference on Theory and Practice in Public-Key Cryptography (PKC 2006), New York, NY, USA, 24–26 April 2006; pp. 207–228.
11. Lochter, M., Merkle, J. Elliptic Curve Cryptography (ECC) Brainpool standard curves and curve generation. *RFC* **2010**, 5639. [CrossRef]
12. Bernstein, D.J.; Lange, T. Explicit-Formulas Database. 2016. Available online: https://hyperelliptic.org/EFD/ (accessed on 15 January 2020).
13. Rivest, R.; Shamir, A.; Adleman, L.M. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM* **1978**, *21*, 120–126. [CrossRef]
14. Menezes, A.J. *Elliptic Curve Public Key Cryptosystems*; Kluwer Academic Publishers: Boston, MA, USA, 1993. [CrossRef]

15. ANSI. Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA), National Bureau of Standards. ANSI X9.62. 2005. Available online: https://standards.globalspec.com/std/1955141/ANSIX9.62 (accessed on 15 January 2020).
16. NIST. Elliptic Curve Digital Signature Algorithm Validation System (ECDSA2VS). Federal Information Processing Standards. FIPS 186-4. 2014. Available online: https://csrc.nist.gov/Presentations/2014/The-FIPS-186-4-Elliptic-Curve-Digital-Signature-Al (accessed on 15 January 2020).
17. SECG. Recommended Elliptic Curve Domain Parameters. SEC 2 Version 2.0. 2010. Available online: https://www.secg.org/sec2-v2.pdf (accessed on 15 January 2020).
18. Arroyo, D.; Diaz, J.; Rodríguez, F.B. Non-conventional Digital Signatures and Their Implementations-A Review. In *Advances in Intelligent Systems and Computing, Proceedings of the International Joint Conference: CISIS'15 and ICEUTE'15*; Herrero, A., Baruque, B., Sedano, J., Quintián, H., Corchado, E.; Eds.; Springer: Heidelberg, Germany, 2015; Volume 369, pp. 425–435. [CrossRef]
19. Itakura, K.; Nakamura, K. A public-key cryptosystem suitable for digital multisignatures. *NEC Res. Dev.* **1983**, *71*, 1–8. Available online: https://scinapse.io/papers/200023587 (accessed on 15 January 2020).
20. Chaum, D.; Van Heyst, E. Group signatures. In *Advances in Cryptology-Eurocrypt'91*; Davies, D.W., Ed.; Springer, Berlin, Germany, 1991; pp. 257–265. [CrossRef]
21. Rivest, R.; Shamir, A.; Tauman, Y. How to leak a secret. In *Advances in Cryptology-Asiacrypt'2001*; Boyd, C., Ed.; Springer: Berlin, Germany, 2001; pp. 552–565. [CrossRef]
22. Merkle, R.C. Method of Providing Digital Signatures. U.S. Patent 4,309,569, 5 January 1982. Available online: https://patents.google.com/patent/US4309569A/en (accessed on 15 January 2020).
23. Chaum, D. Blind signatures for untraceable payments. In *Advances in Cryptology-Proceedings of Crypto'82*; Chaum, D., Rivest, R.L., Sherman, A.T., Eds.; Springer: Boston, MA, USA, 1983; pp. 199–203. [CrossRef]
24. Nakamoto, S. Bitcoin: A Peer-to-Peer Electronic Cash System. Available online: https://bitcoin.org/bitcoin.pdf (accessed on 15 January 2020).
25. Dwork, C.; Naor, M. Pricing via Processing or Combatting Junk Mail. In *Advances in Cryptology-Proceedings of Crypto'92*; Brickell, E.F., Ed.; Springer: Berlin, Germany, 1993; pp. 139–147. [CrossRef]
26. Jakobsson, M.; Juels, A. Proofs of Work and Bread Pudding Protocols (Extended Abstract). In *Secure Information Networks*; Preneel, B., Ed.; Springer: Boston, MA, USA, 1999; pp. 258–272. [CrossRef]
27. Lamport, L.; Shostak, R.; Pease, M. The Byzantine Generals Problem. *ACM Trans. Program. Lang. Syst.* **1982**, *4*, 382–401. [CrossRef]