Universitat Rovira i Virgili

BITCOIN: A SOURCE OF FAIRNESS

Javier Torner López

FINAL MASTER THESIS

Directed by Dr. Oriol Farràs Ventura

MASTER IN
COMPUTATIONAL ENGINEERING
AND MATHEMATICS

Madrid

2017

FICHA DEL TRABAJO FINAL de MÁSTER

Título del trabajo:	Bitcoin: A Source of Fairness
Nombre del autor:	Javier Torner López
Nombre del director:	Oriol Farràs Ventura
Fecha de entrega (mm/aaaa):	06/2017
Área del Trabajo Final de Máster:	Criptografía
Titulación:	*Máster en Ingeniería Computacional y Matemáticas*

Resumen del Trabajo (máximo 250 palabras):

Bitcoin es la primera moneda digital descentralizada, lo que supone que no depende de ninguna autoridad central (como en el caso de los bancos tradicionales). Su creación en 2009 supuso un avance rompedor en la tecnología existente, que hasta entonces no había resuelto el problema de las transacciones monetarias en un entorno sin autoridad. Bitcoin resolvió el problema del doble gasto mediante un ingenioso sistema de registro común actualizado mediante elección en base a pruebas de trabajo basadas en capacidad de cómputo.

El presente trabajo desarrolla en primer lugar una descripción del funcionamiento interno de Bitcoin, para dar en una segunda parte un tratamiento a su aplicación con fines no monetarios, como otras aplicaciones de la tecnología Blockchain en que se basa, centrándose principalmente en su uso para implementar aplicaciones miltiparte de juego limpio, como loterías y cálculos distribuidos con entradas secretas, en las que se garantice un desarrollo independiente de confianza mutua entre las partes.

Abstract (in English, 250 words or less):

Bitcoin emerged as the first decentralised digital currency system, which means it is not dependent on any central authority (as in the case of conventional banking). Its creation in 2009 constituted a breaking progress in existing current technology, which up to then had been unable to solve the problem of monetary transactions in an authority-less environment. Bitcoin addressed the problem of the double-spending by means of a brilliant distributed database system which could be updated only via an election based on proofs of computational work.

The present essay develops in a first part a detailed description of the inner workings of Bitcoin, covering in the second part several some non-financial applications, like those based on the Bockchain technology, and focusing mainly on its use for implementing Fair Multiparty Computations, like lotteries and distributed computation with secret inputs, which guarantee a fair development in an environment lacking of any mutual trust between the parties involved.

Palabras clave (entre 4 y 8):

Bitcoin, Multiparty Computation, Fairness, Cryptocurrency, Decentralisation, Peer-To-Peer, Proof of Work, Cryptography.

Table of Contents

0. Introduction

0.1. Notes on Methodology

This essay was conceived as the Final Master Thesis leading to the Master Degree in Computational Engineering and Mathematics, offered by University Rovira i Virgili and Open University of Catalonia, course plan 2016-2017.

The working methodology which conducted to the present dissertation was mostly based on online supervision and direction by the thesis Director, Dr. O. Farràs Ventura, together with dedicated research by the author, J. Torner López.

Electronic mail and online document sharing services were the selected technologies for communication and work, agreed upon by the Director and the student consensus. Topics picked by the thesis Director where researched and digested by the author, benefitting the later from the Director’s assistance and guidance during the whole process. Current edge innovations and state of the art topics, covered in the final chapter, where proposed by the Director and elaborated based on multiple papers and other sources, while the former parts of the text, which may be considered as introductory topics, describe the Bitcoin system and are all well documented and covered in numerous generalist sources.

While writing this thesis, time emerged as the most limiting factor for proper work. The author thus would like to express his gratitude to Dr. Farràs Ventura for his ability to cope with time constraints and for his advice throughout the course.

0.2. History

Published in late 2008 by an anonymous author, Bitcoin features a previously unknown technological development capable of giving support to a digital currency as well as all of its required facilities, such as secure transactions, issuance of new currency and (most importantly) all of it being an authority-independent, peer-to-peer system. Bitcoin refers to the monetary unit of this system (BTC), as well as to the network and the protocol it is based on.

The design of Bitcoin relies on two fundamental concepts: a protocol for multiparty consensus reaching, and an incentivation policy providing benefits to those peers acting as transactions certifiers. The whole system security rests on well-known cryptographic and computational premises, being Bitcoin in fact a complex engineering design and network setup. Altogether, its inherent features provide support for secure monetary transactions within the Bitcoin network.

Since its conception, the number of parties involved in or accepting Bitcoin as a payment platform has rapidly increased, this growth being accompanied by a wider network population and a higher market value (currently one monetary unit outreaching the value of an ounce of gold at the time of writing). This quick rising popularity fate has attracted several unrequested side partners which use the network for dark economic activities or speculative investments.

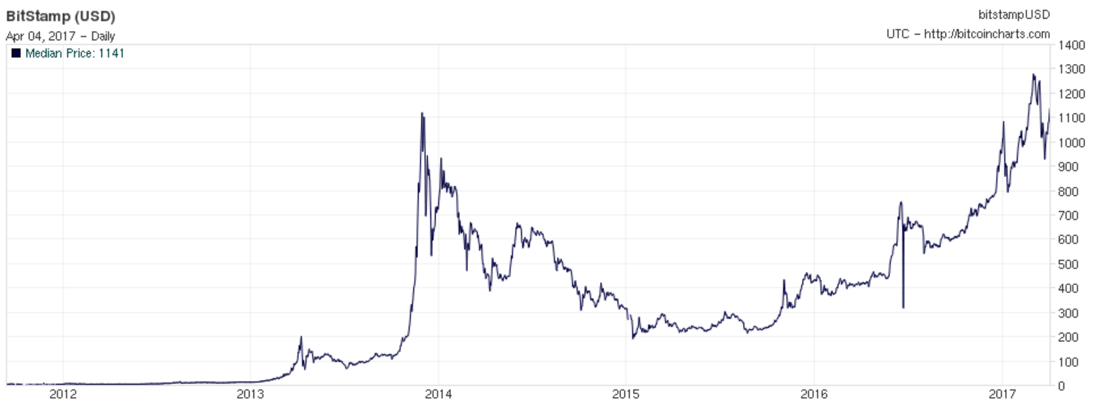


Figure 1: Bitcoin vs. US Dollar value, within time frame of existence (reproduced under consent of BitcoinCharts.com)

While its obscure nature and lack of central governance has recently led to attempts from some governments to legislate over it, Bitcoin may be classified as uncontrollable by design. Anyone may easily and anonymously set up new addresses which are ready to receive or send transactions, and recent advances have complicated matters in terms of money traceability.

Whether or not Bitcoin will continue enjoying wide adoption and user support, it falls apart from doubt that it has set a milestone in the way of current technical progress, settling a firm ground in economic fields were cryptography itself was stalled and could not go much further.

0.3. Technological breakthrough

While several other schemes supporting digital currency transactions already existed before Bitcoin, all of them lacked the key feature of *independence* from a *central authority* (i.e., a central bank). The authority role of verifying (signing) monetary transactions is in fact a workaround for the double-spending problem, this is, guaranteeing that any amount of money may only be spent once by its current bearer. The same holds for currency issue, an attribution reserved to the authority due to the inherent issues arising from the digital nature of the currency.

The way in which Bitcoin solves the double spending problem in an authority-independent fashion is by relying on a decentralized distributed database, namely the *Blockchain*, which serves as a record of all previous transactions from the time of its inception to date. This database is publicly available and ensures not a single coin fraction is spent twice. Transactions are digitally signed by the parties involved (hence proving monetary ownership), and later appended to the ledger by specific network peers known as *miners*, which provide the transaction with a certifying *proof of work* (which renders double-spending unfeasible).

Hence, it is the miners who build the blockchain, which may be freely accessed by anyone via the Bitcoin network. All transactions are thus public and, as such, the Bitcoin system is not anonymous (this constitutes widespread misbelief). The addresses associated to actual people, however, are not necessarily linked to any specific identity, and may then qualify as anonymous. It is therefore worth remarking that anonymity itself is not dealt with within Bitcoin.

The proof of work is the result of a resource-demanding computation involving the transaction details, and it serves a double purpose: On one hand, it helps keeping the transaction database unique, up to date and tamper-resistant (as much as the computational complexity allows), and on the other, it does so by providing monetary rewards to miners providing such validation. Incentivation constitutes the basic mechanism for issuing new currency and keeping the ledger sanity, by rewarding newly created coins to the miners supporting the system (as well as small transaction fees whose intent is motivating a faster acceptance).

The hardness of the computational proofs of work is regularly tuned according to the entire network computational power. This leads to a stable register aggregation pace regardless of the number of existent mining peers. Blockchain records are committed in a linked manner, which the work to be proved depends on. Such a design ensures that any alteration to the existing blockchain will require a higher computational power that the entire well-behaved network mining pool may provide, meaning that the security of the network lies on itself.

Maintaining a distributed database of all valid monetary transactions while having the whole network agreeing upon it is quite a demanding endeavour. For this purpose to be fulfilled, an intricate cryptographic design extends way beyond the simplicity of the described ideas of time-consuming proofs of work and signed public transactions. The whole Bitcoin system, its limitations and how related problems are dealt with will be described in detail in the forthcoming chapters.

0.4. Derived Works

Based on the very same principles which gave birth to Bitcoin, other related works followed. Most of these include alternate rebranded coins (possibly with a richer feature set,) while many other innovative applications, mostly based on the blockchain principles and many of them still experimental, were also developed.

A variety of third party applications, ranging from distributed virtual machines to gambling and data records will be described in the final part of this document. It is the central topic of this thesis to clear the path to introducing such concepts, by building on principles like that of *fairness* in a distributed system, be it computational machinery or a simple card playing platform.

1. Cryptographic Primitives

1.1. Introduction

In order for the Bitcoin system to be reliable and secure, some elementary cryptographic tools are extensively relied upon. These constitute the basis for secure transactions and user identification, and are reduced to a small subset of well-known and extensively researched methods in the field.

The purpose of this chapter is introducing the unversed reader on such topics as public key exchange, secret sharing over a compromised channel, data signing (authorship certification) and other related topics. Two main categories of cryptographic methods are employed, as defined in the following sections:

1. **Hash functions**: These are computations taking an arbitrary-length data input and producing a fixed-size, relatively small value associated to such data in a deterministic way, requiring that no known efficient computation may find an inverse preimage from a specific hash value. Hashes in Bitcoin are used both as brief identifiers of large data fields (which cannot therefore be made up to match a specific hash), and are involved in the concept of proof of work.

2. **Asymmetric cryptography**: This is a two-key scheme cryptosystem enabling secure data signing and communication over a channel possibly affected by eavesdropping. The key pair provides a signing method for any party to easily check whether some transaction was originated by its actual author or not.

1.2. Hash functions

A hash function is a one-way application $H : \{0,1\}^* \rightarrow \{0,1\}^n$ mapping arbitrary-sized binary input blocks (generally byte-aligned) to small, fixed-sized binary numbers (i.e., a 256-bit field), such that the value H(x) can be computed efficiently (with a low computational time cost), whereas finding an inverse preimage $H^{-1}(y)$ is computationally infeasible (i.e., with an exponential cost on the size of input).

According to the above definition, hash values may be used as brief identifiers (fingerprints) for large data blocks, assuring that any alteration to the source data will produce a different hash value. In this way, a signature for a data block may be transmitted without disclosing the data itself, also cutting down on bandwidth. When later needed, inspection of the actual data allows the receiver to easily check if its hash value matches the expected value as stated before.

Hash functions used in cryptography should exhibit the following properties:

- **Determinism**: Given any input x, its associated hash value H(x) is unique.
- **Avalanche effect**: For any two non-equal inputs x, y, the probability of any bit of H(x) to match the same bit position in H(y) should be ½.

- **Pre-image resistance**: Given a hash value y, finding an input x such that H(x) = y should be computationally hard. For this property to hold, it is required that the size of the output field n is large enough to prevent brute-force attacks.
- **Collision resistance**: It should be unfeasible to find two different inputs x, y such that H(x) = H(y). In practice, this property has shown to be weaker than preimage resistance (i.e., easier to bypass), according to available research.

Common hash functions designed and long-studied by cryptographers include the MD, RIPEMD and SHA families, among them being SHA-256 and RIPEMD160 the ones used in Bitcoin. As time passes, new approaches are developed to attack currently existing and widely adopted hash functions, resulting in new standards being developed, frequently with larger hash lengths.

Byte sequence	**SHA-256 (hexadecimal)**
`"Hello, World"`	`03675ac53ff9cd1535ccc7dfcdfa2c458c5218371f418dc136f2d19ac1fbe8a5`
`"Hello, Worle"`	`d970c8cfd1f79f5a2cf4a7f3518e6c2e25c4e375136ffdf4f1fd55fe1a565bad`
??? (unknown to the author)	`03675ac53ff9cd1535ccc7dfcdfa2c458c5218371f418dc136f2d19ac1fbe8a4` (notice difference in last bit)

Table 2: Some hash examples

It should be noted that the existence of real hash functions still constitutes an open mathematical question. Particularly, proving the existence of hash functions would imply a negative answer to the P = NP problem, which seems unlikely to be solved at the time being. In practice, those hash functions for which no known serious attack procedure is known are trusted, assuming they are safe to use.

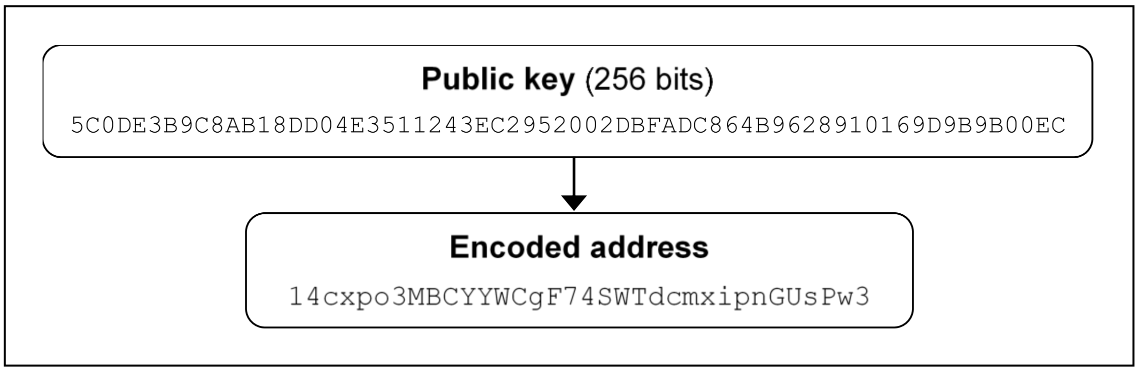
Given the complexity of the current hash standards and the many subtle reasons motivating their design, the reader is referred to the Bibliography section for the details. It is the work of several official agencies to develop, evaluate and/or document these standards for researchers, developers and hardware vendors.

1.2.1. Hashes as identifiers

Hash functions are widely used for Bitcoin internals as various kinds of identifiers. From block numbering to client addresses (taken as a hash of a public key), hashes play a big role in its design. They also provide basic checksum support for human-related data handling issues (i.e., detecting address mistyping errors).

As we will later see, amounts of bitcoin are transferred between client addresses. When it comes to encoding such addresses for identification purposes, it is the hash of the client's public key, as a compound of SHA-256 and RIPEMD160 digests which is referred to in transactions (rather than the actual key).

The users' public keys serve indeed as clients' identifiers. As seen in the next section, public keys are fixed-width 512- or 256-bit numbers. In order to convert these to user ids, a prefix is prepended to the key (0x04 in the case of full-length 512-bit keys, or 0x02/0x03 for *compressed* 256-bit keys depending on parity). The expanded key is then fed to a SHA-256 hasher and its output is in turn passed through RIPEMD160, yielding a 160-bit number. By adding to the result a 0x00 prefix and a 32-bit error detecting checksum suffix, the output is finally encoded in a base-58 humanly-readable format using alphabet “1-9A-HJ-NP-Za-km-z” (this is, numbers plus mixed-case English letters, excluding those characters prone to be misspelled, like 0, I (uppercase i), O and l (lowercase L).



Public key (256 bits)
5C0DE3B9C8AB18DD04E3511243EC2952002DBFADC864B9628910169D9B9B00EC
Encoded address
14cxpo3MBCYYWCgF74SWTdcmxipnGUsPw3

Figure 3: Public key encoded to address

Other identifiers used in Bitcoin benefitting from a similar hash-based processing include scripts, blocks and transactions. These will be covered in Chapter 2, together with the custom private key format used by wallet software.

1.2.2. Proofs of work

As stated above, hash functions are designed to be hard to invert (i.e., finding an inverse image given a specific hash value), provided that their output exhibits random properties and no remarkable relation to the input which could be used to track back their process. The hardness to find a preimage leads to most efficient attacks known being merely brute-force based (exhaustively testing all possible inputs until a match is found), or other methods with similar hardness.

The hash fundamental property of having to search the domain space in order to find an inverse image can be used to construct a simple, yet powerful tool to guarantee that a specific hard work, in terms of time computation, was performed.

Considering a reduced hash field, and requiring that only the bits in it match the full hash value, we are effectively reducing the function complexity of the problem. Furthermore, we can customise the field width in order to lower down the overall complexity accordingly. For example, cutting down the field width by one bit will lead to an actual required time for finding an inverse image of half as long as the same problem associated to the full hash (provided brute-force is used).

Therefore, requiring that an inverse hash function in computed given a specific trim-fielded hash value is a problem which can be tuned to require a specific time on average (this may be set according to overall available hashing speed capacity). In turn, once the reduced problem is solved, providing a solution for others to check it should be trivial. Such a solution is known as a *proof of work*.

Bitcoin relies on proofs of work as a method to reach a network-wide consensus on which specific transactions are to be recorded in the central distributed ledger. Mining peers on the network compile lists of transactions, group them in blocks, and successively test several variable fields on the block header in order to make it match a partial hash. This is a time-consuming task which is expected to produce one valid block every 10 minutes in the entire network (this value is adjusted based on previous observed hash production rate).

Whenever a peer solves a block, it broadcasts it to the network, so that all can check whether it is valid. As soon as a valid block is received, the mining peers move on to solving a new block with newer transactions. It is therefore very unlikely that several peers will come out with valid blocks at once, due to the hardness of the problem involved. This behaviour provides a basis for the network to agree as a whole in a unique common transaction database.

1.3. Asymmetric Cryptography

Asymmetric cryptography, also known as *public key cryptograph*y, allows for secret message sharing between two parties over an insecure channel (e.g., affected by eavesdropping) with no prior agreement on a common secret key. While symmetric methods rely on a single shared key between both parties, which is used for both encrypting and decrypting and is to be kept secret, public key cryptography features two different keys, one used for encryption (named public key) and another for decryption (private key), thus making it possible to share the former key via an insecure channel without compromising the later.

A second application of asymmetric cryptography is *digital signing* of documents, data or, as a matter of fact, transactions. In a similar fashion, a key pair allows the signee or author to produce a digital signature of any data using his private key, while third parties may later check its authorship with the associated public key. The signature cannot be tampered or reproduced for different data without the secret key, whereas the public key may be freely distributed at no risk.

Bitcoin was not designed as an anonymous system, so secret communication is not a necessity. Digital signing, however, is extensively used in transactions to certify ownership and the intention of transferring the stated amount by the issuer. As explained in the previous section, address identifiers within Bitcoin are made from hashed public keys. Having this design in mind, proving the ownership of the funds of a specific address is as simple as providing the full public key associated to the address with an accompanying valid transaction signature.

In the absence of any formal proof of the opposite, public-key cryptography is considered safe. There currently exist a vast amount of methods implementing secret communication and digital signing algorithms which are (believed to be) cryptographically secure. In all of them, the key pair is formed in a way in which the public key may be computed from the secret key in a brief amount of time, while the opposite has not been shown, and currently constitutes an intractable problem (i.e., extremely hard to solve, in terms of time complexity).

There are two popular kinds of mathematical basis for private key cryptography. The early methods were based in the *integer factorization problem*. In contrast, modern asymmetric systems rely their security on elliptic curve algebras, holding similar properties as their integer factoring counterparts (i.e., feasible computation in one direction with no known tractable inverse), while this later offering more effective key lengths and being less known attacks known. The elliptic curve theory used in Bitcoin for transaction signing will be detailed below.

1.3.1. The ECC Model

Elliptic Curve Cryptography comprises a wide family of curves defined in discrete modular domains under which point product (conceived as a recursive addition) may be efficiently computed in the derived field, while the opposite does not hold.

Let C be an elliptic curve over the domain D, and let P, Q be points of C. The addition P+Q is defined as the point R obtained by intersecting the line PQ with C, then reflecting (negating) the Y coordinate. In the case that P = Q, the process is similar, except that the tangent to the curve on P is considered. For such an R to exist and be unique, only specific curves and domains are considered.

Point addition meets two essential properties:

- It is associative, P+(Q+R) = (P+Q)+R = P+Q+R (notation).
- In integer curve domains, the line produced will intersect the curve in a point of the domain.

Under the above statements, computing nP = P+…+P for a high n is possible and may be done with little cost incurred. Indeed, since point addition is associative, n may be effectively decomposed as a sum of powers of 2, $n = 2^{i(1)} + \ldots + 2^{i(m)}$, so that $nP = 2^{i(1)}P + \ldots + 2^{i(m)}P$, with $m \leq \log_2 n$.

While computing point powers nP may be carried out in a quick fashion, it is not the case of the inverse operation, finding n given P and nP. The tangent and point-to-point lines devise an unfolding complex scenario when trying to find n. The presumed hardness of this problem, known as computing the elliptic curve discrete logarithm, is the key property founding the digital signature security.

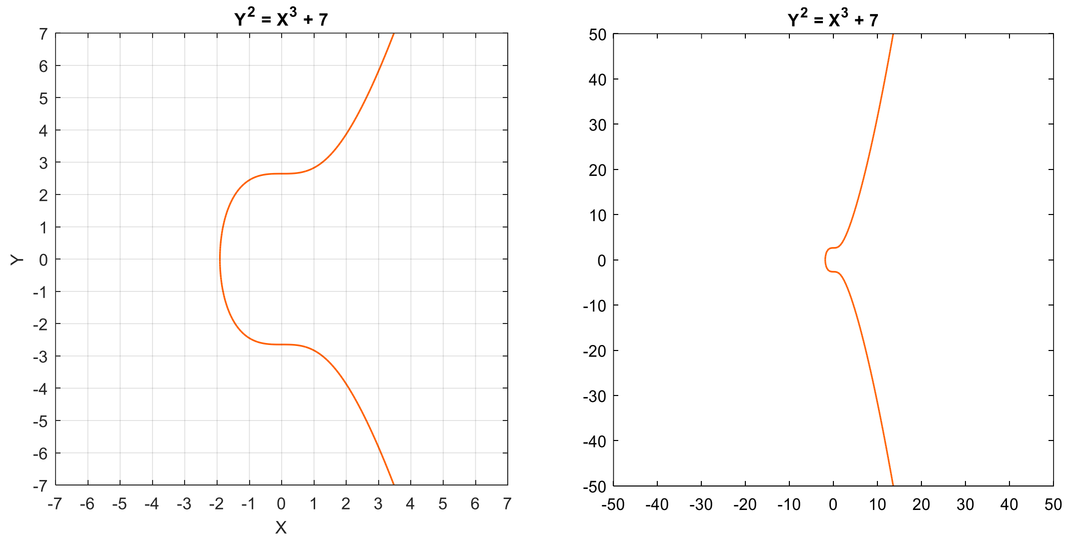


Figure 4: Algebraic curve used in Bitcoin signatures over the real domain (note an integer modular field is used in the actual implementation)

The curve, domain and starting point used in Bitcoin are as follows:

- The domain in which the curve is defined is the integer modular field Z_p, with $p = 2^{256} – 2^{32} – 977$, a very large prime number (256 bits).
- The curve equation is $C : y^2 = x^3 + 7$
- The base point in C is G = (x, y), with hexadecimal notation x = 79BE667EF 9DCBBAC55A06295CE870B07029BFCDB2DCE28D959F2815B16F817 98 and y = 483ADA7726A3C4655DA4FBFC0E1108A8FD17B448A68554 199C47D08FFB10D4B8.
- The order of G is n = FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEBAAED CE6AF48A03BBFD25E8CD0364141 (it is expected that nG = 0).

The process for creating a private-public key pair is:

1. Randomly pick an integer number k in the interval (0, n). This is the secret key. It is advised that many random numbers generators are insecure due to predictability (i.e., short runs/periods or other faults).
2. Compute the public key as K = kG (note this is a modular operation). For the sake brevity, usually only the x coordinate of K and the sign of y are denoted, since together these two values will suffice to obtain y.

Note that, while K can be derived from k easily, the opposite does not hold.

In order to sign a message m, the process is a little more complex:

1. Compute the message hash h as a 256 bit field (with SHA-256).
2. Randomly select an integer r in the interval (0, n). Note r should be different every time a hash is computed in order to prevent known plain-text attacks.
3. Obtain the x coordinate of rG modulo n. If x = 0, find a different r as in 2.
4. Compute $s = r^{-1} (h + xk)$ modulo n, where r^{-1} is the inverse of r in Z_n.
5. Transmit the message signature, (x, s).

Finally, a given signature (x, s) for message m can be checked for validity using the public key K, which encodes the point K in the curve:

1. Check that x and s are in the interval (0, n).
2. Compute the message hash h and s^{-1} modulo n.
3. Calculate $u = hs^{-1}$ and $v = xs^{-1}$, taking both modulo n.
4. The signature is valid if and only if the x coordinate of uG+vK matches x.

2. The Bitcoin Currency System

2.1. Parts of Bitcoin

The Bitcoin currency system may be seen as a multipart composite of complex, independent developments which made up the Bitcoin platform. Studying each of the Bitcoin constituents on its own will help us devise the global picture of the system. While we have recently been introduced to the cryptographic background guaranteeing identification and secure transactions via hash functions and digital signatures, other parts of Bitcoin will be covered in detail in the present chapter.

Firstly, the **Bitcoin network** provides a basis for peer to peer transaction (and script) broadcasting, which enables moving money across addresses, as well as verifying such transactions and recording them in a unique distributed ledger consensually. The network is the most basic mechanism required to support the Bitcoin monetary system, and does not provide any other extra functionality.

Transactions are supported by digital signatures, so that only the current owner of a number of bitcoins is honoured to transfer such funds to other address. In order to prevent double spending of the same coins, the **Blockchain** was defined as a distributed, unique transaction history since the beginning of the network time. The Blockchain constitutes a full track of every coin fraction (as well as newly created currency), so that one holder is not allowed to sign two transactions for the same coin fraction (only one of them will pass through to the blockchain).

Choosing which transactions are to be included in the immutable, tamper-resistant blockchain is not an easy task, taking into account that this database is to be distributed and all copies must match among the entire, decentralized network. For this requirement to happen, a combination of provable hard work and incentivation is used. The process known a **mining** yields unique incremental updates to the blockchain in the form of blocks. A block groups all new transactions known since the last update, and they are produced and broadcast to the network by miners, by making it match a partial hash requirement (each block includes some reserved fields which are set freely for this purpose). The hardness of finding a valid block leads to a constant production rate which ensures the uniqueness of the current blockchain. Those miners solving blocks are rewarded the fees offered in each transaction together with newly created coins, constituting this a basic **incentivation** for those preserving the network.

2.2. Transactions and the Blockchain

Transactions are the basis of the Bitcoin system, representing currency movements across client addresses. Two key features support the motivation for secure transactions: digital signatures (which guarantee authorship) and the work-provable blockchain (preventing double spendings).

2.2.0. Keys and Addresses

Each client is identified in Bitcoin by their user address. These are strings of alphanumeric characters derived by a collision-resistant algorithm relying on hash functions. Addresses are hence the equivalent to account numbers in banking.

Being present in the Bitcoin network requires bearing at least one user address. Anyone can create such addresses freely by choosing a private key and computing its associated public key, as described in the Elliptic Curve Encryption scheme in Section 1.3.1. The public key constitutes the input for the address encoding algorithm, and funds in Bitcoin are be solely associated to this address, while further transactions from the account will require digital signing with the private key and may be indeed verified using the associated public key. It is therefore worth stressing out that the private key allows for access to the money stored in the account, and therefore it should be kept secret.

As stated before, a public key is formed by an integer coordinate pair (X,Y), each of them being a 256-bit integer. While specifying both X and Y was mandatory in order to describe a public key in early Bitcoin versions, the Y component was later dropped in favour of shorter transmission and storage requirements, being since Y computed from X (two possibilities still remain, depending of the sign of Y, given the symmetry the curve exhibits, as detailed later).

The above motivated two possible key formats: one older format, which required both X and Y coordinates to be specified, and a modern variant, known as **compressed key** format, in which only the sign of Y is needed. Storing and distributing a compressed key takes nearly half as much space as the full expanded key, which was the reason why compressed keys were considered.

The process of encoding a public key into an alphanumeric address string begins by prepending a one-byte identifier to the key binary expression (being both X and Y encoded as IEEE 256-bit signed integers), which may take three forms:

Form	**Prefix**	**Key data**
Uncompressed public key	0x04	512 bits X, Y (each 256-bit)
Compressed public key, even Y value (as encoded by IEEE 256-bit signed integer)	0x02	256 bit X
Compressed public key, odd Y value (IEEE 256-bit signed integer)	0x03	256-bit Y

Table 5: Public key formats

Afterwards, the prefixed value is then fed through a SHA-256 hash function, obtaining a 256-bit digest which is again hashed under RIPEMD-160. The obtained 160-bit result is prefixed with a constant version number 0x00. The result is finally coded as a base-58 number using the 0-57 figure value alphabet:

```
123456789ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijkmnopqrstuvwxyz
```

The main purpose of the selected symbol set is facilitating the human readability of Bitcoin addresses while preventing common errors in misspelling (like taking O for 0, capital I for lowercase l, etc.) Finally, a base-58 4-byte checksum is appended to the address, making it possible to detect further transmission errors. The checksum is computed as the 4 highest bits of the result of a double SHA-256 hashing performed on the 168-bit unencoded address.

As may be noted from the method described above, the two possible encodings resulting from uncompressed/compressed keys do produce different actual addresses, which are in fact considered different clients in Bitcoin, even though both of them represent a unique public key derived from the private key.

Recalling the previous chapter, a private key is basically the 256-bit exponent used to compute the public key point (X,Y). As a result, in terms of (private) storage and backup, a private key should not be affected by compression considerations. However, given that the two existent public key encoding standards lead to different addresses, specific information on which public key format is to be used should be stored together with the private key. The common format employed in wallet software (managing addresses and transactions) and backups for storing private keys to be used for compressed public key generation suggests appending a suffix 0x01 (no suffix is used in the case of uncompressed public keys). The encoding process stays the same as with public keys (SHA-256 followed by RIPEMED-160), except that this time a prefix 0x80 is added before encoding and computing the checksum, indicating a private key follows.

Form	**Prefix**	**Suffix**	**Key data**
Private key with associated uncompressed public key	None	None	256-bit exponent
Private key with associated compressed public key	None	0x01	256-bit exponent

Table 6: Private key formats

Multiple addresses may be created and used by the same client for purposes such as security and preventing traceability and certain attacks based in computing time advantage. In most scenarios, the task of dealing with client addresses and transaction relies on specific wallet software, taking care of generating new addresses for the funds which are moved, and managing the private and public keys securely.

2.2.1. Transactions

A transaction in Bitcoin is a properly formed data block defining which amounts of funds are to be transferred from one address to another. The original owner of the funds to be moved is required to provide a digital signature certifying the transaction as originating from him (as identified by its address), and must also specify the sources of the monetary inputs that are to be transferred. Money in Bitcoin can be acquired by means of previous transactions or mining rewards when solving a block (these are incentives rewarded by miners for their work).

The inclusion of such information on the source of the funds related to the transaction allows any other peer in the network for easy verification of whether the funds belong to the stated party (the transaction issuer). Keeping in mind that the Blockchain contains all the transaction information history and new currency created (as rewards to miners), it should be trivial for any other party to check the validity of the transaction and the enclosed signature by its originating author.

The amounts from several previous transactions (*outputs*) may be combined as sources for the new transaction to be issued. Furthermore, it is possible to redirect unused units back to its original author (or any other account of his) as *change* in case the gathered amount will not sum up the exact quantity to be transferred. This mechanism, together with the ability to define and consume fractional coin quantities, confers Bitcoin transfers quite a handy flexibility.

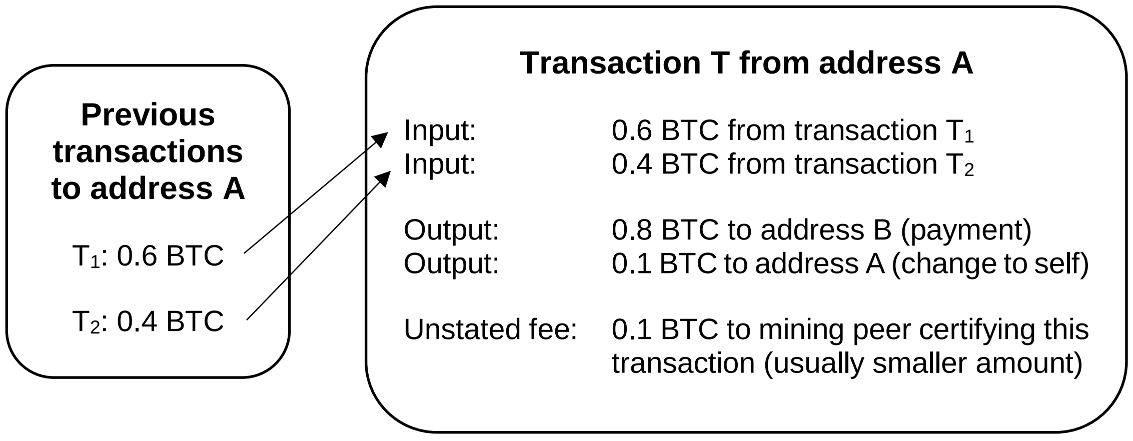

Previous transactions to address A
T_1: 0.6 BTC
T_2: 0.4 BTC
Transaction T from address A
Input: 0.6 BTC from transaction T_1
Input: 0.4 BTC from transaction T_2
Output: 0.8 BTC to address B (payment)
Output: 0.1 BTC to address A (change to self)
Unstated fee: 0.1 BTC to mining peer certifying this transaction (usually smaller amount)

Figure 7: Outputs from previous transactions consumed as inputs to new transaction

Adding up to the design, an unstated surplus may be left undefined in the transaction details, meaning the inputs (previous monetary sources) minus any change involved need not meet the exact amount to be moved (still, they cannot sum any lower than that). In this case, the unspecified remainder is considered as **transaction fee,** or incentives to the that miner certifying the transaction with a proof of work. Such fees aim for a faster acceptance of the transaction by the entire network, since one transaction may not be free from the risk of other parallel spending of the same referred inputs until it is approved by the whole network, by means of it being included in a block within the Blockchain.

2.2.2. Scripts

Scripts are fields present in each transaction block which provide a basic programmatic mechanism for an owner of an address to redeem previous transaction outputs, so they can be spent in further transactions.

By using a reduced instruction set opcodes, the owner of an address private key may provide their public key and a valid ECC digital signature, qualifying them as the owner of the funds, so they can be consumed as inputs in a new transaction.

Generally speaking, the scripting language allows for more complex checks than the aforementioned basic identification via digital signature, being in fact possible to define advanced cryptographical **contracts** under which, prior to being possible to spend the funds, various types of conditions are to be met, such as providing several digital signatures or a specific password, among many others.

The transaction block format defines the following fields, among them, the several inputs and outputs of the transaction, and one script per input/output. Scripts associated to each input (as outputs from a previous transactions) provide a certificate which unlocks the inputs so they can be spent within the transaction, while those scripts associated to each output set constraints for such quantities to be spend in future transactions (i.e., require a signature from address owner).

Field	**Size**	**Value**
Version	4 bytes	Currently 1
Input count	1, 3, 5 or 9 bytes (depending on first byte value)	Number of referred transaction inputs following
Inputs	Variable	Array of transaction input blocks
Output count	1, 3, 5 or 9 bytes (depending on first byte value)	Number of transaction outputs following
Outputs	Variable	Array of transaction output blocks
Lock time	4 bytes	Date from which transaction is valid (usually discarded, set as 0xFFFFFFFF)

Table 8: Bitcoin transaction block format

The format of each input and output block is depicted below:

Field	**Size**	**Value**
Transaction pointer	32 bytes	SHA-256 double hash of the transaction block containing the output referred as input
Output index pointer	4 bytes	Transaction output index
Unlocking script length	1, 3, 5 or 9 bytes (see above)	Size of script code following
Unlocking script	Variable	Script code unlocking the input
Sequence number	4 bytes	Currently unused, set to 0xFFFFFFFF

Table 9: Input block format within a transaction

Field	**Size**	**Value**
Amount in Satoshis	8 bytes	Output amount (1 BTC = 10^8 Satoshis)
Lock script length	1, 3, 5 or 9 bytes	Size of script code below
Lock script	Variable	Lock script code

Table 10: Output block format within a transaction

The (un)locking scripts form output-input pairs: for each referred input in a transaction, there exist two associated script pieces: one from the previous referred transaction output (which is used as an input in the current transaction) and one provided with the current input. Each pair of scripts are concatenated together, being the base transaction script at the end, and executed by a virtual machine equipped with a stack register. In case the machine code yields a True value on top of the stack, the test passes and the input may be redeemed. Otherwise, the transaction is considered invalid and won't be accepted.

Typically, transactions move amounts of bitcoins from one address to another. In this scenario, the transaction output is provided with a script requiring that the public key and signature of the account owner are present in order to spend such output as a new transaction input, being such values provided by the new transaction input unlocking script. This kind of transactions may be considered by the receiver as a valid movement retitling the owner of the coins to them, so that no one else may reclaim the output. Nevertheless, this format is not mandatory, and further conditions may be imposed, making complex contracts.

The current supported scripting instruction set is extensive and will not be covered in this text. The reader may refer to [Script, 2017] and [Contract, 2017] for a comprehensive description of the supported language and samples of different types of contracts. For our purposes, we are illustrating below the scripts associated to a "Pay-To-PubkeyHash" operation, commonly used for transfers between accounts (as opposed to "Pay-To-Script-Hash", used in contracts):

Unlock script in new transaction input:
Push to stack this transaction signature using the address' private key Push to stack the full address' public key
Lock script in previous transaction output:
Duplicate the top (last pushed) stack element
Apply RIPEMD-160 to top stack element
Push destination address to stack
Check if top two elements in stack are equal (if so, remove both elements from stack and continue; if not equal, return False)
Check if top two elements in stack correspond to a valid signature and public key (if so, return True; if not, return False)

Table 11: Common Pay-To-PubKey-Hash script pair

After running the sequential program defined by both script pieces, the returned value will mark the status of valid claim of the previous output. Note that, by this design, it is the issuer of the transaction who sets the constraints on the new ownership conditions in their outputs (as their script is executed at a later stage), qualifying only that one with a private key for further movements of this amount.

2.3. Mining and consensus

As we have just covered, the ECC Digital Signature system provides support to secure transactions, in which particular credentials (such as an address' private key) are required as a certificate of ownership when spending the received funds.

The signing procedure does not exclude, however, that the same output is redeemed twice on multiple, properly signed transactions. As a shortcoming to this problem, the Blockchain was conceived as a distributed database across all peers in the network. Such a ledger does stipulate that once an output is spent, no further transactions may claim it again, being the whole peer base able to check the historic record of movements for matches since the network inception.

Having a large decentralised network agreeing on a common transaction history is not a trivial task. Once a transaction is received by a peer, it is reasonable to include it in the central database, but how to guarantee that this new data is received by all network peers so that no one else may be tricked into thinking the amount was not previously spent? This problem is addressed by the Blockchain.

2.3.1. The Blockchain

The Blockchain, as suggested by its name, is an array of chained blocks, each containing a collection of transactions. Updating the chain by adding blocks to it is a demanding task, consisting of drawing a random pick of a peer, known as **miner**, by weighting the selection by the computational power incurred by each of them. Such a selection can be done by setting a requirement to solve a hard, time-consuming problem which may in fact be quickly verified by anybody.

Field	**Size**	**Description**
Block size	4 bytes	Size of block in bytes
Version	4 bytes	Currently 0xD9B4BEF9
Previous block hash	32 bytes	Double SHA-256 hash of previous block header in the chain
Merkle root hash	32 bytes	Transaction data hash (see later)
Timestamp	4 bytes	Time of block creation (UNIX Epoch format)
Difficulty target	4 bytes	Current proof of work hardness
Nonce	4 bytes	Freely tunable values to produce the proof of work
Transaction count	1, 3, 5 or 9 bytes	Number of transaction blocks following
Transactions	Variable	Array of transaction blocks included in the block

Table 12: Block format

Miners collect transactions from the network in a block, which they try to **solve**. Solving a block involves making it meet a requirement on its hash, for which the field Nonce may be freely set. Whenever a miner finds a solution for a block, it broadcasts it so the rest of the network may check it easily by computing its hash. Blocks are thus chained forming a string known as the **Blockchain**.

Approved blocks cannot be altered in any of their fields without recomputing a valid hash, a task which grows exponentially harder as the chain expands, due to block linking by specifying the previous block hash. As a result, the chain will stay immutable, as well as all transactions listed in the blocks, given the hard computational work required to rebuild it to its current length with different data.

The hash requirement for a block to constitute a valid piece of the chain is related to the number of zero bits preceding the hash value. As the number of required zeros grows, so does the hardness of finding a proper block. This **difficulty** level is stored in the block header and adjusted according to the previous network hash rate, so that an average of one block is produced every 10 minutes.

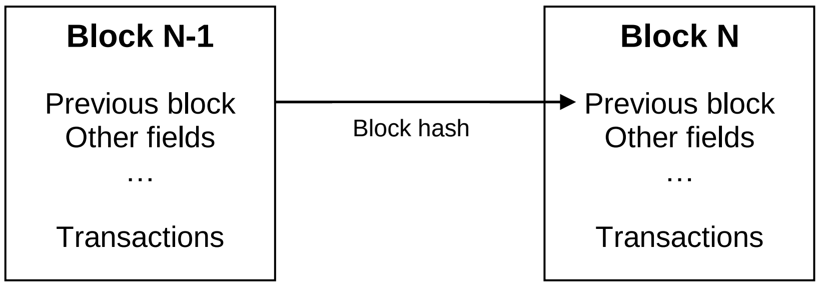


Figure 23: Block chaining

The **Merkle tree** parameter was designed for efficient verification of the presence of a specific transaction within a block, so that it would not be required to scan the full blockchain nor every transaction in the containing block. Transactions within a block are hashed in pairs, and their hashes hashed again in pairs, building a binary tree structure which allows for partial verification when all branch hashes are provided. The root of the tree is included in the block header, so that a change in a single transaction would still invalidate the whole block hash.

The whole blockchain is based on a single initial block, known as the *Genesis* block, which is hardcoded in all Bitcoin software as a trusted chain root.

2.3.2. Mining incentives and currency issuing

In Section 2.2.1, it was discussed how an unstated remainder in the sum of transaction inputs is considered as a fee. When a miner solves and commits a new block to the blockchain, the transaction fees in the block are collected in a trailing transaction in the block by the miner, thus constituting a reward the miners receive as a payment for the work contributed to preserve the network safety.

Apart from the fees defined in each transaction, new coins are created for every block solved. The amount of newly issued coins, which may also be attributed to the miner creating the block, are halved progressively through the years, being 2024 the year by which no more coins will be issued and mining is expected to be solely compensated by a (presumed) increase in transaction fees.

The above policy indeed incentivates the computations which help making the blockchain stronger. In the event that an attacker would decide to alter blocks by redoing the chained hash work, it must be the case that a vast computing power (outperforming the entire well-behaved network) will be required.

2.3.3. Consensus

We say that a transaction is valid if it is well-formed and the input-output pairs point to previously unspent outputs which are unlocked by the given scripts (i.e., they are all valid). Hence, a block is valid if the proof of work and all transactions it contains are valid. Invalid blocks are discarded and will not pass through the blockchain. Whenever a valid block is received, all peers will trust the included transactions and add them to their local copy, thus reaching a global consensus.

Due to uncertainty on the distribution of blocks across the network peers, caused by network latency or link ruptures, it may be possible that the blockchain will branch on different valid paths. If this is the case, later updates to the blockchain will still be accepted as long as they span over the current (local peer version) of the chain. When different valid branches are received by a network node, the one featuring a higher complexity proof of work (i.e., longer length) will be selected.

The global consensus over a decentralised peer-to-peer network was the key feature which raised Bitcoin as the first authority-independent digital currency.

3. Beyond Bitcoin: Other Fair Play Applications

Fairness in Bitcoin follows the principle of “voting by computing power”, meaning that the computing capacity of each client in the network is the measure for their decision-making capability. This approach is far from ideal, but still it is more suitable than those methods based on identities (such as one vote per IP address or user account), which show a tendency to fail at guaranteeing fairness.

The idea behind Blockchain as the only trustworthy party in the network to address the problem of double-spending (this is, taking in some way the role of a central authority), has inspired many other remarkably similar developments of miscellaneous nature, like virtual nations or tamper-resistant health records.

On the other hand, Bitcoin itself can be taken advantage of, thanks to such features as its scripting language and the transaction locking times, for designing various theoretically-secure Fair Multiparty Computation protocols (by relying on the presumed security of the cryptographic primitives used in Bitcoin), in which no party trusts each other. This will be the central topic covered in this chapter.

3.1. Other cyptocurrencies

Shortly after the emergence of Bitcoin as the world’s first decentralised digital currency in early 2009, further cryptocurrencies’ designs were released based on the very same design principles as Bitcoin, usually these including a bigger feature set and additions. These projects are collectively known as **altcoins**.

Among the most popular and currently alive digital currencies, we list:

- **LiteCoin**: This was one of the first forks of the Bitcoin project, featuring lower block production rates and fastest transaction approval, which is currently one of the main drawbacks of Bitcoin.

- **DarkCoin (Dash), Monero**: These cyptocurrencies address the problem of lack of anonymity with Bitcoin transactions, on which statistical tests may be taken to link transactions with their actual owners with certain probabilities. The way to hide the real pathways of the money is by mixing multiple transactions from several recipients, leading to much difficult traceability.

- **Ethereum**: This digital asset is double-edged, meaning that, apart from currency uses involving transactions, it supports a Turing-complete scripting language enabling virtual machines and decentralised applications supported by the currency itself.

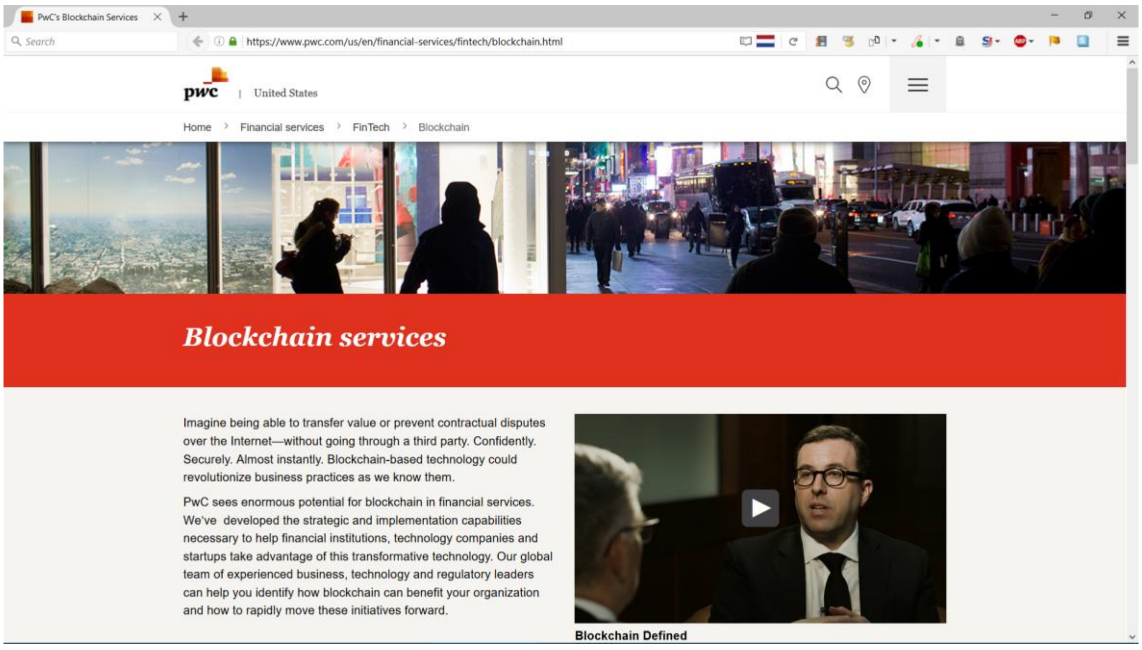
- **Zcoin, Komodo**: Full anonymity during transactions is provided by introducing an alternate auxiliary coin supporting two different kinds of transactions and proofs of work based on Merkle Trees, which allow for ownership certification while fully hiding the origin of the funds.

Yet another breaking feature present in most recent digital currencies is the ability to reach global consensus based on the consensus taken in another problem. Since Bitcoin currently holds the most powerful hash rating, it is frequently the selected basis upon which consensus for other transactions in altcoins is taken.

It must be remarked that Bitcoin was not designed for supporting the addition of new features, and, as such, it is very unlikely that any of the newer developments (like the Zerocash protocol supporting fully anonymous transfers) will ever be implemented. This lack of flexibility of Bitcoin is also due to problems derived from the lack of wide acceptance of deep changes by investors and supporters, which would incur in a loss of their money (both as currency and on equipment). As a result, too much pressure is put on the developing team, which is reticent to introducing radical changes in the system.

3.2. Other blockchain applications

The use of blockchain as a collective, common (as unique), publicly checkable record does not only have applications in the field of monetary transactions by guaranteeing no double spending is possible. Furthermore, by relying on **smart contracts**, it is possible to support several other kinds of applications which are either based in the current blockchain from Bitcoin, or develop their own chain.


PwC's Blockchain Services
Search
https://www.pwc.com/us/en/financial-services/fintech/blockchain.html
pwc
United States
Home › Financial services › FinTech › Blockchain
Blockchain services
Blockchain Defined

Imagine being able to transfer value or prevent contractual disputes over the Internet—without going through a third party. Confidently. Securely. Almost instantly. Blockchain-based technology could revolutionize business practices as we know them.

PwC sees enormous potential for blockchain in financial services. We've developed the strategic and implementation capabilities necessary to help financial institutions, technology companies and startups take advantage of this transformative technology. Our global team of experienced business, technology and regulatory leaders can help you identify how blockchain can benefit your organization and how to rapidly move these initiatives forward.

Figure 14: PwC's Blockchain Services website, https://www.pwc.com/us/en/financial-services/fintech/blockchain.html

Among other possible developments based on the blockchain technology, we may list those providing support for property titles, licenses, marriage or death certificates, inventories, etc. All such options may be handled with a similar treatment like that of the digital currencies, and are commonly referred to as **smart properties**. Such titles could in fact be stored in the blockchain and be checked in a secure, private manner without exposing any private personal data to the public network. As a matter of fact, basically any asset or good be digitally encoded and its ownership protected under a private key within the blockchain.

In a similar fashion, insurances or licenses could be checked about their validity or current bearer, and digitally issued and revoked via public key cryptography and hash functions. On the other hand, digital goods such as software licenses, access keys (be they physical or digital) could also benefit from this design, by being sold, transferred or cancelled in same way.

One topic particularly popular in the last years is the new economy business support model of crowdfunding, which could as well be implemented with the current blockchain technology, for instance by creating digital currencies inside or outside the Bitcoin network, used to collectively fund emerging start-ups or projects, with no prior requirement for third parties being involved.

3.3. Secure multiparty computations

A Secure Multipart Computation (MPC) is a problem of computing a function value whose input is multiple and secret to the participants in the computation, $f(s_1, \ldots, s_n) = v$. Each party holds one part of the input, which should be unknown to all others. The computation involves finding the function value preserving the inputs secrecy, while at the same time providing a proof for correctness which any party may verify. Size on the inputs is unconstrained (they may be even null).

Typically, multiparty computations take place in gambling or simpler games based on randomness with a reduced number of participants. Examples include: coin tossing elections, lotteries, card games or timed secret commitments. Commonly, these scenarios involve a bet of money and require some sort of fairness, for which, as we will see, Bitcoin may provide the required background.

3.3.1. The coin tossing problem

Starting off with an attention to the coin tossing problem as the simplest example, several theoretical approaches have been defined to perform a fair execution, but none of them addresses the case of sudden rupture of the protocol by one party.

Let two participants take part in the problem of uniformly selecting contestant A or B at random. Performing the “coin tossing” may seem obvious when both parties share a common scenario in which they can carry an actual physical experiment. When it comes to the case when participants only share a communication channel (i.e., telephone call), things get not so straightforward.

One possible solution would be a mutual agreement on the result of a trusted third party experiment they may both check (like the parity of the number drawn from a forthcoming lottery). Another possible approach could be one party generating a secret outcome (heads/tails) and sending the other party the hashed result of this value plus some additional data (anything will do). The receiving party would then pick their guess and let the originator know about it, which in turn may safely disclose the secret and added data for the other party to check the winner in a trusted manner, as no tampering is possible in this construction.

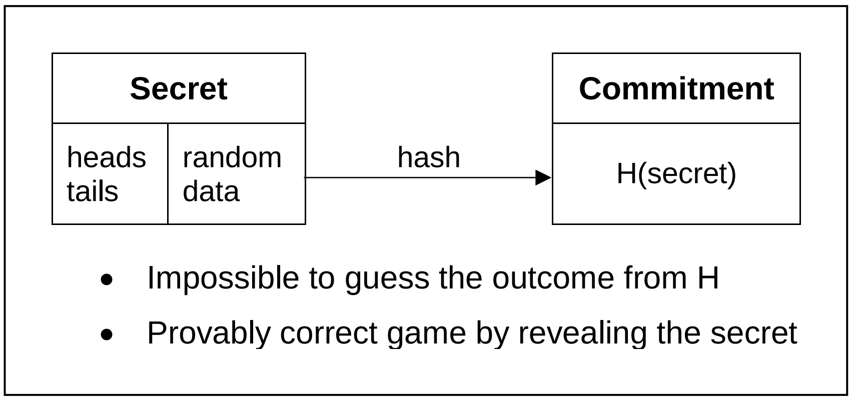

Secret
heads
tails
random
data
hash
Commitment
H(secret)
Impossible to guess the outcome from H
Provably correct game by revealing the secret

Figure 15: Coin tossing cryptoscheme

However, none of the protocols described above is able to deal with the scenario of some contestant prematurely leaving the game before its conclusion (i.e., A may leave in case B won without paying off his debt or even without B being aware of it). This problem is frequently solved, as covered below, by setting up a deposit to be redeemed by the well-behaved parties in case one participant leaves the game. In this context, the term **fairness** relates to compensating the fair players in case of cheating or any other circumstance biasing the game ruling.

The coin tossing problem will be revisited as a particular case to the general lottery problem, described in detail below.

3.3.2. Fairness under Bitcoin

Most MPC problems boil down to collectively selecting uniform random numbers guaranteeing that no party is able to introduce any bias on them. When it comes to fairness, the usual design goes though pre-filling a deposit before the actual game stages take place, which will be used to compensate well-behaved parties in case of an early termination of the protocol by any party.

The scripting language defined in Bitcoin can be effectively used to model the pre-existent conditions for any kind of definable problem, as well as post-conditions the winner(s) are to meet, all linked to currency exchanges. For instance, regarding the coin tossing protocol, the commitment to the hash of the outcome can be encoded programmatically in a transaction script, just like the later disclosure of the secret and payment to the winning party.

As for fairness, the common scenario to be dealt with is the premature rupture of the protocol (i.e., cheating, leaving or actual communication issues). This may be addressed by relying on the transaction lock time field. This value allows to delay the commit time when then transaction will be valid if the specified inputs were not previously redeemed by other transactions. As such, by setting this field we may actually enforce a deposit payment in case the game will not finish properly.

The sort of transaction that will be issued during the protocol execution will not be the common standard transactions moving money from one account to another. Because of the fact that transactions are **malleable**, meaning that their hashes may be changed while preserving the script equivalence (for instance, by adding NOP instructions), many miners in the Bitcoin network refuse to accept other transactions than Pay-To-PubKey-Hash (those moving amounts of money across addresses). This may constitute a problem for the implementation of the methods defined below, but still, there are some more tolerant miners which will accept non-standard transactions, a chance which may be increased by attaching proper fees to such transactions (as discussed in the final part of this chapter).

Lock script (in output), Pay-To-PubKey-Hash:
```
OP_DUP OP_HASH160 <pubkey_hash> OP_EQUALVERIFY OP_CHECKSIG
```

Unlock script (in redeemed input):
```
<transaction_signature> <full_pubkey>
```

Alternative (equivalent) unlock script:
```
OP_NOP OP_CODESEPARATOR <transaction_signature> <full_pubkey>
```

Figure 16: Transaction malleability

The figure shows how a legit redeeming transaction may be forged into another equivalent form, given that the signature cannot cover the script field itself, and so it is excluded from the computation and may be reproduced in the alternate transaction. The signing procedure may not be broken, so the funds cannot be directed or spent elsewhere, but still, as the transaction hash will change, the new transaction may reach the blockchain before the original, fooling wallet software.

3.3.3. Timed secret commitment

The timed secret commitment protocol involves N participants P_i and a host C, who commits to the disclosure of a secret s by time t, or else pay a refund of d BTC to each party. Ideally, such a secret is to satisfy a specific property (like been a valid decoding key for some data). This problem features properties of fairness, being the parties P_i rewarded in case the exchange does not succeed.

As long as the proving test to check s for validity can be encoded in the Bitcoin script language, the whole protocol may be carried out within Bitcoin itself. This validity property is optional, and in our description, it will replaced by an early released plain hash check, H(s) = <provided_hash>.

Avoiding to develop a complex unintelligible notation, we will describe the protocol ruling below using natural language:

1. C broadcasts a transaction with d outputs of N BTC each. Each output i may be redeemed by providing values s, sig_C, sig_P such that:

H(s) = <specific_value>, and sig_C is a valid signature of the redeeming transaction by C
or
sig_C and sig_P are valid signatures of the redeeming transaction by C and P, respectively

Note that at this point, C may only reclaim their money back by disclosing s, so the **commitment** is made as soon as this transaction appears in the chain.

2. For each output i of the previous transaction, C creates a signed transaction body transferring the d coins to P_i by time t (set as time lock), and sends it to P_i without actually broadcasting it to the network.

   These transactions act as **deposits** which each party P_i may sign and broadcast to receive d BTC once time t is reached, provided that the inputs were not reclaimed before (note the only way in which this last condition may fail is by C revealing the secret s in order to meet the first condition in 1).

3. The committer C gets their deposit back by revealing the secret s (and thus fulfilling the first condition in 1).

   This stage is known as the **opening** phase. If C fails to reveal s by time t, the parties P_i will receive the deposit of d BTC each.

3.3.4. A lottery protocol

The fair lottery problem is defined as N participants funding a collective pot with d BTC each, which is finally fully paid to one of the parties, selected uniformly randomly. This is a general case for the coin tossing problem where N = 2.

The key aspect to establishing the fairness of the protocol is each party issuing a deposit to compensate all other parties in case of protocol halting. Getting the deposits back would imply disclosing the secret value the party used for collectively generating the random outcome (i.e., a modular sum). However, since the deposits are time-locked, revealing all secrets before the deadline would allow the honest winner to claim the price. Since the deposits are meant to compensate all other parties, they are much higher than the invested money, so it is sensible to expect that bad behaviour will not be worth the expense.

The winner may be picked by each party secretly choosing a random number, which will all be summed up and the remainder modulo N taken to select the winner. Having at least one party selecting their secret number truly randomly will suffice for the final outcome to be random: $f(s_1, \ldots, s_N) = s_1+\ldots+s_N$ (modulo N).

The following describes the lottery implementation of N participants in Bitcoin. It is assumed that all parties know the public keys of others. In this design the role of the committer does not apply (all players behave as so, indeed):

1. Each player P_i randomly selects a secret number s_i, computes $h_i = H(s_i)$ and issues a transaction of N-1 outputs of d BTC each, as before, for each output $j \neq i$ to be redeemed by giving proper values s_i, sig_{P_i}, sig_{P_j} such that:

$H(s_i) = h_i$, and sig_{P_i} is a valid signature of the redeeming transaction,
or sig_{P_i} and sig_{P_j} are valid signatures of the redeeming transaction

   In case one player did not issue all proper transactions using the same secret value, or if they are not shown in the blockchain in a time limit, all parties may draw back their money by revealing their secrets and signing the transaction. The same thing will happen if different players provide the same committed values $h_i = h_j$, which would clearly indicate copying from one of them (the consequences of such behaviour can be seen in the case where N = 2).

2. Secondly, each P_i creates a signs a time locked transaction paying each party P_j for $j \neq i$ the amount of d BTC from previous transaction, with the parameter t high enough for this **deposit** be paid at the end of the game in case of issues. Each P_j adds their signature and broadcasts the transaction to the network.

   At this point, deposits have been set to be returned at a time limit, expected to happen after the estimated game time. The lottery execution follows.

3. Each P_i creates and broadcasts a transaction of d BTC, which may be redeemed by a transaction signed by P_i. This is known a **payment** phase.

   If some party fails at getting their transaction published in the blockchain within specific time limit, all other parties may reclaim their deposits and money back by revealing their secrets and signing the right recovery transactions.

4. A collective transaction is created from all previous payments, used as N inputs. Each input requires that this new transaction is signed by its issuer. For this purpose, one party acts a collector, receiving the signatures of this transaction by all others, and finally broadcasts the transaction to the network. This transaction has one output to be paid to the winner, by setting the redeeming condition that the provided values $s_1, \ldots, s_N$ and sig meet:

$H(s_1) = h_1$ and … and $H(s_N) = h_N$ and $f(s_1, \ldots, s_N) = 0$ and sig is a transaction signature by P_1.
or
…
or
$H(s_1) = h_1$ and … and $H(s_N) = h_N$ and $f(s_1, \ldots, s_N) = N-1$ and sig is a transaction signature by P_N.

   Once this transaction appears in the ledger, the game is set. In case it does not (for instance, if one party failed to provide their signature to the collector, the parties will be able to reclaim their money deposits after waiting for the time lock to be reached, as well as the d BTC they bet from the pot (they will not be able to do so, however, if this transaction appears in the blockchain).

   In order for any player to get their deposit back, it is required that their secret value is disclosed. Since deposits will not be paid until a certain time limit, the winner may claim the price by providing all secrets and signing the transaction to redeem from this final transaction dN BTC. In case one party refuses or fails at disclosing their secret value, their deposit of d(N-1) BTC will be claimed by all other parties, each receiving the invested d BTC.

While the defined protocol is fair, it will not scale up well. As the number of participants raises, the chances for some of the parties at failing disclosing their secret increases. This is not a big deal, though, as all well-behaved players will still receive a reward in that case. Apart from this, issues can be clearly seen, as in the case of Bitcoin, when a majority of players act maliciously.

3.3.5. Introduction of fees

The protocols described above assume no fees are imposed in any transaction. This however is not the common case in Bitcoin, where fees help for a faster processing and approval of the transactions by the mining peers. Taking into account that we are making use of non-standard transaction forms (unlike Pay-To-PubKey-Hash), the need for fees must be considered.

Modifying the methods to include transaction fees is trivial, but the result of fees to be applied is a reduction in the total amounts gained by the winner, and what seems more important, the deposits and reclaimed money in case of protocol failure will be smaller than the initial invested amount (i.e., the players will lose a small part of the money in the fairest case). So, complete fairness cannot be handled when fees are introduced, but their effects are little and can be diminished further by recycling some transactions of previously early-broken games, like those of the initial secret commitments.

3.4. Conclusions

We have covered some basic protocols involving computations taking place in an environment of fairness for the development of some basic games. These constructions can be implemented taking advantage of the Bitcoin infrastructure, which shows how suitable the system is for other kinds of computations. More complex systems like card games could possibly be implemented, too, at the cost of developing very complex logical conditionals.

However, given that Bitcoin was conceived as a monetary transaction platform, unaware of advanced programming topics, its script language is quite elementary and provides support only for a minimum set of required features, disregarding other secondary topics and focusing on preventing issues of minor importance to us, like correctness of implementations, transaction malleability or non-monetary uses in general. All this results in a cut-down scripting language, which strongly limits the chances of it being used for deploying many MPC solutions.

As an example, the problem of some buyer purchasing a license key from a vendor within a time-limited period does not seem to be possible to be defined on the current Bitcoin platform. Limitations for several such designs include the impossibility of knowing the hash of a transaction until it is finally included in the blockchain (which prevents orphaned transactions to be created in advance, as its input is yet unknown), or the lack of support for selecting which transaction fields are to be included in the signature (even setting a condition on the transaction body hash is not supported).

Right after the emergence of Bitcoin, several other cryptocurrencies and many other applications based in the blockchain technology were launched, many of them including new features, which were not devised in the early Bitcoin design. Some of such systems, like Ethereum, do provide monetary capabilities as well as a richer scripting language (being it even Turing-complete) which surely fits any requirement of any MPC scheme, making of it a suitable choice.

Still, Bitcoin covers many state-of-the-art concepts like global consensus reaching, proofs of work and programmability, which make it interesting by itself, and, while perhaps not being the most suitable platform for supporting MPC, it is the largest and most used digital currency nowadays, with a design in which many newer technologies and all altcoins were based, constituting a technological innovation in Science as well as a major success in Economy.

3.5. Acknowledgments

The author would like to thank his thesis advisor, Dr. Farràs Ventura, for his guidance and support, and for bearing with issues related to timing, which delayed the development of this essay at the early stages.

5. References

Andrychowicz, M., Dziembowski, S., Malinowski, D., & Mazurek, Ł. (2014). Secure Multiparty Computations on Bitcoin. *35th IEEE Symposium on Security and Privacy*, (pp. 443-458). San José, California. Retrieved from http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6956580

Antonopoulos, A. M. (2014). *Mastering Bitcoin. Unlocking Digital Cryptocurrencies.* O'Reilly Media.

Back, A. (2002, August 1). Hashcash - A Denial of Service Counter-Measure. Retrieved from http://www.hashcash.org/papers/hashcash.pdf

Ben-Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., & Madars, V. (2014). *Zerocash: Decentralized Anonymous Payments from Bitcoin.* Retrieved from http://zerocash-project.org/media/pdf/zerocash-extended-20140518.pdf

Bitcoin Network. (n.d.). Retrieved 2017, from Wikipedia, the Free Encyclopedia: https://en.wikipedia.org/wiki/Bitcoin_network

Block. (n.d.). Retrieved 2017, from Bitcoin Wiki: https://en.bitcoin.it/wiki/Block , https://en.bitcoin.it/wiki/Block_hashing_algorithm

Brown, D. R. (2010, January 27). SEC 2: Recommended Elliptic Curve Domain Parameters. (Version 2.0). (S. f. Cryptography, Ed.) Retrieved from http://www.secg.org/sec2-v2.pdf

Comparison of Cryotocurrencies. (n.d.). Retrieved 2017, from Bitcoin Wiki: https://en.bitcoin.it/wiki/Comparison_of_cryptocurrencies

Contract. (n.d.). Retrieved 2017, from Bitcoin Wiki: https://en.bitcoin.it/wiki/Contract

Cryptographic hash function. (n.d.). Retrieved 2017, from Wikipedia, the Free Encyclopedia: https://en.wikipedia.org/wiki/Cryptographic_hash_function

Descriptions of SHA-256, SHA-384, and SHA-512. (2000). Retrieved May 2013, from http://csrc.nist.gov/groups/STM/cavp/documents/shs/sha256-384-512.pdf

Dobbertin, H., Bosselaers, A., & Preneel, B. (1996). *RIPEMD-160: A Strengthened Version of RIPEMD.* Retrieved from http://www.esat.kuleuven.be/~bosselae/ripemd160/pdf/AB-9601/AB-9601.pdf

Hankerson, D., Menezes, A. J., & Vanstone, S. (2004). *Guide to Elliptic Curve Cryptography.* Springer.

Nakamoto, S. (2008, November). Bitcoin: A Peer-to-Peer Electronic Cash System. Retrieved from https://bitcoin.org/bitcoin.pdf

Proof of Work Difficulty. (n.d.). Retrieved 2017, from Bitcoin Wiki: https://en.bitcoin.it/wiki/Difficulty

Script. (n.d.). Retrieved 2017, from Bitcoin Wiki: https://en.bitcoin.it/wiki/Script

Swan, M. (2015). *Blockchain. Blueprint for a New Economy.* O'Reilly Media.

Transaction. (n.d.). Retrieved 2017, from Bitcoin Wiki: https://en.bitcoin.it/wiki/Transaction

Transaction Malleability. (n.d.). Retrieved 2017, from Bitcoin Wiki: https://en.bitcoin.it/wiki/Transaction_Malleability

Wagner, D. (2016, April). Technical Perspective: Fairness and the Coin Flip. *Communications of the ACM, 59*(4), 75.