Beyond Bitcoin – Part I: A critical look at blockchain-based systems

Pasquale Forte[1], Diego Romano[2], and Giovanni Schmid*[2]

[1]PA Advice, Naples, Italy
[2]Cnr ICAR, Naples, Italy

December 1, 2015

An updated and revised version of this work is available for open access at:
10.3390/cryptography1020015

Abstract

After more than six years from the launch of Bitcoin, it has become evident that the decentralized transaction ledger functionality implemented through the blockchain technology can be used not only for cryptocurrencies, but to register, confirm and transfer any kind of contract and property. In this work we analyze the most relevant functionalities and known issues of this technology, with the intent of pointing out the possible behaviours that are not as efficient as they should when thinking with a broader outlook. Our analysis would be the starting point for the introduction of a new approach to blockchain creation and management, which will be the subject of a forthcoming paper.

1 Introduction

The existence of digital currencies is strictly related to the broad diffusion of Internet and on-line markets. A digital currency is similar to electronic money but even if it can be used to buy services or physical goods like traditional money, it is not equivalent or linked to any fiat currency. The idea of using a digital currency became realistic when some important developments in cryptography settled multiple fundamental security challenges in money transfers, e.g. the necessary trust in the transaction.
Several digital cryptocurrencies have been implemented in the last 20 years, but only starting in 2008 the idea of a decentralized currency came out. In fact the first implementations employed a point of control over the money supply, which

*Corresponding author: Giovanni Schmid, email: giovanni.schmid@cnr.it

led to their failure since it exposed the whole system to attacks for taking control over it. This happened for both governmental and criminal interests [1, 2]. BitCoin [3] has been created to avoid a single point of attack, staying free of any central authority. It includes a peer-to-peer network, with both a distributed currency issuance system and a transaction verification system, backed by a public tansaction ledger called *blockchain*.
After more than six years from the its launch, it has become evident that the decentralized transaction ledger functionality implemented through the blockchain technology can be used not only for cryptocurrencies, but to register, confirm and transfer any kind of contract and property. For example, public records such as vehicle registrations or marriage certificates could be migrated to suitable blockchains. Blockchain technology enables indeed the creation and management of *smart contracts* [4] and *smart properties* [4]. On the other hand, these two new digital paradigms can be used to give rise to the creation of *decentralized autonomous organizations* [5].

In this work we will consider the opportunities given by the blockchain, analysing its most relevant functionalities and known issues. We will evaluate the features specific to Bitcoin, with the intent of pointing out the possible behaviours that are not as efficient as they should when thinking with a broader outlook. And we will consider some alternatives both in protocols and in systems, which represent the already available solution to some acknowledged flaws. More in details: in §2, we will review some new potential applications, surveying the actual tools that are already implemented and focusing on their functionalities related to the blockchain. Then in §3 we will stress the role of the transaction concept in the actualization of the blockchain, followed by an inspection of the components used to manage the public ledger, with a special look at the case of Bitcoin. A discussion on some computational aspects will be presented in §4, where some serious inefficiencies will be highlighted. Finally in §5 we will draw our conclusions on the blockchain-based system.

2 Potential applications

Considered as a public ledger, in its nature the blockchain is inclined to record formal agreements between parties. This is the case when a contract must be registered by a notary, or when the ownership of something must be publicly advertised and recognized, or even when a property with some automatic behaviour could interact with another subject enforcing a contract. These are loose examples - different from the original recordings of currency transactions - that show some potential applications of the blockchain, assuming that we properly switch the qualities of its records. This approach does not take into account the inner functionalities of the underlying tools, but rather assumes it is a secure and efficient system. In the following we introduce some interesting real applications in this perspective, leaving other more complex cases to next sections.

2.1 Smart contracts

Smart contracts represent the implementation of a contractual agreement, whose legal provisions are formalized into programming code and verified through a network of peers [6]. Indeed these contracts are defined through the code and executed or enforced by the code, without the need for a trusted third party. An example of a smart contract is the enforcement of a bet between two users about the maximum humidity level tomorrow. On the following day the contract is automatically completed by a software program checking the humidity levels provided by a qualified weather service or some given sensors, as stated by the contract itself, reading and trasferring funds from the loser's to the winner's account. Another example is an inheritance gift that becomes available on the childs eitheenth birthday. As should be clear from these examples, in order to set up a smart contract one needs to choose an event or condition which triggers the transaction expected in the contract, and then check with a program that the event or condition has occurred.

According to [4], smart contracts represent a technological advancement to the practice of law, which allow contracting parties to structure their relationships more efficiently, in a self-executing way and without the ambiguity of natural words. Given the reliance on source code, users can model contractual performance and simulate the agreement effectiveness before its execution [6].

We can point out the following three distinctive properties of smart contracts: autonomy, self-sufficiency and decentralization [7]. Smart contracts are *autonomous* in the sense that, after their deployment on the blockchain, they no longer need heed of their creators. Furthermore, they can accumulate capital over time, such as digital currencies or physical assets, and that is what the term *self-sufficient* stands for. Finally, smart contracts are *decentralized* because they are distributed and self-executing across a network of peer nodes.
Some open source projects have been started in the last years to develop programming languages for the easy creation of smart contracts (e.g. [8, 9]). Using these programming languages, users will be able to get increasingly sophisticated smart contracts, designed to satisfy various requirements and to be used in several contexts.

Two main current projects related to smart contracts are Namecoin and Ethereum. We are going to discuss them briefly in the following, in order to gain some insight on goals and functions in this kind of blockchain-based systems.

2.1.1 Namecoin

Currently, the mapping between IP-addresses and the names of devices and services over the Internet implemented through the *Domain Name Service* (DNS) is supervised by the ICANN [10]. ICANN is a central authority whose main task is to establish the *top-level domain names* (TLD) (e.g. ".com", ".it", etc.), and which third party companies, known as *registrars*, are permitted to manage names in each TLD. In fact each registrar deals with accepting domain name orders and customer's requests for Internet services in the TLDs of its compe-

tence.
Government agencies could contact a registrar and induce it to blacklist some devices and services, for the purpose of political censorship. The extent of *Internet censorship* varies on a country-to-country basis: most democratic countries have moderate Internet censorship, whilst other countries go as far as to limit the access of information and suppress discussion among citizens [11].

Namecoin [12] is a blockchain-based DNS that – at least in the intentions of its designers and promoters – cannot be controlled by any government or organization. A blockchain-based DNS system means that DNS lookup tables are recorded on a blockchain and shared this way on a peer-to-peer system, so that naming in TLDs can be managed cooperatively by customers, without control by any registrar. Since domain name registration is nothing else than a special kind of contract between a registrar and its customers, Namecoin is a service for supplying smart contracts which is based on the Bitcoin cryptocurrency. Namecoin has the unique TLD `.bit`, and domains can be registered directly with the Namecoin system or via registration services like `https://dotbit.me/`. Since `.bit` is not managed by ICANN, in order to resolve names within Namecoin tables, one must set up a `.bit` proxy server for the correct handling of those DNS requests.

2.1.2 Ethereum

The newly introduced Ethereum platform is a blockchain-based system aimed at running applications exactly as programmed, without any possibility of downtime, censorship, fraud or third party interference [13]. Ethereum, by implementing a generic programmable blockchain, enables the execution of stateful, fully-customizable smart contracts. Some examples of applications that Ethereum can be used for are: voting systems, domain name registries, financial exchanges, crowdfunding platforms, company governance, intellectual property and smart property (see §2.2).

Ethereum can be described as a virtual machine which makes use of a blockchain with a built-in programming language for developing and deploying distributed applications [13]. What takes care of the internal state and the computations as specified in the blockchain is indeed the so-called *Ethereum Virtual Machine* (EVM). Basically, the EVM works in the same way as any other virtual machine: it takes some high level programming language designed for writing smart contracts, and compiles it into EVM bytecode that the computer, on which it runs, understands. The EVM can be thought of as a large decentralised computer containing millions of objects called *accounts*. Accounts have the ability to maintain an internal database, execute code and talk to other accounts. A smart contract is itself an account. *Externally owned accounts* (EOAs) are controlled by private keys: if you own the key belonging to a given EOA, you have the ability to send *ether* – the Ethereum's cryptocurrency – and messages from such EOA to other accounts in the system.

2.2 Smart properties

Beyond smart contracts, blockchain technology enables the creation of smart properties. The key idea of *smart property* [14] is the assertion of ownership rights for an asset through its registration in the blockchain, secured by means of a private key. Only who has got the private key can ascertain its ownership of the asset, which on the other hand can be verifed by anyone thanks to the corresponding public key. The owner could later sell the asset by givig its corresponding private key to another user, e.g. through the completion of a smart contract.

Some types of property like trademarks, copyrights and patents are inherently smart properties, and their management as such is natural and can be fruitful. Indeed these entitlements can be easily encoded and processed as digital documents. This is not the case with physical assets, where ownership is more exposed to frauds. In fact in order to register a chattel (e.g a car) in the blockchain, we need to attach a uniquely identifiable tag or a chip to it. If the information contained in that tag or chip could be altered, or the tag/chip could be detached, smart property would not be guaranteed.

An actual implementation of smart properties is *colored coins.* Using this term we are talking about a class of methods for representing and managing real world assets on top of the Bitcoin blockchain. In fact it is possible to store a small amounts of metadata on the blockchain to represent asset manipulation instructions. One can encode in a Bitcoin transaction the information that some units of a new asset were issued and credited to a given Bitcoin address, and a real world value can correspond to those units with the issuer's promise to redeem them for some goods or services. For example, a car dealer can issue 100 units of a "Buy car X during the time promotional period Y" asset, and promise to redeem each unit for a right to buy a certain kind of car as part of a promotional offer [15].

2.2.1 Proof of Existence

An interesting implementation of the smart property concept in the context of autorship protection is Proof of Existence [16]. Proof of Existence is a web-based service used to prove the authorship of things such as software or documents. This tool demonstrates document ownership without revealing the information it contains, and it can be used to prove that a document was created at a certain time. When a user requires the proof of existence for a given digital document, the service does not record the content of the document but instead computes a digest of its content thanks to a cryptographic hash function. Later, that digest is inserted into a block of the blockchain, and the block timestamp becomes the document timestamp. If a user tries to insert a modified version of the same document, a new digest is created which – with overhelming probability – will be completely unrelated to the previous one, because of the properties of cryptographic hash functions.

2.3 Decentralized autonomous organizations

The blockchain technology makes the execution of several smart contracts and smart properties possible, enabling their reciprocal interaction in a decentralized and distributed way. *Decentralized autonomous organizations* [5] operate according to rules and procedures defined by smart contracts, and on the basis of ownerships defined through smart properties. In these organizations, machines and people can cooperate without the need to be incorporated into traditional business identities.

The decentralized autonomous organization concept derived from Artificial Intelligence: a decentralized network of autonomous agents perform tasks, which can be conceived in the model of a corporation running without any human involvement under the control of a set of business rules. Such organizations can charge users for the services they provide, in order to pay others for the resources they need. As long as they receive sufficient funds to operate on their own, they can thus subsist independently of any third party. If a decentralized organization is truly autonomous, no one (including its original creator) can control it after it has been deployed on the blockchain. That is actually a very threatening property: an organization could be conceived to evolve over time in order to adapt to the context in which it operates, thus it can be very difficult to know a priori if it can assume a dangerous state.

From a different point of view, the concept of decentralized autonomous organization can actually be linked to the development of the *Internet of Things* (IoT). The IoT will consist of billions of Internet-enabled devices, but not all can be blindly trustworthy and some could be even malicious. In order to facilitate private, secure, and (trustless) machine-to-machine coordination, these devices will need a central reference point. In this context the *ADePT (Autonomous Decentralized Peer-to-Peer Telemetry)* [17], by IBM and Samsung, is building a proof of concept system for the next generation of the IoT.

2.3.1 ADePT

The ADePT concept [17] uses blockchain technology to provide the backbone of the system, utilizing a mix of peer-to-peer protocols to get secure and fault-tolerant transactions. In ADePT, the blockchain is seen as a way for devices to understand what other devices do, and the instructions and permissions different users have around these devices. In practice this can mean tracking relationships between a user and a device, and even between two devices, with the consent of the user.

In order to realize the ADePT concept, IBM and Samsung chose three protocols:

- **Ethereum**, in order to allow devices to understand contracts and capabilities (this is where blockchain technology comes into play);
- **Telehash** [18], a private messaging protocol used to share information among two or more devices;

Symbol	Meaning	Symbol	Meaning
T	Transaction	M	Miner
I	Transaction input	R	Recipient (payee in cryptocurrency)
O	Transaction output	P	Transactor (payer in cryptocurrency)
B	Transaction block	N	Solution of mining (nonce)
H	Block header	τ	Difficulty of mining (threshold)
D	Digest	Ψ	Pseudo-random function
$\widehat{D}$	Root hash (aka root digest)	Π	Pricing function

Table 1: Special symbols used in the present work.

- **Bit Torrent** [19] a file sharing protocol used to move data around also in case of discontinuous and unreliable connections

According to [**?**], a Samsung W9000 washing machine was reconfigured to work within the ADePT system. When needed, the machine uses smart contracts to issue commands to a detergent retailer in order to receive new supplies. Thanks to these contracts the device has the ability to pay for the order itself, and later receive a notice from the retailer that the detergent has been paid for and shipped. Moreover, if a smartphone is connected to the home network, this information is broadcasted to it, presenting the purchase details to the owner of the washer.

3 Concepts and technologies

Since January 2009, when Bitcoin v0.1 was released and announced on a cryptography mailing list, there has been a growing interest on cryptocurrencies from various communities related in some way with Information Technology. In this section we will try to describe the main technical aspects at the basis of what we called "blockchain-based systems", at the current state of the art. Before going into details it can be useful to recap Bitcoin's basic strategy by sketching its main operations. This can serve as a guideline before our diving into the technicalities, and we hope it will induce the reader to focus on the functional requirements pursued by the designers of these systems, rather than on their actual implementations. Table 1 reports the main special symbols used throughout the following, alongside with their meaning.

3.1 Bitcoin quick overview

In the Bitcoin network, special nodes called *miners* collect transactions that are broadcasted over the network, and use their computing power to try to generate a valid block of transactions. The generation is done through repeated invocation of a hash function on data which reference the specific transactions that a miner decided to include in its block, together with the previous valid block and its own Bitcoin address. Then a public ledger, which consists in a chain of such blocks, is cooperatively and incrementally constructed by the miners over time, as follows:

- When a miner succeeds in generating a block, meaning that the hash of its block data is smaller than a given difficulty threshold, then it broadcasts such block to the network.
- In case other miners see that the above block is valid, and see that it is the longest extension of the blockchain that they are aware of, they move on and extend the blockchain from such block.
- The existence of the above block in the blockchain ratifies that the newly minted coins and the fees from the transactions included go to the public address that it provided. Only the miner having that address can redeem such coins and transaction fees, by using its corresponding private key.

At the same time, the synchronization among peers and the creation of new coins are managed by the system in a way that:

- The difficulty level readapts to the total hashing power of the miners. This is done by updating the hash threshold value every 2016 blocks, so that so many blocks get generated in about two weeks (i.e. each new block is generated every 10 minutes on average).
- The reward in newly minted coins started at 50 coins in January 2009 and halves every 210,000 blocks, i.e. about every 4 years (see above).

3.2 Transactions

At its core, a cryptocurrency is a chain of digital signatures reflecting the coin's paths through the system. A key concept in this context is that of *transaction*, which has an effect similar to a wire transfer between bank accounts. Actually it is a data structure which represents a single addressable unit of a *block*, which in turn is the single addressable unit of the blockchain. There are two different types of transactions: *standard* transactions, which encode one or more coin's transfers from some *transactor* node to one or more recipient nodes of the system, and *generation* transactions, where newly generated coins are granted by the system to miners, as a reward for their effort in building the blockchain.
Every amount of currency which results from a transaction is seen as a pack of coins which can be fractionated only through another transaction. Indeed a standard transaction does not record a single fund transfer from one point to another, but rather represents a scheme with multiple amounts which must be balanced: the number of coins - possibly collected from different packs - representing the requested amount for a given transaction must match the number of coins made available for the possible multiple recipients. In fact also the transactor itself can be considered the recipient of the eventual change, which represent a new smaller pack of coins.

This way parties can more easily keep track of paths followed by coins through the system, and monitor the integrity of the service. Actually, many systems include (optional) *transaction fees* for miners, so that the above balance is achieved by including these fees, even if their payee is not known beforehand. Indeed, a transfer for a transaction fee cannot be explicitly recorded in a standard transaction, since its recipient will be known to the system only afterward, resulting as the winner of a computational challenge (see §3.3). Therefore these fees must be included in the amount of the transaction, but they are not explicitly recorded in the transaction itself.

The specification of a transaction requires an *addressing scheme* so that transactors, recipients and miners (that is, ultimately, *each* node in the system) can be univocally identified. Moreover, transactions must be uniquely binded to transactors, in a way that - at least in theory - neither a party different from the transactor P can generate a transaction indicating P as transactor (*authenticity*), nor P can repudiate a transaction which it has previously generated (*non-repudiation*).
Transactions are processed locally by each node thanks to a programmable interface implemented in the peer-to-peer software. Since that interface makes use of a scripting language (e.g. [20, 9]), we will refer to the code for processing a transaction as the *transaction script*. The ultimate goal of a transaction script is allowing any other node in the system different from the transactor node to ascertain what are the conditions for the occurrence of transfers reported in the given transaction. That is accomplished through the signing and the verification procedures of a suitable digital signature scheme on some elements of the transaction.

These notions were first introduced for Bitcoin [3], and then adopted by many other cryptocurrency systems to date [21]. They can be easily adapted to perform transfers of any resource which admits a suitable, appropriate digital representation. Thus, by definition, we will assume that this approach specifies how resources are managed in any blockchain-based system.

3.2.1 Addressing

Since the validation of transactions relies upon digital signatures verified by system nodes, and nodes are implemented through a special software termed *wallet*, then there must be a mapping between wallets and (sets of) signature verification keys.
Signature verification keys are often linked to identifiers or pseudonyms valid only in the system context, called *addresses*, rather than real-life names and identities as in public-key certificate (PKC) based infrastructures [22]. This way, no trusted third party is required to ascertain the real identities of people and devices involved in the system; moreover, users can hope for some anonymity. *Users' anonimity* is a (controversial) feature aimed by many cryptocurrencies, and this is one reason for the possibility to have different user identifiers associated with the same wallet. Actually, verification keys could be used by

themselves as addresses in the system, since for their purpose they must be kept publicly available. However, this approach is inappropriate if non-volatile addresses are required, since key management imposes practices such as key *upgrading* and key *revocation* [23]. These considerations do not seem to have influenced the designers of Bitcoin and many other blockchain-based systems, where addresses are obtained as hash digests from verification keys, apparently at the sole scope of obtaining shorter identifiers [24]. For example, Bitcoin addresses are base58-encoded strings containing an address version number, the digest got by hashing an ECDSA [25] verification key with the RIPEMD-160 algorithm [26], and an error-detection checksum to catch typos [24].

3.2.2 Transaction inputs and outputs

As we told previously, the scope of a transaction is to keep track of the resource paths (both the granting of new resources and resource transfers) occurring among system nodes. In this way each node can verify the amount and addressing of the newly created resources, as well as the balance between resources requested for the transaction and resources transferred to recipients. In order to accomplish that, a transaction encodes those amounts as suitable data structures, called transaction inputs and outputs, respectively.

A transaction *input* composes of data indicating the source of the resources that a transactor P wishes to transfer, plus a scripting code whose processing should guarantee to every node in the system that P is actually authorized to redeem per se those resources.
Similarly, a transaction *output* composes of the indication of the destination where P intends to transfer those resources and their amount, plus a scripting code whose processing should guarantee that only the system nodes indicated by P as the recipients can actually redeem such resources.

Each input must match with an output related to a previous transaction which was validated (eventually among others) as a blockchain block. The matching between the current transaction input I and a previous transaction output O trasforms *all* the resources provided by O in resources available for I, so that to each input can correspond one and only one output. A different situation occurs inside a transaction, where zero to many inputs can be associated to one or more outpus. A no-input transaction stands for granting some newly created resources to one or more outputs, and corresponds to the case of a generation transaction in the context of cryptocurrecies. Otherwise, a single- or multi-input transaction records a transfer of resources from its input(s) to its output(s). The matching rule in this last case is as follows: one or more outputs O_{mn} must correspond to *each* input I_m, so that a certain amount of units of resource provided by I_m is equal to the sum of the amounts related to O_{mn}, net of transaction fees which cannot be indicated explicitly.

As result of the two above matching rules, resource paths start at outputs of generation transactions and from standard transaction inputs they branch into one or more ouputs, or from multiple transaction inputs they join into a single output. Figure 1 shows an example of a chain graph for resource transfers which

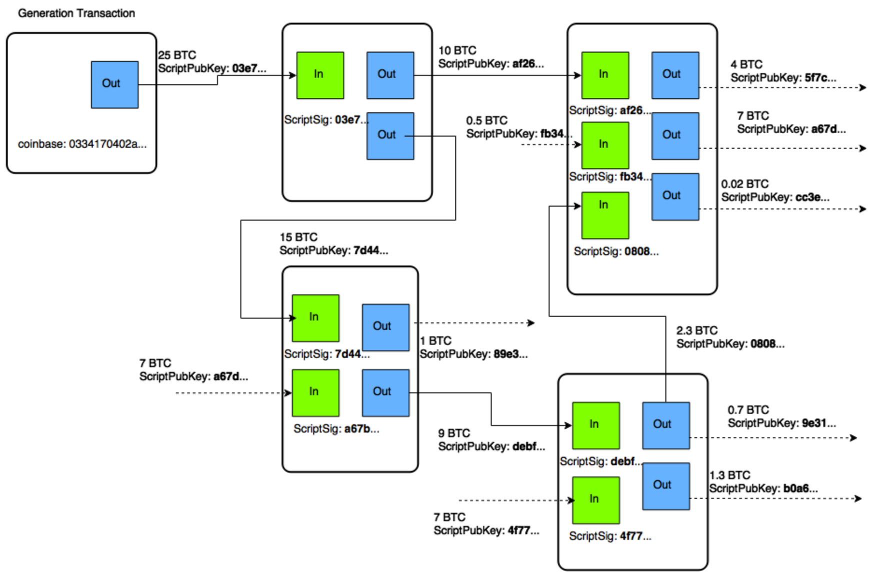

Figure 1: An example of transaction chaining in Bitcoin.

meets such rules.

In Bitcoin, each standard transaction – with its input(s)-output(s) matching – represents a transfer of a certain amount of *sathoshi*, which is the smallest possible unit of currency[1]. Each input I must reference a single output O from a transaction encoded in a previously validated transaction block, so that I must spend the amounts of satoshi in O. Schematically, given a transaction T, each input I and output O of T are defined as:

$$I := (\text{sha256}(T_p), i, \text{SgnCode}) \quad O := (v, \text{VrfCode}) \ , \tag{1}$$

where:

- sha256(T_p) is the SHA-256 digest of a previous transaction T_p. It is used to identify T_p as the transaction used to get input I;
- the *input index* i is a non negative integer which indicates the specific output in T_p used as an input in the current transaction T;
- the *signing script* SgnCode is a set of instructions and data provided by the transactor in order to satisfy the conditions placed in the pubkey script of output i in T_p. In its simplest form, such script just applies the transactor signature on the SHA-256 digest of all the I/O fields of the current transaction, with the exclusion of SgnCode itself;
- the *output value* v is the amount of satoshis associated with O;

[1]A satoshi is a one hundred millionth of a single bitcoin (BTC)

T: a44d97e0290a7979d2a9ff4b0ea4b009cf4d50b9f739693a73c0d7d6d297e650

```
{
  "hash": "a44d97e0290a7979d2a9ff4b0ea4b009cf4d50b9f739693a73c0d7d6d297e650",
  "ver": 1,
  "vin_sz": 1,
  "vout_sz": 2,
  "lock_time": 0,
  "size": 223,
  "in": [
    {
      "prev_out": {
        "hash": "98ad911be7ff50439110e402745f49bb42e9fdc3b722198b96fd442915392ed6",
        "n": 1
      },
      "scriptSig": "30440220362feb8862be8e89442856cc69ac6e8094a2260a76e34fef2c7cd99\
                   c5f41bbed0220070bde5c073b63427fa211261da8bd5343d31fdeef5a885ed6\
                   74e816e4cf463a01033c586247a1abbbf1f3895adaee521b20cb 5e96948278\
                   1d0c71dadd8c0d26fad1"
    }
  ],
  "out": [
    {
      "value": "608.90406349",
      "scriptPubKey": "OP_DUP OP_HASH160 fa6b6e87bd2a264e001ce6ddfce39499d269ade2\
                      OP_EQUALVERIFY OP_CHECKSIG",
      "address": "1Pq6dSvv4UoJ1nC8L2TgWyKzVUMfeCHpPK"
    },
    {
      "value": "0.18040121",
      "scriptPubKey": "OP_HASH160 f5c9e9ed35bb7405bbd4c1f3c3aeeb17232f663b OP_EQUAL"
    }
  ],
  "nid": "934d7bc180e7c88699e89a65b5ce50022400a4ce8dc7c66adfd658ccf95de985",
  "block": "0000000000000000ca23d66664e258fff1293987d1302b8d6e5dcee26b92cd5",
  "blocknumber": 380336,
  "time": "2015-10-24 12:53:20"
}
```

Figure 2: An example of Bitcoin standard transaction.

- the *pubkey script* VrfCode is a set of instructions and data provided by the transactor that specifies the recipient(s) authorized to collect the value v in satoshi associated with O. The most used VrfCode returns true if matched with a signing script consisting in a signature of a (subsequent) transaction (see §3.2.3).

An example of Bitcoin standard transaction in row encoding format is given in Figure 2. The transaction, identified by the SHA-256 hash indicated at the top of the frame and listed in line 2 of the code, takes a single input I (coded in the `"in"` section) and converts it in the two outputs O listed in section `"out"`. I is identified by the `hash` and `n` fields given in lines 11-12, which correspond to the SHA-256 digest and index of a previous Bitcoin transaction. The first output transfers a `value` of $v = 60890406349$ satoshi to the Bitcoin address given by `1Pq6dSvv4UoJ1nC8L2TgWyKzVUMfeCHpPK`, whilst the second output redeems $v = 18040121$ satoshi to the transactor. Some of the lines of code shown in Figure 2 are not recorded as such in the blockchain, but they are rather the result of some processing by a *blockchain browser*, a web tool designed to offer human-readable views of a blockchain (e.g. [27, 28]). For example, values are expressed in BTC in the code of Figure 2, whilst they are recorded in satoshi in the blockchain. Moreover, the field `blocknumber` is not actually recorded in the blockchain [29].

3.2.3 Transaction processing

In order to be validated without the intervention of a trusted third party, transactions require some processing at the peer nodes to ascertain with corroborate evidence: their origin (i.e. the identities of their transactors), the fact that the resources deemed to be transacted are effectively possessed by the transactors, and – last but not least – that only the recipients satisfying the conditions required by transactors are accounted for such resources.
A key point in Bitcoin, and more generally in blockchain-based systems, is that transactions include code that can be customized by the transactor in order to allow for different kinds of verification procedures.

To verify that inputs are authorized to collect the values of referenced outputs, Bitcoin uses a scripting system which resembles to Forth []. With reference to Figure 2, the input's SgnCode (denoted as `scriptSig` in the code) and the referenced output's VrfCode (`scriptPubKey`) are evaluated, and the input is authorized only if `scriptPubKey` returns true. Through the scripting system, the transactor can create many different conditions that peers have to meet in order to claim the output's value. For example, it's possible to create outputs that can be claimed by anyone without any authorization, or that require multiple signatures, or that can be redeemable with a password instead of a key.

A similar approach is used in the other blockchain-based systems to date, although some differences exist in the scripting system used, and in the way transaction processing takes place.
Particularly relevant seems in this respect Ethereum (see §2.1), which uses its own digital coin, called *ether*, to "fuel" the execution of applications stored on its blockchain. *Fueling* is designed as a way to pay for the execution of Ethereum bytecode and storage of data on the blockchain. How much a specific computation will cost is defined by the complexity of the computation in term of its basic operations, as follows. Basic operations such as addition, subtraction and multiplication cost one "unit of gas", as indicated by the parameter `GASPRICE`; at the moment of writing $\texttt{GASPRICE} = 10^{-6}$ ether. The *Ethereum virtual machine (EVM)*, after the compilation of the code related to a given transaction, computes the amount `NUMBAOPS` of its basic operations and fixes the total cost for the transaction as:

$$\texttt{TRANCOST} = \texttt{NUMBAOPS} * \texttt{GASPRICE} + \texttt{STARTGAS} \ ,$$

where `STARTGAS` is the initial value of (units of) gas at disposal of the transactor. Coherently with the above approach, a transaction on the Ethereum blockchain contains six fields. Three such fields have the same meaning than in Bitcoin: they are the recipient of the transaction (be it a user or a smart contract), the

signature of the transactor, and the amount of ether to send with the transaction. The three other fields are the `STARTGAS` and `GASPRICE` values, plus an optional data field.
Notice that each transaction is limited on the number of computational steps for the code execution it can generate, since those steps are proportional to the transaction cost in ether. If a transaction execution "runs out of gas", then all state changes revert except for the payment of the fees; conversely, if the execution halts with some gas remaining, then the remaining portion of the fees is refunded to the transactor.
One of the high-level programming language used to write Ethereum contracts is termed *Serpent* [8], which was designed to be very similar to Python. Unlike the Bitcoin scripting language Script, Serpent is *Turing-complete* [30]; infinite loops in code can indeed be avoided thanks to the fueling mechanism.

3.3 The blockchain technology

As we have seen in the previous sections, blockchain-based systems use digital signatures in order to guarantee the proper attribution of each transfer of resources. However, one main concern for such systems is that of protecting a chain of transfers like that depicted in Figure 1 from tampering. The problem arises since such chains are built incrementally by a network of peer nodes, without the support of any trusted third party. Actually, neither miners nor transactors can be trusted, since each of them could cheat in order to get more resources or to avoid that other parties get the resources they deserve.
Suppose, just to fix ideas, that the system goal is to manage payments for the purchase of goods within a set of parties, where each party can play both the roles of buyer and seller[2]. Then, obviously, the sellers want to be assured that payments that were agreed with the buyers – as recorded in the transactions – cannot be declined by the buyers themselves at a later time, when goods have already been shipped. One main question arises in this respect: can a chain of digital signatures protect sellers from buyers who cheat?
The answer is affermative in theory, even if this solution is not currently implemented. Indeed a seller R can protect herself from attemps to alter a transaction T_P in her favor if she creates a subsequent transaction T_R which redeems per se the money provided by T_P. Therefore, the only way for P to alter T_P is to collude with R, since only R can generate the input of T_R, and such input must match with the output of T_P.
Thus, the chains of standard transactions illustrated in §3.2 can protect against the so-called *double-spending problem*, where malicious users try to transact some resources (the same digital coins, in the context of cryptocurrencies) twice or more.

Things get more complicated if the system has to provide for the creation of new resources (e.g., the coining of money): to which node(s) should in this case

[2]That is, actually, similar to the application scenario for cryptocurrencies, but without the issue of coining new currency (i.e. we are assuming that the system only manages money transfers)

the new resources be granted? And how to account such grants in the system?

The idea in Bitcoin was to assign the task of assessing the correct matching of standard transactions to special nodes called *miners*, and to reward miners for their work with some (optional) transaction fees plus new generated coins. Thus, each miner must associate a generation transaction to the processed standard transactions. Hence standard transactions are grouped together so that they refer to the same generation transaction, thus the miner could eventually be awarded with newly generated coins and relative transaction fees (see §3.2). Moreover, it is better to group several standard transactions together, since the network overhead could eventually grow in an unfeasible way if many transactions ensues.
Following this idea, the "unit of mining" becomes the *transaction block*, a data structure grouping one or many standard transactions (by one or more transactors) with a single generation transaction. This is where the blockchain technology comes into play.

To put it simply, the blockchain technology consists of a data structure (the *blockchain*), plus a protocol for managing the creation of new transaction blocks. The goal pursued with this technology is to have a public ledger of transactions built over time thanks to the miners, and whose consistency and authenticity can be monitored by any node in the peer-to-peer system, without any central authority.

However, the shift from single transactions to transaction blocks exposes again the system to double-spending problems. Indeed, in the context of an organization of recorded data in transaction blocks, chains of transfer paths cannot be easily managed and validated through digital signatures as described in §3.2. To overcome this difficulty, the designer(s) of Bitcoin introduced a new chaining mechanism, in this case between different blocks.
Roughly speaking, it consists in inserting a cryptographic hash digest related to the preceding block in the current block. Since a cryptographic hash digest is supposed to correspond univocally to its originating data with overhelming probability, then the above approach results in the circumstance that any tampering in a given block involves changes in all its subsequent blocks. This way, in order to achieve its purpose, a malicious miner has to be faster than the honest miners in mining transaction blocks. Therefore the double-spending threat is mitigated by making the mining of a transaction block a costly operation.

Overall, the management of the blockchain - that is, the collection of new transactions and their insertion upon validation in the chain of previous blocks - is a process composed of the following steps:

1. Collecting incoming transactions into new blocks;
2. Mining of new blocks;
3. Blockchain updating.

All the above steps are performed by miners thanks to suitable client software. This software enforces protocol executions using cryptographic mechanisms as

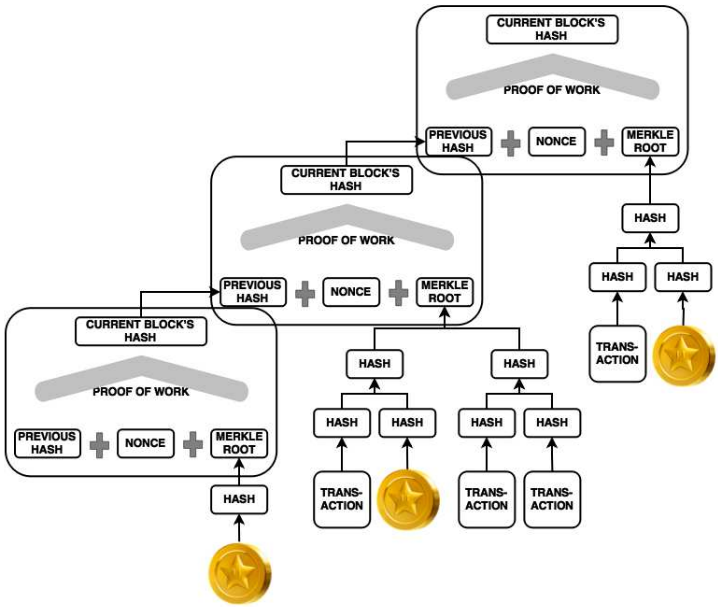

Figure 3: Collecting new transactions into blocks.

preventive and detective controls, and implements the decision-making process required at step 3 through a kind of consensus expressed by the nodes involved in the system. The following sections discuss each step in detail. The core protocol for the construction of a blockchain is illustrated in Figure 3.

3.4 Collecting new transactions into blocks

It's a miner task to collect new transactions that are taking place on the network in the same time , and to assemble these transactions into a transaction block. In Bitcoin and other cryptocurrencies, miners are totally free to choose which transactions to include in their blocks. Most miners will consider any transaction that reaches them over the network, assuming it includes an appropriate fee, but nothing forces their choice. This freedom is intended to get resilience and performance of the system in case of network delays or faults, since new transaction broadcasts do not necessarily need to reach all miners, nor a miner has to wait for a minimum of transactions in order to start on working on a new block [3, 24]. Actually, it could be also the case that a miner with malevolent intentions will decide to mine a block composed of just the generation transaction, with the purpose of having a greater chance of getting that block inserted in the blockchain. As we will see later, there are very few deterrents to a such bad behavior, that would eventually result in lack of faith in the system (see §3.6).

There is, however, an upper limit to the size of a transaction block, which is currently 1MB in the Bitcoin system [29]. Such limit was chosen as a tradeoff between constraints on peers resources and the throughput of the transaction management: a bigger limit would result in less nodes being able to mine or validate a full-size block, whilst a smaller limit would imply less transaction throughput [31].

In Bitcoin, the standard transactions $T_1, \ldots, T_h$ that a miner M chooses to assemble into the current mining block B_c – alongside with the generation transaction T_0 – are processed by M in pairs[3], in order to calculate a unique hash digest $\widehat{D}_c$ for all of them (*root hash*) thanks to a *Merkle tree* [32]. The hash function used to generate the Merkle tree is SHA-256 [33], thus $\widehat{D}_c$ is a 256-bit string. The root hash is then recorded in the so-called *block header* H_c of B_c, alongside with the digest D_p of the block header H_p of the previous (valid) block B_p , where D_p is computed by applying twice SHA-256 to H_p. Besides the 256-bit fields provided for storing D_p and $\widehat{D}_c$, a block header consists of the 32-bit fields `version`, `time`, `bits` and `nonce` for a total of 80 bytes. The first three fields store the block version number, the current time in seconds since 1970-01-01T00:00 UTC and a string encoding the difficulty of the proof-of-work, respectively. Instead, the `nonce` field is used to store (part of) a string representing the solution given by M to the proof-of-work (see §3.5) for B_c [34, 35].

Notice that if a miner generates a block with one or more invalid transactions, whatever the error and the intentions are, then the other (honest) miners will not accept the new block. They will ignore it and continue trying to build a block on top of the last block they think is valid.

3.5 Mining of new blocks

In one form or another, *mining* is at the heart of any current blockchain-based system. Mining is indeed the way the system imposes a computational cost for the recording of new transaction blocks into the blockchain, so to mitigate the double-spending threat. An important point to stress here, since often misunderstood, is that the goal of a mining protocol does not actually consist in the determination of the miner in charge of block recording, but rather of the set of miners that are eligible for recording a new block in the blockchain (see §3.6).

The mining protocol chosen in Bitcoin is similar to the (*partial inversion*) *proof-of-work* as described in [36], even if in this meaning it has new characteristics and purpose.
In a proof-of-work, a prover demonstrates to a verifier that she has performed a computational work in order to gain access to a resource. The original application for this type of protocols was in the context of being able to deter spam email, and the main idea was to require the prover to solve a moderately hard but easy to verify computational problem tied to a particular email message [37]. That is conceptually like affixing a postage stamp to a message: for a

[3]If $h > 0$ and $h + 1$ is odd, then T_h is considered twice.

legitimate sender who is only sending out a small number of messages such type of proof will not amount to very much computational effort, but for a spammer it might be prohibitively expensive.
The proof-of-work implemented in Bitcoin consists in the search for a peculiar hash digest D, calculated applying two subsequent SHA-256 to H, such that

$$D = \text{sha256}(\text{sha256}(H)) \leq \tau \ , \tag{2}$$

where τ is fixed by the system in a way that we will see later in this discussion, and H is the header of the block of transactions collected by miner M. As seen in §3.4, H consists of the same kind of double-hash digest D_p of the previous valid block header H_p, the root hash $\widehat{D}$ of the transactions collected by M in the current working block B, the `nonce` field, and other minor fields. The `nonce` field is 32-bit long and is used for storing (the first part of) a string N which the miner has to find in order to satisfy condition (2).
With the introduction of special purpose hashing hardware (see §4.1) the `nonce` field in the header has become too small, and the workaround was to have an `extraNonce` field in a generation transaction so that a solution N has to be found in a much larger string space. This makes the proof-of-work more costly and involved, since M must recompute the root digest $\widehat{D}$ each time it changes the `extraNonce` string [35].
The solution N of the proof-of-work is called *nonce*; the term was inspired by [38] and refers to the fact that, because of the properties of a cryptographic hash function, N behaves as a pseudo-random string as one or more of the proof-of-work's parameters change.
A miner M to succeed in the search for a D satisfying (2) will write N in the `nonce` and `extraNonce` fields of its working block B, and will broadcast B to the other peers for verification. The other miners in the system express their acceptance of the block by starting to work on the creation of a next block in the chain, therefore they use the digest D of the accepted block as one component of the block header of the new working block, as previously described.

Now the point is to understand how τ is fixed. Practically, the goal of finding a $D < \tau$ corresponds to the goal of finding a D such that it has a certain number of zeroes as first Hex digits. A SHA-256 digest is always 32 bytes or 64 hexadecimals long, so the probabilistic difficulty of the goal increases as this number of Hex zero digits increases, and therefore it increases inversely as τ decreases.
For example, if we need digests beginning with fifteen zeroes like:

$$0000000000000001b6b9a13b095e96db41c4a928b97ef2d944a9b31b2cc7bdc4,$$

then τ will be the integer number whose Hex representation is given by:

$$0000000000000010000000000000000000000000000000000000000000000000.$$

More generally, if we need digests having z leading zeros, then τ will be the integer coded by the Hex string of all zeros except the digit "1" in the z-th position.

As we are going to better explain in §4, Bitcoin's proof-of-work is a way to get a valid and reliable transaction processing by ensuring that a set of peer nodes (the miners) compete in solving a computational challenge within a preset time. However, the problem to be solved is a kind of artificial puzzle of no practical concern (besides, of course, the existence of the network), which requires an increasing amount of computational resources over time, and where miners' fitness is proportional to their hashing rate. In an attempt to overcome these limitations, some alternatives for implementing a requirement like (2) have been proposed over time, concerning both the underlying function used to compute D and the way τ is chosen. These alternatives were introduced mainly in an attempt to mitigate threats deriving from accumulators of hashing power. A single party, or a *pool* of them [4] could indeed invest in hardware equipment to obtain a substantial percentage of the total hashing power, and with such power they could succeed in double-spending by reversing the recent blockchain history. Or alternatively they could carry out a denial-of-service by refusing to include transactions in the block they generate, unless perhaps the transactions useful to their own purposes.
Although only a percentage greater than 50% of the total hashpower allows an attacker to get the mathematical certainty in subverting sooner or later the system [3], also attackers with a substantial minority of the total hashpower can plausibly attempt double-spending and denial-of-service attacks [3, 39].

Litecoin [40] makes use of a variant of Bitcoin's proof-of-work that employs sCrypt [41] instead of SHA-256. sCrypt was originally designed as an alternative to the iterative use of pseudo-random functions like MD5 [42] or DES [43] for deriving a key from a password/passphrase. Indeed sCrypt's goal was to implement a pseudo-random function Ψ which is also *sequential memory-hard*, i.e. such that:

- the fastest sequential algorithm for computing Ψ requires an amount of memory roughly proportional to the number of operations to be performed;
- it is impossible for a parallel algorithm to asymptotically achieve a significantly lower cost in terms of both \$ and *secs* than the fastest sequential algorithm.

This feature is used in Litecoin and a bunch of other cryptocurrencies [21] to

[4] Since the threshold τ exponentially decreases over time, with the consequent introduction of the `extraNonce` field in generation transactions, mining has become a very difficult problem which at current time is typically solved by mining pools, where a bunch of miners share work and rewards. Mining pools use higher thresholds than the target threshold τ to see how much work miners are doing. For example, if mining requires a digest having 15 leading zeros, then the mining pool can ask for digests having (at least) 10 leading zeros. Each partial solution proves that the miner is working hard on the problem, and gives the miner a share in the final reward, if any. Eventually one of these partial solutions could get the target number of leading zeros, therefore successfully mining the block and winning the reward for the pool. The reward is then split basing on the fraction of total shares that each miner counts, and the pool operator takes a small percentage for her work [35].

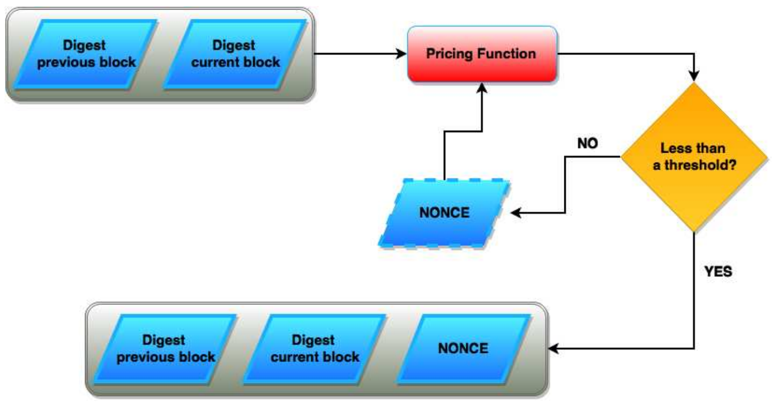

Figure 4: Given the digests of the current and previous blocks, the act of mining the current block is a process which consists in finding a nonce so that the value returned by a suitable pricing function is less than a parametrizable threshold. The output consisting in the association between the digests and the nonce actually represents what is indicated with "proof-of-work" or similar terms (e.g. "proof-of-stake",) in the literature.

hinder the use of specialized hardware, thus avoiding the side effects of the rapid shift in mining technologies observed for Bitcoin, as described in §4.

Cryptocurrencies that do not make use of Bitcoin's proof-of-work - or its variants - commonly rely on the *proof-of-stake* approach to mining [44]. These are mechanisms that extend the chances in mining new blocks to the stake-holders of the system, with the aim to get a counterweight to hashpower. In a proof-of-stake mechanism the threshold τ is indeed proportional to the number of coins owned by the miner at current time, meaning that a miner which owns e.g. 100 coins is 10 times more likely to create the new block than a miner which owns 10 coins. By itself this would result in "coin pile issue" similar to the hashing power trouble that a proof-of-stake is intended to mitigate, thus these mechanisms also include some other parameters to compensate such bad behaviour. For example, in the PeerCoin system τ is proportional to *coin age*, a number derived from the product of the number of coins owned times the number of days the coins have been held. An important difference here is that the hashing is timed in one digest calculation per second, and not more frequently; moreover it is possible to calculate it only once for each unspent coin in the wallet [44]. In this way miners are not motivated in accumulating hashing power. However, this comes at the price of a (loose) time synchronization among nodes, which was presumably avoided in Bitcoin to overcome the difficulties of getting a trusted "global clock" without a *Time-Stamping Authority* [45].

We conclude this section by osserving that, at least at our knowledge, all the mining approaches proposed so far can be described as particular instances of the workflow depicted in Figure 4, here described.

Pricing functions were introduced in [37], where a such function $\Pi : X \times Y \to \mathbb{R}$ can be informally defined as satisfying the following properties:

1. Given $x \in X$ and $t \in \mathbb{R}$, it is moderately hard to find $y \in Y \ : \Pi(x, y) \leq t$;
2. for any $(x, y) \in X \times Y$ it is easy to compute $\Pi(x, y)$;
3. Π is not amenable to amortization: for any finite set $x_1, \ldots, x_m \in X$ of input values and corresponding threshold values $\tau_1, \ldots, \tau_m \in \mathbb{R}$, the cost of finding $y_1, \ldots, y_m \in Y$ with $\Pi(x_i, y_i) \leq t_i$ on the whole set is comparable to the total cost of finding $y_i : \Pi(x_i, y_i) \leq t_i$ from scratch for each (x_i, t_i);

Requirement (3) means that there are no "shortcuts" in finding $y : \Pi(x, y) \leq t$ for multiple inputs and thresholds, also if these inputs and thresholds are strictly related. In other words, the solution y of the search problem related to Π behaves like a random variable in Y.

The sense of the workflow of Figure 4 is as follows: through a suitable pricing function Π and a tunable difficulty threshold τ, a miner node M can give corroborate evidence to its peers that it found a solution to a moderately hard computational problem, and that its effort involved processing of the previous block in the blockchain alongside with the current transactions.

3.6 Blockchain updating

As we have described in §3.2.2 and illustrated through Figure 1, the chaining of a single transaction results in a quite complex graph, where a single input may branch in multiple transaction outputs, and multiple inputs may merge in a single transaction output. Things go better when transactions are assembled in blocks and these blocks are chained together to form the blockchain (see §§3.4, 3.5). The blockchain has indeed a tree data structure: the genesis block is its root, each block is a child of the block it references, and it may be the case that a given block has more than one child, thus resulting in one or more branches in the chain. That sounds odd, since a single coherent history of transactions is desired, and each branch represents one different version of such history. However, it is consequence of the approach chosen to add a new block in current blockchain-based systems. Following [3], indeed, all these systems adopt an implicitly-defined majority decision, consisting in the fact that (honest) miners will spend their next mining work on the "longest chain"[5], that is the chain of blocks with the most mining effort in it. Such effort is measured as the sum of the difficulties that were required to mine all the blocks composing the chain [46]. In proof-of-work based systems, for example, the longest chain is the one with the longest string got by concatenating the "leading zeros" of all the proofs of work that were required to build the chain (see §3.5).

In order to better understand the process of updating a blockchain, it may be worth pointing out here the following circumstances.

[5]This is a frequently-used but misleading term, as it will be clear from the sequel.

- The majority decision mechanism described above is not mandatory, and the choice of the block to be referenced in the chain is actually left to each single miner. A miner M is just encouraged to adehere to the mechanism, on the basis of the fact that the longer is the chain linked to M's new mined block B_M, the more are the chances that M gets a reward for the mining activity spent on B_M. This design was presumably choosen in [3] because it is really difficult to guarantee and ascertain the correctness of peers local processing, but – as we are going to discuss shortly – it is the root cause of the double-spending threat.
- The "longest chain" is not actually unique. Indeed, it may be the case that the same mining effort (e.g. the total number of "leading zeros") corresponds to two or more chains that differs for some block and/or the block order. Thus, also assuming that all the miners follow the majority decision mechanism, branches are possible.
- Each branch must be internally consistent and can never include two conflicting transactions; however, branches do not need to be consistent with one another, and one branch can include a transaction which contradicts a transaction in another branch.
- The blocks that will be incorporated in the blockchain are determined by the upshot of a sequence of mining activities and, ultimately, by the relative computing powers of all miners working at that section of the blockchain. As we will detail in §4, at least in the case of Bitcoin, this has given rise to a rude mining-hardware race which undermines the confidence of users in the cryptocurrency and its stability on the market.

Branching in a blockchain may result from the fact that different, honest miners are in temporary disagreement about which chain should be considered valid, as they found out different "longest chain". That could happen, for example, because of network latencies or network outages.
However, secretly minining a branch is at the core of the double-spending attack. Indeed, in its essence a successful such attack consists in what follows. The attacker P broadcasts to the network a transaction T_P in favour of the seller R. Then P mines a block which builds on the latest block B_l before the one that eventually will contain T_P, and which contains a *conflicting* transaction $\tilde{T}_P$, i.e. a transaction which pays the amount of T_P to another party than R (maybe P herself). P waits until R is confident in her payment and sends her the merchandise. In the meantime, P uses the secretly mined block to extend the chain in order to get a branch longer than the one known by the network. Finally, she release the secret branch to the network, thus overriding the payment in favour of R.

A main concept concerning the confidence that a seller can put in a payment in his favour is that of *confirmations of a transaction*. A transaction is said to have c confirmations if it is included in a block which is part of the valid chain, and there are c blocks in the path from that block to the leaf of the chain, inclusive [47]. Intuitively, the more confirmations a transaction has the more likely that it will remain in the chain permanently. Nakamoto's paper [3] outlines the number of confirmations versus the relative hashpower of an attacker P, such that the probability of success of P in a double-spending attack is less than 0.1%. From those calculations, it follows that if P has less than 10% of the total hashing power of the network, then $c = 6$ confirmations are sufficient. However, if P has a greater percentage of the hashing power, then an increasingly bigger number c of confirmations is required. For example, if P has 30% of the total hashing power, then $c = 24$ confirmations are required in order to limit the chances of a double-spending attack to less than 1 in 1000. If P has more than 50% of the total hashpower, then sooner or later she will inevitably succeed in a double-spending attack; this is know as the "51% attack".

4 Computational aspects

One important feature in the consensus mechanism implemented in blockchain-based systems is the unpredictability of the miner in charge of recording the next transaction block. First and foremost, this is useful to prevent frauds: with some degree of certainty, and for every single recording, no one should know in advance who would be the peer entitled to record the transaction. If that happened, a malicious user could do a denial-of-service attack against such peer to defraud her of the prize.
At the same time, that unpredictability results in an incentive for peers in doing mining, which is vital for a blockchain-based system. Although there are nodes that have some advantage with respect to others in finding the nonce, which ultimately depends on the specific mining approach in use, even the less fitted node has some chance of finding it (see §4.1).

Another important feature is the time-lapse mechanism implemented through the average time that the miners as a whole spend in mining a new block. Indeed, such average time is somewhat predetermined in each blockchain-based system, and this turns out in a synchronization mechanism for updating the blockchain on every single node in the system.

With this design in mind, and after fixing a specific mining approach, we want to discuss the following two issues: (i) the measure of performance in mining, and (ii) the way the system tunes the difficulty of mining in order to get the preset time-lapse for the blockchain update. In §4.1 we will analyze these two concepts to understand how computation is involved in the Bitcoin blockchain implementation.

4.1 Difficulty and performance in Bitcoin

The average time that the miners as a whole spend in searching the nonce N that solves (2) before succeeding was predetermined by the Bitcoin creators as about $t = 10$ minutes. This means that the system should create a new block in the blockchain after spending 10 minutes on average in searching for a correct

digest, and at the same time establishing (almost randomly) the peer entitled for the block recording.
Since this is a *trial and error* algorithm, the more peers in the system participate to the race with a fixed τ, the shorter is the average time to find a correct solution. But we know that $t = 10$ minutes is the predetermined time that the system should take to find a solution, and on average it should be met. This looks like a *soft real-time* system [48], but we think it is not, as the time expected to have a result should not significantly exceed 10 minutes, but it should neither be less than that.
When actual $t < 10$ repeatedly for a number of blocks, then the system increases the difficulty to make the interval longer, that means the threshold τ in (2) must be lower for the next blocks. As we told in §3.5, lowering the threshold of one single level in Bitcoin corresponds to add one leading zero to the Hex string encoding the previous threshold value or, equivalently, to shift its single "1" digit from position z to $z + 1$. In turn, it can be shown that – at least under the assumption that the hash function is *well balanced*[6] [49] and $z \geq 10$ – the above shift results in a proof of work which is probabilistically harder than the previous one by a factor very close to $2^{16} = 65536$.

In Figure 5 we can see how the implemented difficulty increased over the last two years. This is referable to two distinct facts:

- the growth in the number of competitors to the race, due to the diffusion of the currency and to the prize offered;
- the growth of the performance of the single computational resource, due to the usage of better hardware.

But what is the parameter used for performance measuring in this context? The SHA-256 hash function does not need all the features available in a complex Instruction Set Architecture, as the ones equipped in general purpose computers. Therefore a new appropriate parameter for evaluating the performance of an hashing system has been introduced: the *hash-rate*, currently measured in GigaHash per second (GH/s).
In Figure 6 we can notice that the global hash-rate of Bitcoin is oscillating in a daily order, depending mainly on the number of participating clients in the global competition, while the trending growth is due to improving computing systems.
But where does this improvement lie in the Bitcoin mining systems?
As we have shown, the solution to our problem can be found through a trial and error algorithm. This immediately leads to the probabilistic observation that the more calculating units the client has, the better chances to win the race.
In fact, in the short lifetime of this cryptocurrency, we have seen a shift in terms of technologies used for the computing resources in the mining competition, from single core to multicores, then GPUs, and FPGAs, to end up with ASICs.

[6]Roughly speaking, in a well balanced hash function each output has about the same number of pre-images.

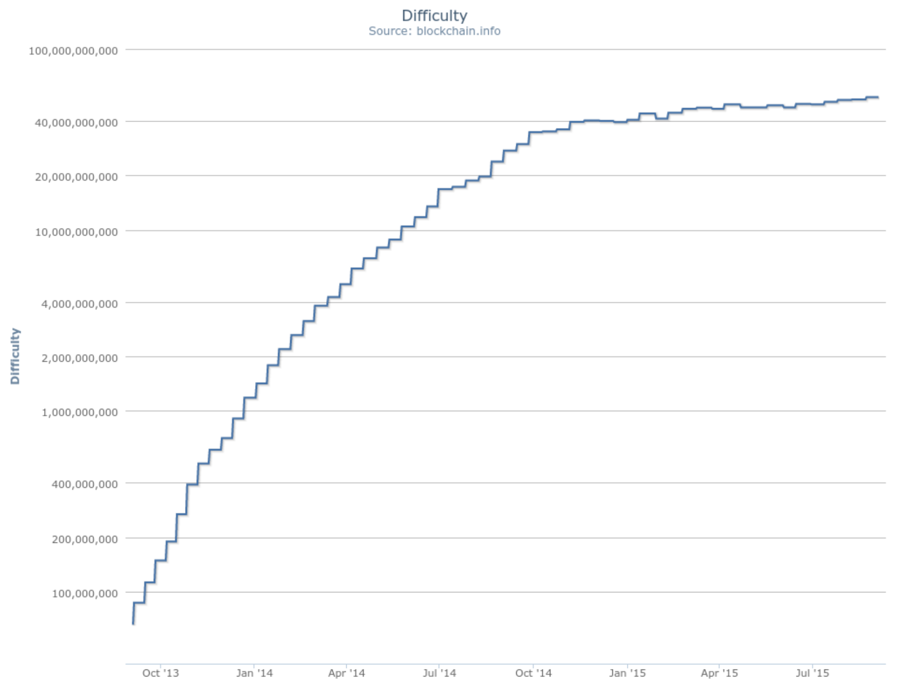

Figure 5: Difficulty of Bitcoin mining in the last two years. It represent the difficulty to find a nonce for a new block when compared to the easiest it can ever be.

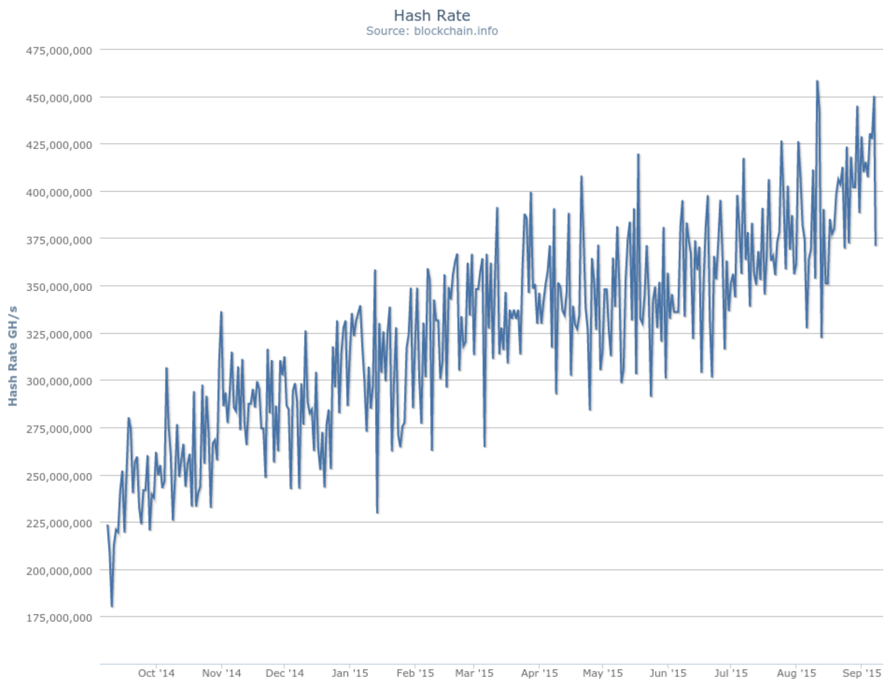

Figure 6: Estimated global hash-rate of the Bitcoin system in the last two years.

Technology	Device	Mhash/s	W	$	Mhash/s/$	Mhash/J
CPU - Intel	Core i7 3930k	66.6	130	670	0.10	0.51
CPU - AMD	4x Opteron 6174	115	320	220	0.52	0.36
Coprocessor Intel	Xeon Phi 5100	140	225	4000	0.03	0.62
GPU - Nvidia	Tesla S2070	749.23	900	3000	0.25	0.83
GPU - AMD (ATI)	5870x6	2568	1200	300	8.56	2.14
FPGA	Butterflylabs Mini Rig	25200	1250	15295	1.64	20.16
ASIC	AntMiner S5	1155000	590	370	3121	1957
ASIC	Spondooliestech SP35 Yukon	5500000	3650	2235	2460	1506

Table 2: Illustrative comparison of different hashing technologies [50, 51]. Mhash/s stands for million hashes per second. W is for watt. $ is the estimated cost at the time when the device has been put on the market. Mhash/s/$ is the performance per dollar. Mhash/J is the throughput per watt.

This is the effect of the quest for the correct solution, or - using a nostalgic pleasant term - the Nonce Rush. Anything looking like something useless is sacrificed in the name of the hash-rate. Multicores had a short life since they were not efficient horse carriages. GPUs had a fancy number of lightweight cores working well (with differences due to the hardware implementation of the instructions) but soon they lost appeal. In fact all these technologies seemed like wasted money since someone had the idea to build specific hardware to compute uniquely the double SHA-256 hash function required to check (2). It resulted to be very fast with FPGAs, and incredibly fast with the introduction of ASICs.

As we can see in Table 2, the convenience of using specialized technologies does not lie just in the computing speed, but also in device cost and power consumption. In fact we have to remark again that Bitcoin mining is a computational race based on casualty and attempts. The key to win virtual coins - which are actually usable to buy real goods - is to have the fastest and overall less expensive computing units. Spending thousands dollars without improving the chances to win the race is a terrible investment.
At the same time, if Ms. Smith would decide to make a good investment today by buying the fastest ASIC available, the repayment plan could not be successfully accomplished: in fact a new technology could arise and a certain number of investors could decide to rely on it, making Ms. Smith hardware obsolete and unfit to win any competition.

4.2 Making it useful

While some supporter of Bitcoin argue that the costs of the current mining system is a fraction of the costs to maintain the real material currencies, some alternative initiatives are spreading.
The underlying idea consists in implementing a proof-of-work system while doing some useful calculations.

An example of such a different proof-of-work algorithm is proposed in Primecoin [52], where the computing time is used to find prime numbers chains known as Cunningham chains and bi-twin chains, both of high scientific interest. This proof-of-work has the property of being efficiently verifiable, through the Fermat test [53] of base 2 together with Euler-Lagrange-Lifchitz test [54], which verify probable primality for the prime chains. Also difficulty can be implemented using the remainder of Fermat test, so that an approximately linear continuous difficulty curve can be constructed for a given prime chain length.
Some developers proposed to use distributed computing tools like folding@home or SETI@home for the currency mining, so that the user computations could be somewhat useful in the cure of protein misfolding diseases or in the search for signals of alien intelligence in the universe.
However the point is that a consensus-based blockchain needs an efficiently verifiable proof-of-work, like SHA-256, but folding@home and SETI@home do not have such a property. Since now, the participants in the protein folding and SETI networks were volunteers, with probably no intentions other than actually help the goal of the project. If tied to a currency mining, these networks could be weakened by participants searching for profits, not bothered with the actual computations. Instead those miners could be prone to provide fake data that has no value to the humanitarian goals, but that is likely indistinguishable from a genuine output.
Actually there are working implementations of cryptocurrency connected to a protein folding network (CureCoin [55]) and to the BOINC distributed scientific computing network (Gridcoin [56]). In these currencies the research networks are not used for a decentralized consensus, but rather interconnected to another proof-of-work or proof-of-stake system to allot a credit based on the research work done. Let's take CureCoin: in this system the security of the currency is entrusted by nodes running SHA-256 like in Bitcoin, but there is a percentage of the daily mined CureCoins that are granted to the foldinghome participants in the system. Indeed those involved in the research job get 76% of the total coins, while blockchain maintainers get another 19%. Finally there is also a sort of *development tax*: 2% of the total mined coins are distributed to people who donated to project development, while the remaining 3% is reserved to Curecoin developers, and will be used for compensating development costs and for community care. Let us say this would be a fair reward scheme, but it is significantly more centralized than the original Bitcoin one, and probably less resistant to fraud attacks.
A different approach is used in Gridcoin: the consensus is granted with a proof-of-stake, but the amount credited is based on the effective work done in the BOINC network. The interesting point here is that the mining mechanism involves some concepts from Bitcoin, like *Network Average* which is similar to difficulty in the proof-of-work, but also implements a complex rewarding mechanism which adds layers of interactions between servers and clients. Its finality would be the fairness of rewarding with some degree of fraud resistance, but with several actors involved, the system could overall expose more weaknesses.
Generally speaking, when some user talks about a material currency, she does

not really care on the costs incurred in the maintenance of working currency, or better she is interested in her direct costs (interest rates, etc.). On the other hand, when talking about mining and making computing power available, some user would prefer to give her hardware to some higher purpose than to a consensus mechanism.

Finally in our view the Nonce Rush implemented in Bitcoin does not have any sense when thinking of a blockchain uncoupled from a currency, as it implies an enormous waste of computing power for mostly useless and costly calculations.

5 Conclusions

Nowadays Bitcoin is at the highest international interest because of the prospects both on opportunities and risks. At the time of writing, it has an estimated market capitalization of about 4 billion USD [57], while there is a pletora of alternative cryptocurrencies that try to emerge proposing their specific characteristics. If not enough, a new generation of systems relying on concepts and technologies inspired by those of Bitcoin are emerging. All this makes the focus on the features of the blockchain-based systems extremely interesting. There are a number of advantages inherently embedded in these systems. First of all they generally exclude any central authority, avoiding in this way a single point of attack. Besides that, they represent an attempt to coniugate both transactor's and recipient's privacy with trusted, publicly verifiable transations. All this is possible thanks to protocols that combine different cryptographic primitives and schemes into innovative designs, and which in turn pave the way to new very interesting applications with an high impact on society. When talking about currencies, even more advantages emerge. For example merchants are motivated to accept cryptocurrencies because fees are lower than the $2\% - 3\%$ typically imposed by credit card processors, and futhermore these fees are usually self-imposed and paid by the purchaser, not the vendor.

But at the same time some weaknesses are evident. If the theory behind dependable and secure computing is quite young, as it has only been developed at the beginning of this century [58], then the theory on cryptocurrencies is in its infancy, and no one knows if important issues will arise in the near future. Actually at present time blockchain-based systems can be seen as "moving targets", above all Bitcoin. Several security threats exist (e.g. 51% attack, other double spending attacks, wallet theft) and from time to time some new vulnerability is spotted, even if the current practice with these system (Bitcoin, first of all) is encouraging and it indicates that there is a successful effort to face attacks or recover from critical mishaps [59].

But some other flaws impend over a broader diffusion of cryptocurrencies or other blockchain-based systems. First of all, the good practice of waiting for transaction confirmations sets serious limits to the use of Bitcoin (and other similar cryptocurrencies) in “main street” stores, restricting its usage range primarily to on-line vendors with postponed delivery services. Another important criticism is about mining costs. Indeed the mining problem is actually a kind of artificial puzzle of no practical concern (besides, of course, the existence of the network), which requires an increasing amount of computational resources over the time. Current alternatives to the Bitcoin proof-of-work can mitigate in a way the Nonce Rush escalation experienced in these last few years (see §§3.5 and 4.1), but all the blockchain-based system to date use a considerable amount of electric energy in order to perform computing tasks at the sole scope of mining. Some statistics on Bitcoin report that - at the time of writing - the network hashrate would be equivalent to 6809823.32 petaflops, which is more than 200000 times China’s Tianhe-2, the fastest supercomputer in the world [60]. Of course such comparison is not significant in term of the actual ability to solve computational problems, as Bitcoin’s mining does not rely - if not perhaps marginally - on floating point operations, nor are its hash calculations used to solve any problem beside that of mining. But it gives an idea of the tremendous amount of operational energy for these kind of networks when deployed on a large scale. Some examples of useful proof-of-work calculations and mining exist, but are limited to specific problems or rely on complex and maybe vulnerable protocols.

In a future work we will show that the use of a workflow like that depicted in Figure 4 is actually a main misconception in blockchain-based systems. We also intend to introduce an alternative system where a computational problem of “public interest” can be solved decoupling it from the system management, as well as from the (optional) currency management and mining. This approach aims to be a more efficient solution when compared to the other currently available systems.

References

[1] Stephanie Condon. Judge spares e-gold directors jail time. `http://www.cnet.com/news/judge-spares-e-gold-directors-jail-time/`, 2008. Accessed: 2015-10-14.

[2] Jack Cloherty. ’black market bank’ accused of laundering $ 6b in criminal proceeds. `http://abcnews.go.com/US/black-market-bank-accused-laundering-6b-criminal-proceeds/story?id=19275887`, 2013. Accessed: 2015-10-14.

[3] Satoshi Nakamoto. Bitcoin: A peer to peer electronic cash system. `https://bitcoin.org/bitcoin.pdf`, 2008.

[4] Nick Szabo. Formalizing and securing relationships on public networks. `http://ojphi.org/ojs/index.php/fm/article/view/548`, 1997. Accessed: 2015-11-27.

[5] Vitalik Buterin. Daos, dacs, das and more: An incomplete terminology guide. https://blog.ethereum.org/2014/05/06/daos-dacs-das-and-more-an-incomplete-terminology-guide/, 2014. Accessed: 2015-07-20.

[6] Aaron Wright and Primavera De Filippi. Decentralized blockchain technology and the rise of lex cryptographia. http://papers.ssrn.com/sol3/papers.cfm?abstract_id=2580664, 2015.

[7] Melanie Swan. *Blockchain: Blueprint for a New Economy.* " O'Reilly Media, Inc.", 2015.

[8] Kevin Delmolino, Mitchell Arnett, and Ahmed Kosba. A programmer's guide to ethereum and serpent. https://mc2-umd.github.io/ethereumlab/docs/serpent_tutorial.pdf, 2015. Accessed: 2015-06-08.

[9] Solidity documentation. https://solidity.readthedocs.io/en/develop/. Accessed: 2015-06-08.

[10] Icann (internet corporation for assigned names and numbers). https://www.icann.org/. Accessed: 2015-07-14.

[11] V. S. Ramaswamy and S. Namakumari. *Marketing Management: Indian Context with Global Perspective.* MacMillan, 2013.

[12] Namecoin website. https://namecoin.org/. Accessed: 2015-07-14.

[13] Ethereum development tutorial. https://github.com/ethereum/wiki/wiki/Ethereum-Development-Tutorial. Accessed: 2015-07-20.

[14] Bitcoin wiki: Smart property. https://en.bitcoin.it/wiki/Smart_Property. Accessed: 2015-11-27.

[15] Bitcoin wiki: Colored coins. https://en.bitcoin.it/wiki/Colored_Coins. Accessed: 2015-08-10.

[16] Proof of existence website. http://www.proofofexistence.com/. Accessed: 2015-06-13.

[17] Adept live demo. https://www.youtube.com/watch?v=U1XOPIqyP7A. Accessed: 2015-07-20.

[18] telehash home page. http://telehash.org/. Accessed: 2015-11-27.

[19] Bittorrent home page. http://www.bittorrent.com/. Accessed: 2015-11-27.

[20] Bitcoin wiki: Script. https://en.bitcoin.it/wiki/Script. Accessed: 2015-07-21.

[21] Altcoins website. http://altcoins.com/. Accessed: 2015-11-26.

[22] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet x.509 public key infrastructure certificate and certificate revocation list (crl) profile. `http://tools.ietf.org/html/rfc5280`, 2008.

[23] Alfred J Menezes, Paul C Van Oorschot, and Scott A Vanstone. *Handbook of applied cryptography*. CRC press, 2010.

[24] Bitcoin developer guide. `https://bitcoin.org/en/developer-guide`. Accessed: 2015-07-14.

[25] Don Johnson, Alfred Menezes, and Scott Vanstone. The elliptic curve digital signature algorithm (ecdsa). *International Journal of Information Security*, 1(1):36–63, 2001.

[26] Hans Dobbertin, Antoon Bosselaers, and Bart Preneel. Ripemd-160: A strengthened version of ripemd. In *Fast Software Encryption*, pages 71–82. Springer, 1996.

[27] Bitcoin block explorer. `https://blockexplorer.com/`. Accessed: 2015-07-13.

[28] The explorer for the ethereum blockchain. `https://etherchain.org/`. Accessed: 2015-07-20.

[29] Bitcoin wiki: Protocol documentation. `https://en.bitcoin.it/wiki/Protocol_documentation`. Accessed: 2015-07-14.

[30] Turing completeness. `https://en.wikipedia.org/wiki/Turing_completeness`. Accessed: 2015-11-29.

[31] StephenM347. Bitcoin@stack exchange: What's the purpose of a maximum block size? `https://bitcoin.stackexchange.com/questions/37292/whats-the-purpose-of-a-maximum-block-size/37303#37303`. Accessed: 2015-07-25.

[32] Ralph C. Merkle. A digital signature based on a conventional encryption function. In *Advances in CryptologyCRYPTO87*, pages 369–378. Springer, 1988.

[33] Carlos M. Gutierrez and Patrick Gallagher, editors. *SECURE HASH STANDARD*, volume 180–3 of *Federal Information Processing Standards Publication*. National Institute of Standards and Technology, 2008.

[34] Bitcoin wiki: Block hashing algorithm. `https://en.bitcoin.it/wiki/Block_hashing_algorithm`. Accessed: 2015-07-25.

[35] Ken Shirrif. Bitcoin mining the hard way: the algorithms, protocols, and bytes. `http://www.righto.com/2014/02/bitcoin-mining-hard-way-algorithms.html`, 2013. Accessed: 2015-07-25.

[36] Markus Jakobsson and Ari Juels. Proofs of work and bread pudding protocols. In *Proceedings of the IFIP TC6/TC11 Joint Working Conference on Secure Information Networks: Communications and Multimedia Security*, CMS '99, pages 258–272. Kluwer, B.V., 1999.

[37] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *Advances in Cryptology - CRYPTO92*, pages 139–147. Springer, 1993.

[38] Phillip Rogaway. Nonce-based symmetric encryption. In Bimal Roy and Willi Meier, editors, *Fast Software Encryption*, volume 3017 of *Lecture Notes in Computer Science*, pages 348–358. Springer, 2004.

[39] Meni Rosenfeld. Analysis of hashrate-based double spending. arXiv preprint arXiv:1402.2009v1, 2014. `http://arxiv.org/abs/1402.2009v1`.

[40] litecoin website. `https://litecoin.org/`. Accessed: 2015-07-20.

[41] Colin Percival. Stronger key derivation via sequential memory-hard functions. `http://www.bsdcan.org/2009/schedule/attachments/87_scrypt.pdf`, 2009.

[42] Ronald Rivest. The md5 message-digest algorithm. `http://tools.ietf.org/html/rfc1321`, 1992.

[43] William M. Daley and Raymond G. Kammer, editors. *DATA ENCRYPTION STANDARD (DES)*, volume 46–3 of *Federal Information Processing Standards Publication*. National Institute of Standards and Technology, 1999.

[44] Sunny King and Scott Nadal. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. `https://peercoin.net/assets/paper/peercoin-paper.pdf`, 2012. Accessed: 2015-07-20.

[45] Akira Takura, Satoshi Ono, and Shozo Naito. A secure and trusted time stamping authority. In *Internet Workshop, 1999. IWS 99*, pages 88–93. IEEE, 1999.

[46] Pieter Wuille. Bitcoin@stack exchange: What does the term "longest chain" mean? `http://bitcoin.stackexchange.com/questions/5540/what-does-the-term-longest-chain-mean/5542#5542`. Accessed: 2015-11-29.

[47] Joshua Kolden. Bitcoin@stack exchange: What are bitcoin "confirmations"? `http://bitcoin.stackexchange.com/questions/146/what-are-bitcoin-confirmations/160#160`. Accessed: 2015-11-29.

[48] Hermann Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Real-Time Systems Series. Springer, 2011.

[49] Giuliano Laccetti and Giovanni Schmid. Brute force attacks on hash functions. *Journal of Discrete Mathematical Sciences and Cryptography*, 10(3):439–460, 2007.

[50] Non-specialized hardware comparison. `https://en.bitcoin.it/wiki/Non-specialized_hardware_comparison`. Accessed: 2015-09-15.

[51] Mining hardware comparison. `https://en.bitcoin.it/wiki/Mining_hardware_comparison`. Accessed: 2015-09-15.

[52] Sunny King. Primecoin: Cryptocurrency with prime number proof-of-work. `http://primecoin.io/bin/primecoin-paper.pdf`, 2013. Accessed: 2015-07-20.

[53] Caldwell C. Finding primes and proving primality. `http://primes.utm.edu/prove/merged.html`, 2002. Accessed: 2015-09-15.

[54] Lifchitz H. Generalization of euler-lagrange theorem. `http://www.primenumbers.net/Henri/us/NouvTh1us.htm`, 1998. Accessed: 2015-09-15.

[55] Curecoin: What is curecoin? `https://curecoin.net/knowledge-base/about-curecoin/what-is-curecoin/`. Accessed: 2015-09-15.

[56] Gridcoin wiki home page. `http://wiki.gridcoin.us/Main_Page`. Accessed: 2015-09-15.

[57] bitcoin charts website. `http://bitcoincharts.com/`. Accessed: 2015-11-26.

[58] Algirdas Avižienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *Dependable and Secure Computing, IEEE Transactions on*, 1(1):11–33, 2004.

[59] Wikipedia: History of bitcoin. `en.wikipedia.org/wiki/History_of_Bitcoin`. Accessed: 2015-11-28.

[60] Top 500 supercomputing sites. `www.top500.org`. Accessed: 2015-11-30.