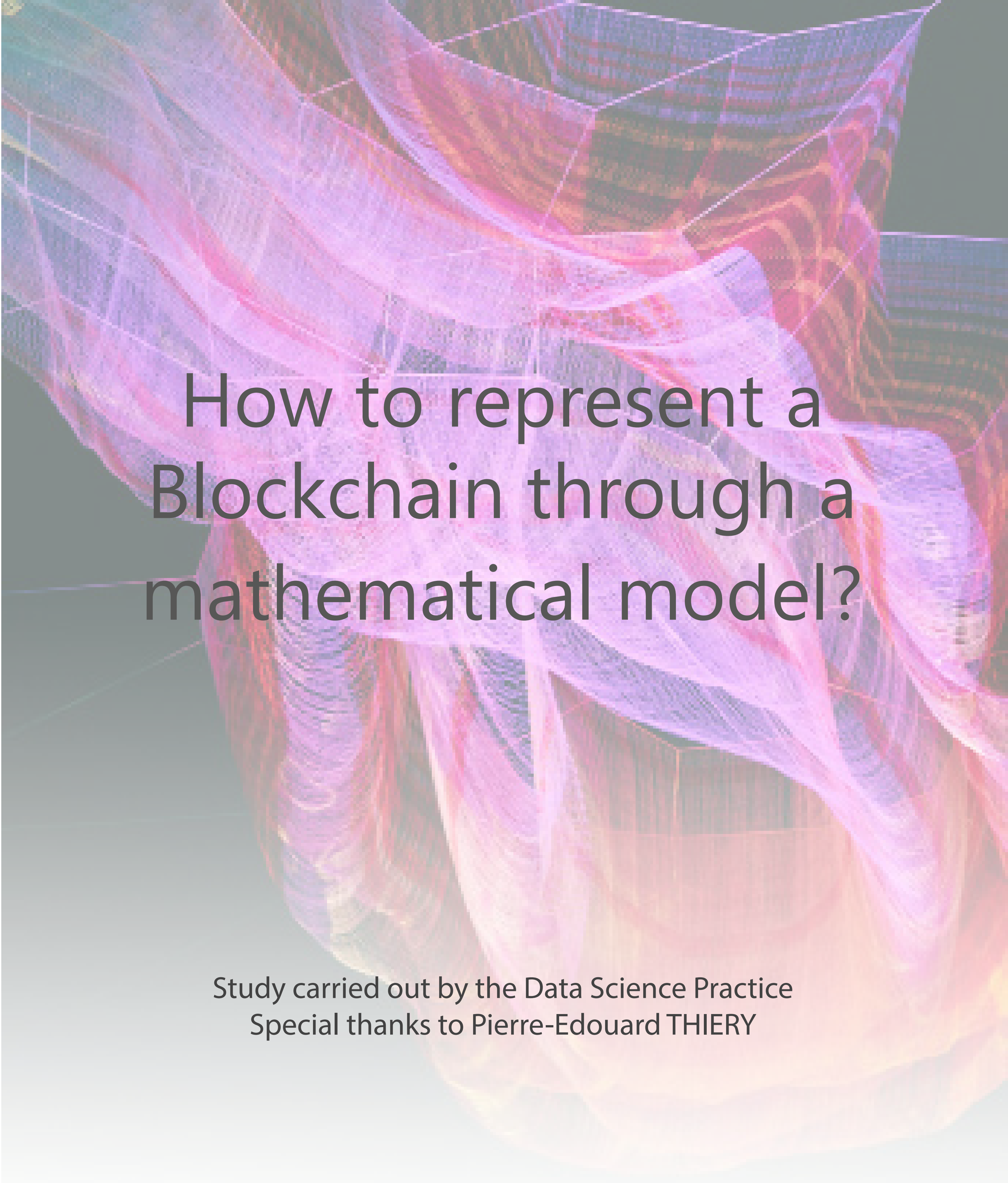


How to represent a Blockchain through a mathematical model?

Study carried out by the Data Science Practice
Special thanks to Pierre-Edouard THIERY

coperneec

AVRIL 2020

Summary

Introduction

When it comes down to Data Science, most people directly think of Machine Learning and its ever more powerful algorithms; nonetheless Data Science would not be possible without data, and the first condition for a widespread use consists in storing and accessing the data in a secure way. Among the many solutions which exist, a new possibility has arisen for around ten years: Blockchain. This concept deserves to be analyzed carefully; indeed, due to its rising popularity, ever more people mention it without fully understanding the technical complexity which lies behind the word "Blockchain". A first pedagogical difference can be made: from now on Blockchain will refer to the overall theory, whereas "a" blockchain will refer to a specific implementation of the theory. For instance, Bitcoin (with a capital B) is the name for a given blockchain which aims at storing the exchanges made with a dematerialized currency called "bitcoin". As a first definition, we can say that Blockchain is a way of storing data through a network of nodes which work on a peer-to-peer basis, without any central authority, and in a secure way.

Bitcoin is not the only blockchain nowadays, but it is the first one, implemented in 2008 [1]. Many other blockchains, going beyond the mere storage of exchanges of a dematerialized currency, have been created for ten years, like Ethereum, and each one of them is part of the Blockchain world.

We believe that a first step into this world is to precisely understand the nuts and bolts of Bitcoin: the latter is probably the simplest blockchain that exists, but its key principles can prove to be helpful before going further into the fairly uncharted territory of Blockchain. Therefore this paper's purpose is not to depict all the subtleties of Bitcoin, but to define as precisely as possible a mathematical framework to grasp the gist of it.

1 Some Mathematical Preliminaries

In this first part, we present several mathematical tools and notions which are ubiquitous within Blockchain [2] [3], and so particularly necessary to fully understand Bitcoin.

● HASH FUNCTIONS

Definition 1 (Hash Function)
A hash function is a deterministic function which transforms data of arbitrary (huge) size into a fixed-size number. The purpose of a hash function is to characterize an input using the output fixed-size number. To do so efficiently, a hash function must respect several qualitative properties:

- *the "avalanche effect", meaning that a slight change in the input must lead to a completely different result;*
- *uniformity over the output range;*
- *a limited number of collisions, a collision being defined by two different inputs giving the same output value;*
- *the non-invertible property, meaning that it is not realistic to compute the input from the output value.*

So the gist of hash functions is that it is pretty simple to compute the output value for a given input, and by doing so verifying that the latter has not been modified, whereas it is extremely costly to find the input from a given output.

From the above definition, we see that the idea of hash functions is rather simple: a function which computes an output as a way of characterizing an input. But the construction of a "good" hash function, according to the qualitative properties, is overly complex; it would be pointless to delve into such details within this paper.

One of the most famous hash functions is $SHA256$, which transforms any message (a string) into a binary number of 256 bits (which is sometimes written as a 64-digit hexadecimal number). To illustrate the avalanche effet, we can notice that $SHA256(Blockchain)$ is worth:

625da44e4eaf58d61cf048d168aa6f5e492dea166d8bb54ec06c30de07db57e1

whereas $SHA256(blockchain)$ is worth:

ef7797e13d3a75526946a3bcf00daec9fc9c4d51ddc7cc5df888f74dd434d1

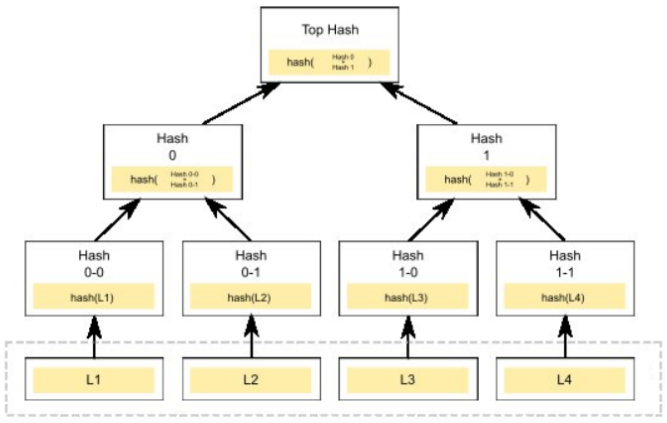
● MERKLE TREE

Definition 2 (Merkle Tree)
A Merkle Tree is a way of storing a potentially huge amount of data, while providing the user with a simple way to check that the data have not been modified. To build a Merkle tree, let us consider a hash function H and a set of data:

$$\mathbb{D} = \{d_1,\ d_2, \ldots,\ d_n\}$$

The leaves of the tree are given by the hash values of the elements within $\mathbb{D}$: $H(d_1)$, $H(d_2)$, etc. We can then build the tree recursively. Let us assume an entire layer L is known, the first node within the next layer is given by the hash value of the first two nodes within L, the second node is given by the hash value of the third and fourth nodes within L, etc. If there is an odd number of nodes within L, then the last node within the new layer is simply equal to the last node within L.

To make the comprehension of Merkle trees easier, we provide the reader with a graphical example. We consider $\mathbb{D} = \{L1,\ L2,\ L3,\ L4\}$, and figure 1 shows the corresponding Merkle tree.



Definition 3 (Merkle Root)
The Merkle root for $\mathbb{D} = \{d_1, d_2, \ldots, d_n\}$ *is denoted* $R^H(\mathbb{D})$ *and is equal to the top hash of the corresponding Merkle tree.*

● Asymmetric Cryptography

How to encrypt a message to make sure that only the emitter and the receiver will be able to decipher it? How to verify that the so-called emitter of a message is the true one? Asymmetric cryptography [4] is a way to address those two questions. The staple principle of asymmetric cryptography is the use of public and private keys.

Definition 3 (Principles Of Asymmetric Cryptography)
We start with the encryption of a numeric message M. For instance Max would like to send M to Mary, without anyone else being able to read it. Max's public and private keys are denoted $(K_{pu}^{Max}, K_{pr}^{Max})$*, and Mary's ones* $(K_{pu}^{Mary}, K_{pr}^{Mary})$*. Max can encrypt M by using an encryption function C and Mary's public key:*

$$V = C(K_{pu}^{Mary}, M)$$

Once Mary has received the value V, she can decipher the message using her private key and the deciphering function D associated with C:

$$M = D(K_{pr}^{Mary}, V)$$

The signature of a message M is the second question: how can Mary be certain that Max is really the one who has emitted the message? The solution is the following: Max sends to Mary both M and the signature S where:

$$S = C(K_{pr}^{Max}, M)$$

Once Mary has received (S, M), she can decipher S with Max's public key:

$$m = D(K_{pu}^{Max}, S)$$

If m = M, it proves that Max has really sent the message M.

The pivotal point is that the private key must always be kept private: only his/her owner must know it, whereas the public key can be sent freely to anyone. The RSA algorithm is probably the most famous technique of asymmetric cryptography: it is based on prime numbers to generate public and private keys, and simple arithmetical operations (power function and modulos) are used to encrypt and decipher a message.

2 A First Approach Of Bitcoin Via Mathematics

As of now, H denotes the hash function $SHA256$.

Definition 4 (Entry)
An entry is, by definition, the staple piece of information which is stored within a blockchain. To rephrase it, a blockchain's purpose is to store entries in a secure way. As of now, a generic entry is denoted Tr*; we choose this notation since, for Bitcoin, an entry is actually a transaction between two entities.*

Since we consider Bitcoin, an entry would be a mere transaction: for instance Max sends 1.17 bitcoins to Mary. But by writing this, we point out the first difficulties a blockchain such as Bitcoin faces: does Max really own 1.17 bitcoins? Are we sure that Max has initiated such a transaction? To answer those questions, we will introduce later the concept of validation protocol.

As we may easily imagine, a blockchain is a chain of blocks; so we have to define what a block is.

Definition 5 (Block)
A block B is defined as a vector of entries. We denote its size N_B*:*

$$B = (Tr_1, \ldots, Tr_{N_B})$$

So to create a block, we just pile up entries.

To "chain" the blocks, another notion plays a significant role: the proof-of-work, sometimes denoted PoW.

Definition 6 (Proof-Of-Work)
The proof-of-work is a mathematical problem, whose purpose is to create a link between two blocks. This link will be materialized within the header of the second block. Someone trying to work out the proof-of-work is called a miner.

Let us consider two blocks, denoted B^{prev} *and B, and a number called bits and denoted b. b gauges how difficult the proof-of-work is: from b, a target number can be directly computed. This target is a 64-digit hexadecimal number with several 0 for its left digits, for instance:*

$$0000000000000000021047526c065745de75af6dcd473556dced2bc$$

We assume that the hash of the previous block is known, $H(B^{prev})$*; we will see shortly after how to define the hash of a given block. Solving the proof-of-work for the block B, also known as mining the block B, amounts to finding a number, called the "nonce", such that:*

$$H\left(H(B^{prev}) \oplus R^H(B) \oplus timestamp(t) \oplus b \oplus nonce\right) \leq target$$

where

- $\oplus$ *denotes the concatenation operation;*
- $timestamp(t)$ *denotes the current time, up to the seconds.*

Since H is a hash function, the only possibility to find a suitable nonce is to try brutally, by increasing the value of nonce while t is moving. We assume that, at t^0, we find a nonce $nonce^0$ which solves the proof-of-work. The hash $H(B)$ of the block B is defined by:

$$\underbrace{H\left(H(B^{prev}) \oplus R^H(B) \oplus timestamp(t^0) \oplus b \oplus nonce^0\right)}_{H(B)} \underbrace{\leq target}_{by\ definition}$$

Since the hash of a block is defined recursively following the above procedure (we supposed that $H(B^{prev})$ was already known), we need an initialization: if B is the first block, there is no previous block, so $H(B^{prev})$ is a mere convention.

Definition 7 (Block's Header)
Once the proof-of-work has been solved for a couple of blocks (B^{prev}, B), we can define the header of the block B, using the above notations:

$$Head(B) = \left(id_m,\ H(B^{prev}),\ R^M(B),\ timestamp(t^0),\ b,\ nonce^0,\ H(B)\right)$$

where id_m denotes the identity of the mining entity.

The proof-of-work is a pivotal notion in the world of Blockchain. A few things are worth noticing.

It is called proof-of-work for a specific reason: solving the above-mentioned mathematical problem is a way to prove that you have dedicated some time to find the answer. Thus you prove that you have worked. Indeed, since H is a one-way function, the best method to find a valid nonce is to try every possible value: if *nonce* does not fit, then $nonce \leftarrow nonce + 1$. A "miner", i.e. someone with one (or many) computer(s) trying to work out the proof-of-work, needs an ever-more important computational power, since the difficulty of the proof-of-work is increasing with the number of "mined" blocks: b is adjusted by an algorithm, it depends on the number of blocks already present in Bitcoin.

Once a miner has solved the proof-of-work, it is easy for anyone to check that the solution is correct: all it takes is to compute the hash

$$H\left(H(B^{prev}) \oplus R^H(B) \oplus timestamp(t^0) \oplus b \oplus nonce^0\right)$$

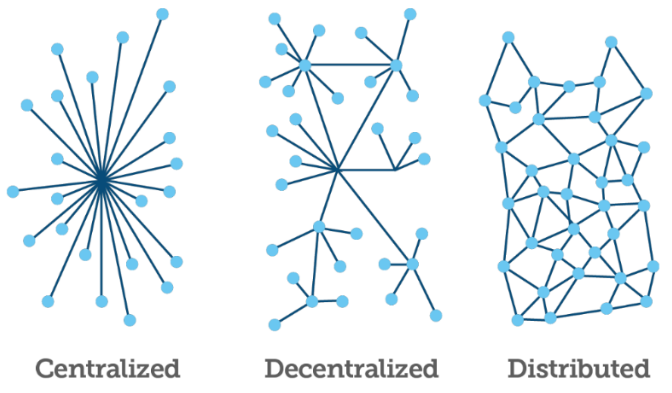
with t^0 and $nonce^0$ provided by the miner.

Since mining requires some computational power, it is worth doing it if there is a reward: no incentive would mean no one ready to try. With Bitcoin, a miner who has correctly solved a proof-of-work problem is rewarded with... bitcoins. The reward is decreasing with the number of blocks in Bitcoin. That is how the overall system can create new bitcoins.

To better understand all those mechanisms, we present in the following sections all the steps which lead to the construction of Bitcoin, the blockchain(s) storing bitcoin transactions.

3 The construction of the Bitcoin blockchain step by step

To construct the Bitcoin blockchain, the first point is to understand who the "players" are. In the introduction, we have given a primary definition for a blockchain: a way of storing data through a network of nodes which work on a peer-to-peer basis without any central authority. So the staple players are the nodes, and they are organized in a distributed manner (Figure 2).



To have a deeper insight into the particularities of Blockchain, we provide the user with two examples for the two extreme kinds of networks [5].

The commercial banks within the Eurozone are part of a centralized network, whose central authority is the European Central Bank (ECB): the commercial banks depend directly on the ECB for most of their activities, and by default, they have no other choice than trusting the ECB. Confidence is a decisive notion: the good working of a centralized network hinges on the ability of the central node to be trusted. If the confidence within the central authority disappears (corruption, lies, hack, etc.), the very network itself will dysfunction. For instance, when it comes to central banks, there is a huge amount of academic papers investigating how a central bank can maintain a high level of confidence.

The purpose of Blockchain is to provide a framework to implement distributed networks in a secure way. Indeed without security, it is impossible for such a network to be socially or economically acceptable. Here, since there is no central or even semi-central authority, the question of confidence is utterly different; that is why Blockchain is sometimes considered a "revolution of trust". In the centralized and decentralized system, the efforts aim at preventing the decisive node(s) from behaving badly: in a perfect world, the decisive node(s) should be impervious to corruption, computer attacks, or any other threat. In Blockchain, the problem is not that a node can be subject to an attack, but how to make sure that the overall network keeps working even if some of its nodes are corrupt. To put it simply, there is no room for confidence in Blockchain: it has to be replaced by computational techniques.

After this rather extensive introduction, we can delve into the details of the Bitcoin blockchain's construction [6].

Definition 8 (Node)
A node is the staple entity within the distributed network; in Figure 2, each blue point represents a node. The physical reality

behind a node is merely someone with one or many computers. There are several kinds of nodes:

- *simple nodes, whose role is to transmit information, without holding a local copy of the blockchain;*
- *complete nodes, whose role is to mine new blocks; they keep a local copy of the whole blockchain.*

Here we have to mention the most important truth about Blockchain: since there is no central authority, there is no special node which would "wisely" keep the true version of the blockchain, otherwise the whole construction would collapse: you would just need to hack the "wise" node to hack the entire system.

Every complete node keeps its own version of the blockchain; but due to the nuts and bolts of the network, those nodes reach a consensus. This consensus is sometimes referred to as "the" blockchain, but the reality is that there is no such thing as "the" blockchain.

$*$ STEP 1: a new entry in the system

Let us denote Tr_X the new entry, for instance Max sends 1.17 bitcoins to Mary; it enters the network via a node, denoted n_{enter}, be it simple or complete. Every node has its own validation protocol, i.e. a program to check whether an entry is a valid one or not. A node can choose to use a standard program for the validation step, but it is not compulsory: for many reasons (experimental, fraud, local necessity, etc.) a node can opt for a customized validation protocol.

Definition 9 (Validation Protocol)
A node does not trust the information it receives (no room for trust in Blockchain), so it performs a few checks using its own validation protocol. The latter can be seen as a function:

$$\begin{array}{rcccc} PrV & : & \mathbb{N}\times\mathbb{E} & \to & \{True, False\} \\ & & (n, Tr) & \mapsto & PrV(n, Tr) \end{array}$$

where $\mathbb{N}$ is the set containing all the nodes, and $\mathbb{E}$ the set of all transactions.

So in our case, n_{enter} computes $PrV(n_{enter}, Tr_X)$. If the answer is $True$, n_{enter} considers the transaction as a valid one: Tr_X is transmitted to the nearby nodes. Otherwise, n_{enter} rejects the entry, it is not transmitted.

What is a non-valid transaction? To better understand, we focus on two kinds of issues: a fake identity and the double-spending problem.

The fake identity is a classic issue: you pretend to be someone else for your own interest. For instance, if Mary could steal Max's identity, she could initiate the transaction "Max sends 1.17 bitcoins to Mary". To cope with those issues, Bitcoins is based on asymmetric cryptography: each transaction is signed by the one who initiates it.

The double-spending problem means that someone could try to use bitcoins he/she does not own. To make things clear, it is like pretending in real life that you have 20 euros in your pocket when you only have 10. The problem is particularly critical with bitcoins (and other cryptocurrencies) because you cannot rely on coins and notes, or any central authority.

With our notations, if n_{enter} is a simple node, it will carry out some elementary checks: for our model we assume that a simple node only verifies that the transaction is signed by its initiator.

If n_{enter} is a complete node, we explain in step 2 what happens.

$*$ STEP 2: an entry reaches a complete node

Once a transaction Tr_X has entered the network, it can eventually reach a complete node, denoted n_c. Similarly to a simple one, a complete node first computes $PrV(n, Tr_X)$. The checks carried out by a complete node are more extensive than those of a simple node. In our model, we assume that a complete node verifies both the identity of the initiator and that there is no double spending.

Indeed it is possible for a complete node to make sure that there is no double spending since it keeps its local version of the blockchain: by accessing the history of all the transactions, the node can give an answer to the following question: does Max own 1.17 bitcoins he can send?

If the transaction Tr_X is deemed valid, n_c adds it to its local list of valid transactions $L_{loc}^{n_c}$:

$$L_{loc}^{n_c}.append(Tr_X)$$

and then, Tr_X is transmitted to the nearby nodes.

As a complete node "mines", it creates a new block by adding into it some of the valid transactions present in its local list $L_{loc}^{n_c}$:

$$B_{n_c} = \left(Tr^1, \ldots, Tr^N\right)$$

where $Tr_i \in L_{loc}^{n_c}$ for $1 \leq i \leq N$. Then n_c tries to solve the proof-of-work for $(B_{last}^{n_c}, B_{n_c})$ where $B_{last}^{n_c}$ denotes the last block of the local version of the blockchain.

The complete nodes are always competing with each other: when there are enough valid transactions in its local list, or after a given period of time, a new block is generated by a complete node which then tries to solve the proof-of-work problem. Once this is done, a new block with its header is emitted by the presumed winner.

$*$ STEP 3: a new block has been emitted

When a new block B_{new} and its header have been emitted by a node n, it means that n claims to have solved a proof-of-work problem. The new block is transmitted to the nearby nodes: since there is no room for confidence in Blockchain, again the nodes carry out many checks. Let us denote n_r a node which has just received B_{new}: if n (complete) is trying to solve its own proof-of-work, it stops. If n (complete or simple) has already received B_{new}, the latter is automatically rejected. Otherwise it starts by verifying that all the transactions in

B_{new} are valid ones. If one transaction is not valid, B_{new} is rejected.

Then n checks that the proof-of-work has been correctly performed: it takes a single computation of the hash function to see whether or not the output value is inferior to the target. It is also quick to check that the Merkle root of the new block and its hash value are correct.

If n is a complete node, the situation is a little bit more complex. Indeed, in this particular case, n must decide whether or not B_{new} is going to be added to its local version of the blockchain. The block is added if the hash of the previous block as it appears in the transmitted header $Head(B_{new})$ is equal to the hash of the last block in the local version of the blockchain. In this case, n deletes the transactions within B_{new} from its local list, and the miner who has emitted the new block is rewarded: n starts to create a new block with valid transactions, and this new block begins with a specific line saying that the winner of the previous competition (whose identity is stored in $Head(B_{new})$) is rewarded with some bitcoins.

Otherwise B_{new} is stored by n: maybe n has not yet received an intermediary block ($\star$).

– What is a conflict of version?

The last point leads us to address the difficult question of conflicts within the nodes about the "true" version of the blockchain: what happens when some nodes store a version Bk_1 of the blockchain, and other nodes a version Bk_2?

First it is important to understand that such a situation is possible. Let us assume that, at a given time t, all the nodes have the same version Bk_0 of the blockchain: so "the" blockchain really exists. At $t_1 > t$, two different nodes emit two different valid blocks, B_a and B_b. Such a situation can occur because a complete node is free to choose any valid transactions in its local list to generate a new block.

Since it takes time for a new block to reach every node of the network, some nodes will receive B_a before B_b, and others B_b before B_a. Those two blocks have been assumed to be valid ones, so in both cases they will be added: at $t_2 > t$, we may find nodes with a local version $Bk_0 + B_a$, and others with $Bk_0 + B_b$. "The" blockchain no longer exists: this is called a "fork".

– The rules of consensus

To solve such a discrepancy, rules of consensus are necessary: a complete node will always keep the "longest" version among all the local ones it stores. By "longest", the practical idea is to keep the version which has required the biggest amount of work; as we will see shortly after, this is a way to beef up the security of the blockchain.

Let us assume that, at $t_3 > t_2$, a node with local version $Bk_0 + B_a$ emits a new valid block B_c, and that this block reaches all the nodes before another one is emitted. The nodes which have stored $Bk_0 + B_a$ as a local version will now keep $Bk_0 + B_a + B_c$. But the nodes which have stored $Bk_0 + B_b$ are facing a dilemma.

When such a node n receives B_c, it is important to keep several things in mind: the local version is $Bk_0 + B_b$, but n has also received B_a before. Since it was not possible to chain B_a with $Bk_0 + B_b$, B_a is stored ($\star$). B_c cannot be added to $Bk_0 + B_b$, but it can be added to B_a: $B_a + B_c$. There is a doubt concerning the blocks after Bk_0: between $Bk_0 + B_b$ and $Bk_0 + B_a + B_c$, n will keep the longest chain, i.e. $Bk_0 + B_a + B_c$. We reach a consensus between all the nodes.

4 A First Insight Into The Security Of The Blockchain

To have an overall view of Bitcoin, it is important to understand why the above-mentioned mechanisms create a secure blockchain.

The proof-of-work plays a determining role. It makes sure that it is not possible to modify the history of the blockchain: indeed changing one block means changing its hash and so all the hash values of the next blocks. The whole chain is no longer coherent: to change the past, it would take a impressive amount of work to "rebuild" a coherent chain of blocks, starting from the block which has been changed.

Then, if someone would like to cheat the overall system, hacking a single node, even a complete one, for instance to make it accept an illegal transaction, would have no effect. Since there is no room for confidence in Blockchain, the other nodes would immediately detect the illegal transaction and reject it. So the only way to make the system accept an illegal transaction would be to take over a majority of the complete nodes. And even in this case, the illegal transaction would only be recognized as a valid one by the corrupt nodes; this would create a discrepancy between the versions of the blockchain kept by the nodes: no consensus would be reached. Thus, even if one manages to take over half of the complete nodes, we could believe that the illegal transaction is legal by considering the majority of the nodes, but we would also observe an absence of consensus. Of course this situation is merely theoretical, since it would require a computational effort beyond imagination to take over half of the complete nodes.

Conclusion

In this paper, we have set forth a mathematical approach of Bitcoin, as a first step into Blockchain. Mathematics notions are ubiquitous within this world, and thus we believe it is by using a framework inspired by mathematics that we may achieve a sound comprehension of its most important mechanisms.

Nonetheless, it would not be wise to claim that we have spanned all the aspects of this new technology. Blockchain is an expending area, thus facing many issues – for instance how to challenge the proof-of-work, which requires huge amount of electrical power, with other systems of proof (proof-of-stake, proof-of-authority) – but also carrying the promise of new perspectives.

It is on the latter we would like to dwell on to finish this paper. Bitcoin is only the simplest blockchain that exists; this technology can be used for more complex operations than mere transactions. This is for instance the purpose of Ethereum, the second most popular blockchain: Ethereum relies on a cryptocurrency, ethereum, which can be of course exchanged, but it is only a medium for something bigger. Ethereum aims at storing smart contracts into its blockchain, i.e. contracts whose behavior is predetermined in some source code stored in a blockchain (for instance, if your house is burnt down, then you receive some money). Since trust is pivotal for any traditional contract, Ethereum's bet is that it is possible to use Blockchain technology to create a new kind of contract, the so-called "smart contracts".

References

[1] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *www.bitcoin.org*, 2008.

[2] Della Chiesa. Hiault and Tequi. *Blockchain, vers de nouvelles chaînes de valeurs.* Prospectives Accuracy, 2018.

[3] Laurent Leloup. *Blockchain, la révolution de la confiance.* Eyrolles, 2018.

[4] Johannes Buchmann. *Introduction à la cryptographie.* Dunod, 2006.

[5] Philippe Rodriguez. *La révolution Blockchain.* Dunod, 2017.

[6] Gérard Dréan. La blockchain pour les nuls. *www.contrepoints.org*, oct 2016.

A propos de Coperneec

« From Revolution to Performance »

Coperneec est un cabinet de conseil cross-sectoriel spécialiste de la valorisation de la Data. Nous intervenons sur l'ensemble de la chaîne des savoir-faire autour de la Data Science, la Data Analyse et du Data Management.

Nos méthodes et techniques scientifiques éprouvées permettent de résoudre des problématiques dans tous les secteurs de l'industrie.

Notre vocation : extraire la connaissance à partir des données et pérenniser les avancées technologiques qui en découlent. La R&D est au cœur de notre ADN et les expertises de nos consultants (data scientists, data analysts, data engineers) sont en permanence challengées afin d'accompagner au plus près les révolutions technologiques et scientifiques.

Contactez-nous

Aymeric LISBONNE
Partner
alisbonne@coperneec.com
06 88 69 67 75

coperneec

est une marque de

canopee group