Chapter 1

Consensus in Blockchain

> Codes are a puzzle. A game, just like any other game.
>
> *Alan Turing*

In this chapter, we introduce the blockchain abstraction. We first present the history of blockchain and the key results that led to its invention. We then present the key elements that allow it to be used to transfer digital assets. We finally motivate the need for a set of distributed machines, referred to as nodes or processes, to agree upon the series of blocks that constitute the blockchain. This consensus will be described in more detail in Chapter 3.

1.1 A Brief History

Blockchain builds upon discoveries in different domains of computer science. In particular, it relies on the achievements of cryptographers, cypherpunks and distributed computer scientists around the world. Modern cryptography appeared in 1975 with the publication of the Data Encryption Standard (DES) [IBM75]. At the beginning of the 80's, David Chaum, an American computer scientist and cryptographer proposed the blind signatures as a form of electronic carbon copy for payment systems [Dav83]. This system offers auditability and privacy of payments. In 1989, David Chaum founds DigiCash Incorporated as an electronic money corporation that sold its assets to eCash.

This privacy idea is one of the main motivations behind the Cypherpunk movement that Eric Hughes, Timothy May and John Gilmore started in the 90's. They decided to meet on a regular basis in San Francisco to discuss ideas related to programming and cryptography and created the Cypherpunk mailing list. As the number of people in their discussion group grew, this mailing list allowed them to reach out to a wider group of people exchanging ideas and discussing development. A famous manifesto that Eric Hughes published on this mailing list compares cash to a primary way of offering an anonymous payment system required to maintain the privacy of an open society.[1]

Advances in distributed computing contributed to discovering the building blocks of blockchain as it is known today. In 1992, Cynthia Dwork and Moni Naor required a user to compute a moderately complex crypto-puzzle before being able to access a resource in order to avoid abuse of resource usage [DN93]. The solution to this crypto-puzzle progressively led to the concept of *proof-of-work* that is presented today by machines in mainstream blockchain systems to demonstrate the workload they have computed. At the end of the 90's, the British cryptographer Adam Back proposes a hashcash mint, a partial hash collision generator, on the cypherpunk mailing list.[2] The idea is to use partial hashes to be made arbitrarily expensive to compute, but to be verified instantaneously to detect double spending. In 2002, Adam Back published his "Hashcash" concept in a technical report [Bac02]. In 2004, Hal Finney extended the previous proof-of-work with the idea of reusability.[3] He offered a system allowing a server's integrity to be publicly verified by online users.

[1] https://www.activism.net/cypherpunk/manifesto.html.
[2] http://www.hashcash.org/papers/announce.txt.
[3] https://cryptome.org/rpow.htm.

Digital money became a reality at the end of the 90's when Wei Dai sent a message on the cypherpunk mailing list, describing a system called "b-money". Similar to today's blockchains, this system allowed users to transfer money between accounts represented as public keys. In 2005, Nick Szabo proposed Bit Gold, the crypto-currency that applies a chain of hashes to guarantee immutability [Sza05]. It is in 2008 that the first protocol to transfer a cryptocurrency in a peer-to-peer fashion without the need for a central intermediary is authored by Satoshi Nakamoto [Nak08].

The corresponding Bitcoin protocol is released open source as the first blockchain in 2009. Several years later, another type of blockchain appeared that many commonly refer to as "Blockchain version 2.0": Vitalik Buterin describes Ethereum based on a generic crypto-puzzle and extends this notion of cryptocurrency by allowing users to upload not only transactions but also *smart contracts* [Sza97]—programs that can be invoked by other users. Gavin Wood proposed then a detailed description of Ethereum by publishing its Yellow Paper that kept evolving since then [Woo15]. In 2016, the Decentralized Autonomous Organisation (DAO) was proposed as a smart contract system uploaded to the Ethereum blockchain. This raised $150 million dollars but was hacked.

Since then, continuous efforts have been devoted to make blockchains more secure and their performance scalable. We will discuss some of these results in Chapter 5.

1.2 What is Blockchain?

A blockchain is a chain of blocks similar to the linked list that is usually taught to undergraduate computer science students. While it is the goal for a blockchain to be a chain, it is often the case that the existing protocols fail to implement a perfect chain. Instead they often allow the chain to fork as we will explain in Chapter 4, this is why we consider here that the blockchain can be a directed acyclic graph. Figure 1.1 depicts a blockchain whose prefix is a linked list but that forks at the end.

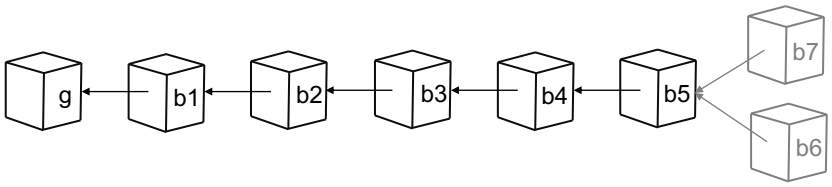


Figure 1.1: The blockchain is a directed acyclic graph that can be viewed as a chain of blocks as long as it does not fork

1.2.1 The blockchain abstraction as a directed acyclic graph

Let the *blockchain* be a Directed Acyclic Graph (DAG) $\ell = \langle B, P \rangle$ such that blocks of B point to each other with pointers P and a special block $g \in B$, called the *genesis block*, which does not point to any block but serves as the common first block. What is interesting is that a new block gets appended to the chain by pointing towards the block that was appended to the chain last. More precisely, pointers P are recorded in a block as a hash of the previous block. For example, $\langle b_1, b_0 \rangle \in P$ is a pointer from a block b_1 to the block b_0 that was appended immediately before b_1: this means that block b_1 contains the result of a hash function applied to the content of block b_0.

This abstraction aims at offering a service to track ownership of digital assets. Consider a set of participants that act as users or *clients* and have an associated account, called *address*. These clients may send data to the blockchain abstraction to indicate how to transfer their assets to another address. These data can be of the form of simple transfers, or invocations of the function of a complex program, also known as *smart contract*, however, for the sake of simplicity we refer to all these requests as *transactions*. To transfer digital assets, a participant Alice simply creates a request comprising a signed transaction $\sigma_A(T_A)$ that is a representation of a transaction T_A that is signed by Alice's signature function σ_A. The transaction T_A contains the digital assets or *coins* that she wants to transfer and the recipient address to which she wants to transfer them. Hence the transaction request contains the signature, the assets and the recipient address.

1.2.2 Signed transactions

It could be the case that a malicious user, say Bob, would be tempted to try to steal the assets of Alice. In this case Bob would write a transaction T_A that transfers some of Alice's assets to his own address. To make sure that this request is not accepted by the service, each user keeps its own signature function private. Hence Bob can only issue a $\sigma_B(T_A)$ signed by his own signature function σ_B but cannot use Alice signature function σ_A. The system can detect that the request is not signed by the owner of the address from which the assets are withdrawn. As a result, the system considers this request as *invalid* and ignores it.

This is achieved using a public key cryptosystem: the transaction signature results from the user encrypting the transaction with its private key, the system verifies that the transaction is correctly signed by decrypting the signature with the public key of the owner of the address from which the assets are withdrawn. If the result of this decryption matches the transaction, then the signature is correct and the transaction can be recorded by the system, otherwise it is deemed invalid and the transaction is not recorded.

For a transaction to be *valid* it needs both to be correctly signed and to transfer assets that exist at the source address. The results of valid transactions are stored into the blocks of the blockchain either in the form of Unspent Trans-

action Output (UTXO) or in the form of balances, so that the blockchain and all its stored transaction results indicate what assets each user owns. It can be the case that two transactions *conflict* with each other when two concurrently issued transactions try to withdraw from the same account the same assets. For example, if two transactions withdraw 5 coins each from an account with a balance of 9 coins: both transactions cannot execute without violating the integrity of the corresponding account.

Unfortunately, the problem of maintaining asset ownership is not so simple because the blockchain is implemented in a distributed fashion. In particular, the blockchain abstraction is replicated at multiple nodes called *miners*, and users may perceive that the same asset is owned by different participants if they contact different miners.

1.2.3 Distributed implementation of the blockchain abstraction

The *blockchain system* often refers to the distributed protocol that implements the aforementioned blockchain abstraction. Due to its distributed nature and because it records asset transfers, the blockchain system is also commonly referred to as a distributed ledger. During the blockchain system execution, nodes generate new transactions and exchange them through the network. These transactions are grouped into blocks by special nodes, called *miners* and miners then propose blocks to other nodes. Miners try to agree on a unique block to append it to the chain. If they succeed, the current state of the blockchain indicates how digital assets have been transferred between accounts.

We consider a communication graph $G = \langle V, E \rangle$ with nodes or processes V connected to each other through fixed communication links E. (Notation $\langle v_1, ..., v_k \rangle$ of objects $v_1, ..., v_k$ represents a *tuple* of these objects.) Processes are part of a blockchain system S. Processes can act as clients by issuing transactions to the system and/or servers by *mining*, the action of trying to combine transactions into a block. For the sake of simplicity, we consider that each process possesses a single account (or *address*) and that a *transaction* issued by process p_i is a transfer of digital assets or *coins* from the account of the source process p_i to the account of a destination process $p_j \neq p_i$. Each transaction is uniquely identified and broadcast to all processes in a best-effort manner.

Algorithm 1 Blockchain construction at node p_i

1: $\ell_i = \langle B_i, P_i \rangle$, the local blockchain at node p_i is a directed acyclic
2: graph of blocks B_i and pointers P_i

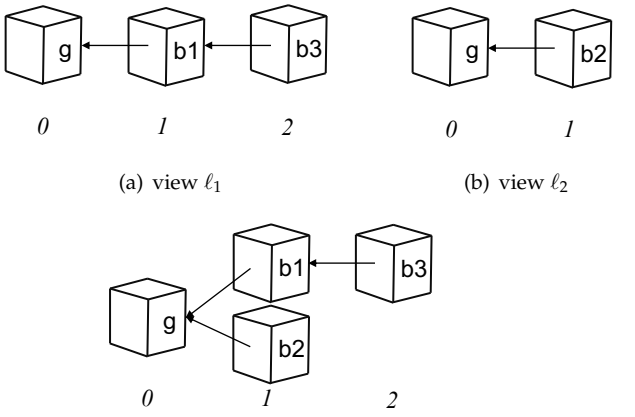
3: **Upon reception of blocks** $\langle B_j, P_j \rangle$**:**
4: $B_i \leftarrow B_i \cup B_j$
5: $P_i \leftarrow P_i \cup P_j$

Algorithm 2 describes the progressive construction of a blockchain at a par-

ticular node p_i upon reception of blocks from other nodes by simply aggregating the newly received blocks to the known blocks (lines 19–21). As every added block contains a hash (or digest) of a previous block that eventually leads back to the genesis block, each block is associated with a fixed index. By convention we consider the genesis block at index 0, and the blocks at j hops away from the genesis block as the blocks at index j. As an example, consider the simple blockchain $\ell_1 = \langle B_1, P_1 \rangle$ depicted in Figure 1.2(a) where $B_1 = \{g, b_1, b_3\}$ and $P_1 = \{\langle b_1, g \rangle, \langle b_3, b_1 \rangle\}$. (Notation $\{v_1, ..., v_k\}$ is used to denote a *set* of objects $v_1, ..., v_k$.) The genesis block g has index 0 and the block b_1 has index 1.

1.3 Double spending

This distributed nature of the blockchain implementation relies on the fact that different miners maintain their own replica of the blockchain, and these replicas typically diverge when some miners did not receive yet the data that others already received through the network. It is thus possible that the blockchain forks in which case distinct blocks are appended to the same block. This typically happens when each of two replicas append distinct blocks at the end of the same blockchain prefix. If the two branches contain transactions that consumes the same assets, then we call this a *double-spending*.



(a) view ℓ_1 (b) view ℓ_2

(c) global view $\ell_0 = \ell_1 \cup \ell_2$

Figure 1.2: The first view ℓ_1 (Fig. 1.2(a)) and the second view ℓ_2 (Fig. 1.2(b)) diverge, leading to a fork ℓ_0 (Fig. 1.2(c)) where distinct blocks are at the same index of the chain

1.3.1 Forks as disagreements on the blocks at a given index

As depicted by views ℓ_1 and ℓ_2 in Figures 1.2(a) and 1.2(b), respectively, nodes may have a different views of the current state of the blockchain. In particular, it is possible for two miners p_1 and p_2 to create almost simultaneously two different blocks, say b_1 and b_2. If neither block b_1 nor b_2 was propagated early enough to nodes p_2 and p_1, respectively, then both blocks would point to the same previous block g as depicted in Figures 1.2(a) and 1.2(b). Because network delays are hard to predict as we will explain in Chapter 4, node p_1 may create the block b_3 without hearing about b_2. The two nodes p_1 and p_2 thus end up having two different local views of the same blockchain, denoted $\ell_1 = \langle B_1, P_1 \rangle$ and $\ell_2 = \langle \{g, b_2\}, \langle b_2, g \rangle \rangle$.

We refer to the *global view* as the directed acyclic graph $\ell_0 = \langle B_0, P_0 \rangle$ representing the union of these local blockchain views, denoted by $\ell_1 \cup \ell_2$ for short, as depicted in Figure 1.2, and more formally defined as follows:

$$\begin{cases} B_0 &= \cup_{\forall i} B_i, \\ P_0 &= \cup_{\forall i} P_i. \end{cases}$$

A *fork* is a set of multiple edges pointing towards the same block. For example, the global view ℓ_0 includes a fork as there exist two edges $\langle b_1, g \rangle \in P_0$ and $\langle b_2, g \rangle \in P_0$ that point to the same block destination g. Figure 1.2(c) depicts this fork.

1.3.2 From forks to double spending

Consider now that the DAG that represents the union of all blockchain views has multiple blocks at the same index k of the chain as in Figure 1.2(c). Let Bob and Carole be two merchants who sell real goods in exchange of coins and let Alice be a malicious user that issued two properly signed transactions T_A and T'_A that were included in blocks b_1 and b_2, respectively, such that:

- T_A consists of Alice transferring all her coins to Bob whereas
- T'_A consists of Alice transferring all her coins to Carole.

It is easy to see that both transactions should not be accepted by the system, because if Alice transfers all her coins to Bob she has no coins left to transfer to Carole, and vice versa. We say that these transactions *conflict* because they cumulatively withdraw more coins than what the balance would allow. However, because the transactions are accepted in different branches of the DAG, it is possible for Bob to observe that the transaction T_A has been included in a block, without seeing transaction T'_A, say because Bob did not receive block b_2. Similarly, it is possible for Carole to observe that the transaction T'_A has been included in a block, without seeing transaction T_A, say because Carole did not receive block b_1. If this fork situation is maintained for long enough so that Bob and Carole take an irreversible actions as a result of these transactions—for example, shipping the supposedly paid good—then we say that Alice has double spent: she has been successful at spending the same coins twice.

1.3.3 How to avoid forks?

To avoid forks, one must guarantee the uniqueness of a block at any index of the chain. This can be achieved by having nodes agree on this unique block, a problem referred to as the consensus problem [LSP82]. Put into the blockchain context, the consensus problem is for *nodes* (or processes) of a distributed system to agree on one block of transactions at a given index of a chain of blocks. This consensus problem can be more precisely stated along three properties: *(i)* agreement: no two correct processes decided different blocks; *(ii)* validity: the decided block is a block that was proposed by one process; *(iii)* termination: all correct processes eventually decide. A protocol solving the consensus problem, as will be presented in Chapter 3, is necessary to guarantee that blocks are totally ordered, hence preventing concurrently appended blocks from containing conflicting transactions.

1.4 Conclusion

Double spending is a problem that stems from the distributed nature of a blockchain system. It allows a malicious user to spend the same assets in two conflicting transactions. An easy way to achieve this is to create forks that will fool merchants. In order to avoid such a fork situation, we will have to solve consensus, hence guaranteeing that nodes agree on a unique block per index of the chain.

1.5 Bibliographic notes

Malicious behaviors are generally modeled by an arbitrary or Byzantine failure model that is named after the problem of generals attempting to reach an agreement in the presence of traitors defined by Pease, Shostak and Lamport [PSL80]. This failure model is detailed in Chapter 2. The uniqueness of the block avoids forks that could otherwise allow an attacker to double spend its coins in two branches [Ros12]. Consensus is known to be needed in payment systems if distinct nodes can issue conflicting transactions [GKM+19]. The UTXO model is the model used by Bitcoin [**?**] that lists all unspent transaction outputs for each address whereby executing a transaction consists of consuming some UTXO and producing new ones. At any moment, the balance of an address can be retrieved by summing up all the UTXO of an address. The balance model as the one used by Ethereum [Woo15] does not use this notion of UTXO.

A way of achieving double spending consists of executing Finney's attack [Fin11] that consists of an attacker solo-mining a block with a transaction that sends coins to itself without broadcasting it before issuing a transaction that transfers the same coin to a merchant. When the goods are delivered in exchange of the coins, the attacker broadcasts its block to override the payment

to the merchant. The vector76 attack [vec11] consists of an attacker solo-mining after block b_0 a new block b_1 containing a transaction to a merchant to purchase goods. Once another block b_1' is mined after b_0, the attacker quickly sends b_1 to the merchant for an external action to be taken. If b_1' is accepted by the system, the attacker can issue another transaction with the coins spent in the discarded block b_1.

Bibliography

[Bac02] Adam Back. Hashcash - a denial of service counter-measure. Technical report, 2002.

[Dav83] Chaum David. Blind signatures for untraceable payments. *Advances in Cryptology*, 1983.

[DN93] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *Advances in Cryptology (CRYPTO)*, volume 740 of *LNCS*. Springer, 1993.

[Fin11] Hal Finney. Finney's attack, February 2011.

[GKM+19] Rachid Guerraoui, Petr Kuznetsov, Matteo Monti, Matej Pavlovič, and Dragos-Adrian Seredinschi. The consensus number of a cryptocurrency. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 307–316, 2019.

[IBM75] IBM. Data encryption standard, 1975.

[LSP82] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, July 1982.

[PSL80] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, April 1980.

[Ros12] Meni Rosenfeld. Analysis of hashrate-based double-spending, 2012.

[Sza97] Nick Szabo. Formalizing and securing relationships on public networks, 1997. http://szabo.best.vwh.net/formalize.html.

[Sza05] Nick Szabo. Bit gold. Technical report, 2005.

[vec11] vector76. The vector76 attack, August 2011.

[Woo15] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger, 2015. Yellow paper.