1 *Blockchain and Distributed Ledger Technologies*

CLAUDIO DI CICCIO

Introduction

Distributed ledger technologies (DLTs) have attracted significant attention in the last few years. They gained a noticeable momentum, particularly after the introduction of blockchains as a basic building block for the development of new cryptocurrencies and tokens. This opportunity opened up new research directions to support the modern economy with numerous possibilities to redesign and innovate the market in accordance with the digital revolution we are witnessing. However, these technologies are yet to prove in practice their capability to match all the dependability and security requirements imposed in the economic and banking sector. In this chapter, we will provide an overview of the technical features of DLTs (and of blockchains in particular), outlining their potential impact in the economic field (Box 1.1). We will first introduce the reader to their definition from a technical point of view, illustrate its core mechanisms and the guarantees they provide, and describe how these features are realised in a decentralised way. Finally, we will draw opportunities and challenges stemming from the adoption of this technology. We begin this journey with a synthetic definition.

As is typical of synthetic definitions, the one below can also be seen as simplistic yet complex to catch at first read. In an attempt to compensate for both issues, we will delve deeper into the fundamental notions and devices behind distributed ledger and blockchain technologies.

Transactions and Ledgers

The building block of blockchains is the **transaction**. A transaction is a digital record that registers the transfer of value (and data) between accounts. Once processed, the transaction triggers the movement of crypto-assets (also known as cryptocurrencies) such as Bitcoin (BTC)

https://doi.org/10.1017/9781009411783.003 Published online by Cambridge University Press

Box 1.1 Distributed ledger and blockchain

A distributed ledger is a registry replicated over a network of nodes that records the sequence of transactions between senders and recipients. A blockchain is a distributed ledger that uses blocks to collate sections of the ledger. DLTs, such as those underneath the blockchain platforms, are designed to guarantee properties that preserve the storage, exchange, and update of data, such as verifiability, liveness, robustness, and permanence.

or Ether (ETH). A transaction involves at least a sender account for the input and a recipient account for the output. Every transaction is digitally signed by the sender to show evidence that the account owner – and nobody else! – issued it. To do so, the owner retains a **private key**, with which only they can sign data, and a **public key** associated with the account address. Everyone can verify that the digital signature belongs to the owner through an automatic procedure based on the public key. As the account address is derived from the public key, the fact that the signee owns the account is also automatically verifiable. Please note, thus, that transactions do not require the personal details of the account holder to be known. The link between the signature and the account number is cryptographically guaranteed. In fact, the owner of an account can remain completely unknown within the blockchain. Nevertheless, all transactions report the account address of recipients and the senders. Therefore, one can trace all the transfers from and to an account. This setting thus guarantees *pseudonymity* within the blockchain platform, instead of complete anonymity.

Figure 1.1 illustrates a simplified example of a transaction. In the figure, a transfer of 2,000,000,000 units of cryptocurrency to account 0x1472…160c is requested from the 0xca35…733c account's owner. Considering the Ethereum[1] blockchain platform (Buterin 2014), the transferred value is expressed in Wei, and the amount in the picture equates to two Gwei (i.e., 0.000000002 Ether). The transaction is cryptographically signed by the sender to attest that the transaction is authentic. Every transaction has a unique ID and can bear additional information in the payload.

[1] See *Ethereum*. Available online: www.ethereum.org/ (accessed: 20/01/2023).

https://doi.org/10.1017/9781009411783.003 Published online by Cambridge University Press

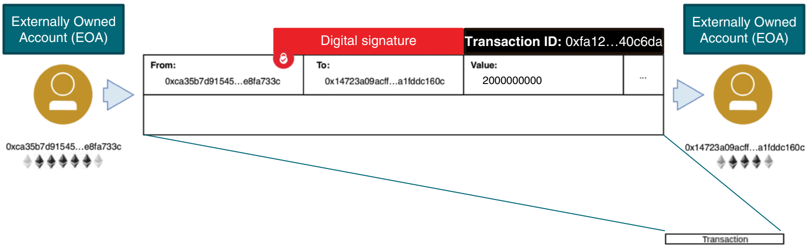

Externally Owned Account (EOA)
0xca35b7d91545…e8fa733c
Digital signature
Transaction ID: 0xfa12…40c6da
From:
0xca35b7d91545…e8fa733c
To:
0x14723a09acff…a1fddc160c
Value:
2000000000
...
Externally Owned Account (EOA)
0x14723a09acff…a1fddc160c
Transaction

Figure 1.1 A simplified example of Ethereum transaction
Note: The sender account (to the left) sends 2,000,000,000 Wei (i.e., 2 Gwei) to the recipient account (to the right)
Source: author's elaboration.

Ledger is a term that began being used in England during the fifteenth century to indicate a register of accounts. This term represents the collation of transactions. Notice the use of the word *collation* in place of *collection*: maintaining the order of transactions is crucial as it prevents the so-called double spending.

The following example, illustrated in Figure 1.2, aims to give an idea of what double spending is, and why the order of transactions is necessary to prevent it. Assume that the account 0xB belongs to a digital service provider, Bob. Accounts 0xA and 0xC belong to the same owner, Alice. Alice wants to purchase a digital package from 0xB at the price of 90 units of cryptocurrency (which we shall henceforth denote with ¢, so 90 ¢ here). Remember that the ownership of accounts (let alone the purposes of the owners) is a piece of information that is not recorded within the blockchain, that is it is *off-chain*. In the beginning, the balance of account 0xA amounts to 100 ¢, the balance of 0xB is 10 ¢, and the balance of 0xC is 50 ¢. Alice sends a transaction worth 90 ¢ to 0xB to purchase the digital product. Let us associate this transaction with the identifier 0xA90B. This operation reduces 0xA's balance from 100 to 10 ¢ and increases that of 0xB from 50 to 140 ¢. Bob, then, sends the digital product to Alice. Notice that the digital product is not shipped *on-chain* but *off-chain*. After receiving the package (which we assume is transmitted at very high speed), Alice tries to issue a new transaction from 0xA to 0xC, worth 50 ¢. Let us identify this transaction with 0xA50C. Since negative balances are typically not allowed in the blockchain, transaction 0xA50C is rejected – notice that 0xA's balance would drop to −30 ¢ otherwise.

https://doi.org/10.1017/9781009411783.003 Published online by Cambridge University Press

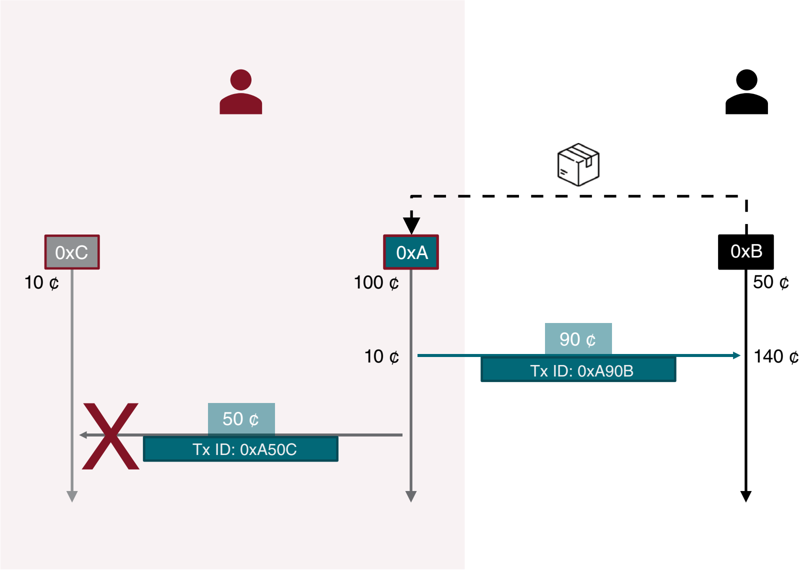


Figure 1.2a A double-spending scheme
Note: Account 0xA sends 90 units of cryptocurrency to buy a digital product, sent outside of the blockchain by the owner of the recipient account 0xB. Afterwards, a new transaction from the 0xA account of 50 units of cryptocurrency is rejected due to insufficient funds
Source: author's elaboration.

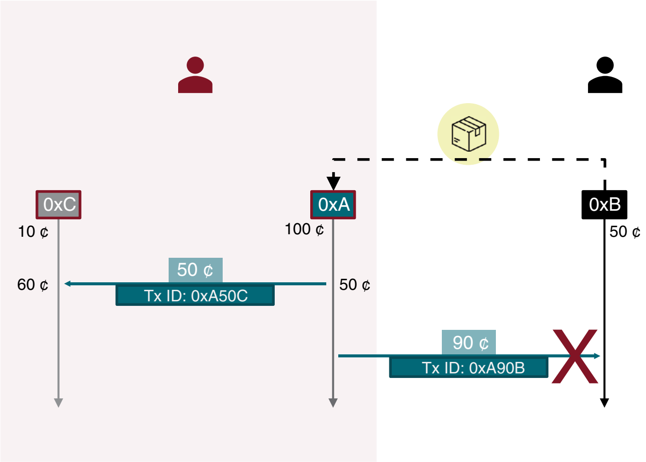


Figure 1.2b A double-spending scheme
Note: The effect of inverting the order of transactions reveals the double spending mechanism: the transaction paying 0xB is rejected, although the digital product has already been delivered
Source: author's elaboration.

https://doi.org/10.1017/9781009411783.003 Published online by Cambridge University Press

Here we get to the double spending issue. Alice could try to argue, once she has received the digital package, that transaction 0xA50C took place *before* 0xA90B. In that case, the latter would be rejected. As a result, Bob would have already delivered the service without being paid, whereas the total balance of Alice's accounts would remain intact as if the 90 ¢ of transaction 0xA90B were spent twice. We conclude that not only keeping transactions unaltered but also preserving their order is vital in this context.

Distributing the Ledger

If the ledger were saved on only one computer system, however much it can be secured, it would impose a question of trust: it is assumed that those to whom the administration and safeguarding of this system are delegated do not let the content be lost or destroyed, made invalid or corrupt, truncated thus bearing incomplete information, or altered with forged transactions. In our setting, this problem is overcome by ensuring that multiple copies of the ledger are saved in separate locations through intercommunicating systems on a computer network (in jargon, these systems are called */nodes/*), as illustrated in Figure 1.3. These nodes are entitled to the same rights on the data, so they are interchangeable. We name this paradigm peer-to-peer. In a peer-to-peer network, any node can crash, get offline or become unavailable for whatever reason: the other nodes will let the information safely stored. In theory, it is sufficient that just one node resists, and the whole history of transactions is preserved. The platform is resilient even to events such as nodes being under cyber threat or overtaken by malicious players. The ledger can still be considered safe

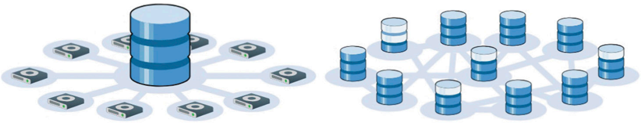


Figure 1.3 Centralised and distributed architectures
Note: The centralised architecture to the left illustrates a single system retaining the information and offering services to the other nodes in the network. The peer-to-peer network to the right shows a replica of information and services on all nodes. Notice, however, that replicas may not be identical
Source: author's elaboration.

https://doi.org/10.1017/9781009411783.003 Published online by Cambridge University Press

if the majority is still correctly functioning and properly behaving. A deliberate attack or malfunction must propagate on a large scale to take effect: on one copy (or on a limited number of copies), it does not affect the system. These are the main arguments in favour of a distributed solution to handle the ledgers. Hence the name: *distributed ledger technology*.

To keep the local copies of the ledger synchronised, all nodes should receive updates from the network. This requirement entails that every node being informed about an update should register it and bounce it to the neighbouring nodes in the network. It is noteworthy to recall that the updates pertain to transactions. Therefore, every node is aware of every transmission of value among accounts. From this standpoint, we may agree that keeping *pseudonymity* is a good compromise between the need for nodes to update the status of accounts (if they did not know from and to what accounts the transactions were issued, how could they keep track of the balances?) and the preferably avoidable situation in which all nodes know every detail about personal belongings and exchanges. Notice that if full anonymity was kept, nodes would not be aware of the transactions' sender and recipient accounts nor would they be able to reconstruct this piece of information. They could keep on piling up new transactions from the network but then who could guarantee that transactions were legitimate? For instance, who could verify that there were enough cryptos in account 0xA to send 90 ¢ to 0xB? An authority should be invoked to solve this conundrum, in case – which would dismantle the whole idea of decentralisation and restore the risks of data loss, corruption, and crashes.

Transactions are thus collated in ledgers, one after the other. As a consequence, ledgers tend to grow. Thinking about the old, paper-based one, it would seem natural to write down the ledger onto separate books. Books would also be sorted to ensure that the order of transactions is preserved across the books. In blockchains, the notion of a book is replaced by that of a *block*. A block contains a segment of the ledger plus additional heading information – including the timestamp.

To keep the order among blocks, each block is linked to the previous one through a one-way function called *hashing*. In brief, hashing is a mathematical one-way function that produces a number (the digest, also commonly known as hash) that works as a digital fingerprint for any piece of input data. The hash characterises the input data (just like a fingerprint identifies a person) though being typically

https://doi.org/10.1017/9781009411783.003 Published online by Cambridge University Press

of a fixed size (normally, smaller), regardless of the input data size. More than the fingerprint metaphor, though, the hash is tightly bound to the input data. The hashing function returns the same hash out of two identical pieces of data. Altering a bit in the data, though, turns the associated hash into a completely different number to the extent that it is not feasible to reconstruct what the alteration was by solely looking at the digest.

Equipped with this notion, we can see what happens when every block stores the hash of the previous one in its header. This scenario serves as a good intuition of the approach underlying Bitcoin[2] and Ethereum, among others. The actual mechanism is a bit more refined to save computation efforts, but we can omit the details for the sake of understandability. Let us consider three blocks in a sequence now, which we shall refer to as previous, current, and next. If we try to remove, add, change, swap, or reorder transactions in the previous block, its content is altered and, thus, its hash turns into something that is completely different. Consequently, the previous block's hash does not match the copy stored in the current one anymore. To keep it consistent, the current block has to change its local copy of the previous hash accordingly. This update, in turn, changes the header and hence the hash of the current block. The next block, then, has to modify its copy of the current block hash accordingly. At this stage, we can quickly figure out what happens to the block following the next one. We conclude that this mechanism makes every change reverberate as a sort of domino effect along the whole sequence of blocks. The older the changed block, the longer the domino effect.

The sequence of hash-based links thus forms a chain, as depicted in Figure 1.4. Hence the name, *blockchain*. DLTs such as Bitcoin and Ethereum are blockchain platforms as they employ the slicing of ledgers into consecutive blocks that are backlinked from the current to the previous one. IOTA,[3] instead, is a DLT that does not implement this approach. Blockchain platforms use blocks as packets transmitted within the chain to update the ledger with a new segment.

Transactions are broadcast by all nodes in the network to all their neighbours, so that every node can be aware of the fact that the transaction was issued. In blockchain platforms, they remain in a temporary

[2] See *Bitcoin*. Available online: www.bitcoin.org/en/ (accessed: 21/01/2023).

[3] See *Iota*. Available online: www.iota.org/ (accessed: 20/01/2023).

https://doi.org/10.1017/9781009411783.003 Published online by Cambridge University Press

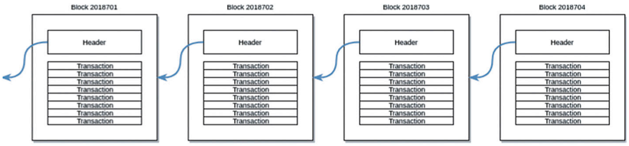


Figure 1.4 A schematised view of the backward link–based chain of blocks
Note: Every block contains a segment of the sequence of the transactions issued to that moment. The collation of transactions goes under the name of ledger
Source: author's elaboration.

storage named *transaction pool*, from which the publishing node collects the ones to include in the block. Until a block is published and accepted by the nodes in the network, indeed, the new transactions are not appended to the ledger.

The power in the hands of nodes publishing the blocks should not be overlooked: whether a transaction is included or not in a block depends on their choice. This is why *transaction fees* are usually included in the transactions by the senders: they are an economic incentive to motivate the miner to include the transaction in the next block.

Right to Publish and Consensus

We have already observed that decentralising the management and maintenance of a ledger strengthens the platform. However, this solution comes with a few infrastructural drawbacks. Firstly, the nodes must send digital messages through the network to keep the nodes updated with the latest transactions. These messages are prone to possible delays or complete loss. As a result, different nodes may have diverging views of the historical sequence of transactions. Secondly, the emission of the updates should be granted to nodes that give evidence of their reliability. Otherwise, malicious nodes could find it too easy to attack the network by flooding it with data reporting wrong or false information. Therefore, integrating at least two mechanisms appear fundamental to preserving safety and operational continuity. One should guarantee that the network eventually achieves a univocal view of the ledger (consensus). The other should cater for self-certification, ensuring that new messages are propagated only by nodes demonstrating their reliability. Terms like Proof

https://doi.org/10.1017/9781009411783.003 Published online by Cambridge University Press

of Work or Proof of Stake fall in the second category. Let us begin with the former, as it was historically introduced first and is still in use with platforms such as Bitcoin.

Proof of Work

The idea behind Proof of Work is that the nodes which aim to publish a new block should provide evidence of their will to keep the infrastructure in operation. To this end, those nodes show that they are ready to put computation time and resources at risk by attempting to solve a complex cryptographic puzzle. The solution to this puzzle is easily verifiable once it is given: finding it is the source of the difficulty. More specifically, Bitcoin's Proof of Work requires nodes to find a number (called a *nonce*) to be injected in the header of the block so that the hash of the block's header is a number that is lower than a given *target*. For example, the miner of the Bitcoin block number 769,424 inserted 2,927,826,006 as the nonce to make the whole block header's hash equal to 310,844,154,145,111,873,655,715,160, 191,695,224,044,144,394,078,051,380, which is less than the set target (762,342,638,057,996,256,581,733,267,702,136,683,580,848, 909,336,969,216).[4]

Recall that reverse-engineering the input of a hashing function given the output is nearly impossible. Therefore, the only way to find the nonce is adopting a brute-force approach: try all possible numbers until the hash of the whole block is right (i.e., less than the target). The challenge is already hard on its own but, to put more pressure on the nodes, it is an open race: if another node finds a suitable nonce first, the challenge for that block is over, and a new round starts with the next block. This approach is in line with the distributed computation scheme: any node in the network can concurrently run its own operations to be entitled to the right to publish the block. To make things worse, so to speak, we should consider that the previous computations do not lead to any advantage for the subsequent round. The new block is indeed different from the previous, so the nonce is to be inserted in a different data box. As a result, the new hash has to be recomputed from scratch for every possible candidate. Finally, notice that

[4] See *Blockchair*. Available online: www.blockchair.com/bitcoin/block/769424 (accessed: 20/01/2023).

https://doi.org/10.1017/9781009411783.003 Published online by Cambridge University Press

the *difficulty* of the puzzle is set by the target: intuitively, the lower the target, the fewer the numbers that are below it. These numbers form the set of acceptable block-headers' hash values. Every such hash value roughly corresponds to a nonce. Therefore, fewer nonces are there to be guessed, and finding one becomes a tougher challenge. The difficulty is thus *tunable*. The need to keep the average time to publish a block stable and around 10 minutes determines how the knob is turned. If the average publication time is lower than that, the difficulty gets increased. Otherwise, it is lowered. Notice that this decision is not governed by any single actor: the protocol lets every node know how to autonomously determine the necessary change at regular intervals of 2,016 blocks (about 2 weeks).

At this stage, a doubt could legitimately arise: what moves the nodes to try and guess the nonce, given the required considerable efforts in spite of no guarantee to win the game? To pay the electricity bill and the hardware consumption back, the winners who manage to publish the block included in the blockchain are rewarded with freshly minted cryptocurrency (plus other non-negligible extras such as the transaction fees, discussed earlier). As such a prize is akin to finding gold behind a stone wall of a cave after extenuating excavations, we say that nodes publishing blocks (or trying to do it) are *mining* nodes. At the time of writing, the *mining reward* amounts to 6.25 BTC. Cryptocurrencies therefore perform the function of *cryptofuel*, which supplies monetary resources to the nodes that maintain the infrastructure using their own computational resources. Cryptofuel attracts miner nodes as cryptocurrencies are traded in fiat money on dedicated markets by investors around the world. At the time of writing, the mining reward equates to about 20,000 €.

Notice that the mining reward was established in May 2020 and is getting halved every 210,000 blocks (i.e., about four years: in 2009, it amounted to 50 BTC). Quoting the Bitcoin white paper: 'Once a predetermined number of coins have entered circulation, the incentive can transition entirely to transaction fees and be completely inflation free.' (Nakamoto, 2008) The incentive can help encourage knots to remain honest. Quoting the white paper again: 'A greedy attacker [...] ought to find it more profitable to play by the rules, [as] such rules [...] favour [them] with more new coins than everyone else combined' (Nakamoto 2008).

Verifying that the solution to the puzzle is correct has to be relatively easy because every node in the network should be capable of

https://doi.org/10.1017/9781009411783.003 Published online by Cambridge University Press

validating it: if the solution is incorrect, the block should be rejected. Other reasons for a proposed block to be discarded include incorrect transactions, wrong signatures, or excessively distant timestamps reported in the block header. We remark that these checking operations are carried out virtually by every node in the network.

Proof of Stake

Proof of Work has been subject to criticism due to the high consumption of electricity, heating, and ultimately pollution that mining nodes cause. With regular periodicity, new reports are published on the comparison between the power consumption of entire states and that due to mining. The trend has long been upward, that is comparable states increase in size at every update. Furthermore, in an attempt to win the puzzle, larger and larger mining rigs have been assembled. Mining rigs consist of a multitude of machines equipped with dedicated hardware that have the sole objective of running the mining processes in parallel to increase the chances to win. As a consequence, the risk of re-centralising the decision process becomes more and more tangible.

To overcome this issue, Ethereum transitioned to a different approach: the Proof of Stake, already applied natively in other blockchain platforms such as Algorand.[5] In Proof of Work, miners put capital at risk by expending energy. In Proof of Stake, *validators* explicitly stake capital in Ether, the Ethereum platform's cryptocurrency. More specifically, candidates propose themselves as validators by depositing a given amount (32 ETH, at the time of writing) from their balance: this amount is referred to as the *stake*, indeed. The stake cannot be used by the original owners as long as they remain in the role of validators. Validators are pseudo-randomly selected to become part of validator *committees* (currently) of 128 members each. Within the committee, one node is chosen as the *proposer*. The proposer replaces the role of the miner as block publisher. The members of a validation committee vote for (i.e., they broadcast their *attestation* to) the next block to be put at the head of the chain. For every published block that gets included in the blockchain, both the proposer and the validators get a *reward* that is topped up to the stake.

[5] See *Algorand Foundation*. Available online: www.algorand.foundation/ (accessed: 20/01/2023).

https://doi.org/10.1017/9781009411783.003 Published online by Cambridge University Press

With Proof of Stake, the block time is fixed in Ethereum (set to 12 seconds at present). The block time is named /*slot*/. The first half of the slot is the time the proposer has to submit the new block. Every 32 slots (an /*epoch*/, i.e., 6 minutes and 24 seconds currently), committees are drawn from the set of members with sufficient ETH at stake, one per slot and so that no two committees share any members in the epoch.

The *finality* of blocks is handled explicitly by the protocol: the first block published in every /epoch/ requires an additional vote from the committees. The decision is two-staged. The committee votes for pairs of epoch-boundary blocks: an older one (source) and a newer descendant (target). Both source and target need to be attested by two-thirds of the votes to be bound by a so-called *supermajority link*. The target in the supermajority link becomes *justified*. The source is typically already justified (as it was the target in a previous voting round) and thus becomes *finalised*. Validators attesting to blocks that are included in a supermajority link receive specific extra rewards for justified and finalised blocks. A finalised block is nearly impossible to be removed from the net. Notice that not all attestations have the same weight. The weight depends on the quota left at stake – in particular, the so-called *effective balance*, which cannot amount to more than 32 ETH. Notice that the staked ETH can also decrease.

With this scheme, indeed, inactive members could hamper the process. This is why the protocol includes the so-called *inactivity penalties*. Penalties remove limited amounts of the capital at stake as errors, temporary disconnection from the network, or seldom malfunctions can happen, after all. What is treated with a more severe countermeasure is the occurrence of (allegedly malicious) misbehaviour: nodes that publish multiple blocks (/*equivocation*/) attest to different blocks (*double vote*) in the same round, or vote for a source and a target that occur in epochs surrounding an already voted pair, are subject to *slashing* of their staked funds. Notice that slashing leads to the removal of the member from the set of validators. Interestingly, conditions for slashing can only be verified if other nodes signal and report evidence of them. This is why a special reward is dedicated to *whistleblowers* (who notify the misbehaviour) and proposers who include the whistleblowing messages in the block. Also, notice that the slashed funds become higher if the misbehaving player operates in collusion with other players. The reason why finalised blocks are considered as such lies in the enormous slashing that their replacement in the

https://doi.org/10.1017/9781009411783.003 Published online by Cambridge University Press

blockchain would require: as two-thirds of the validators have voted for it, there cannot be another two-thirds of validators attesting to another block unless one-third of the total were double votes.

Choosing the Fork

As we said, the network is subject to delays and loss of blocks. It is perfectly reasonable that, at some stage, a node observes that more alternatives are coming as an update – different blocks that are valid but report on different transactions.

The temporary situation in which more branches are taken into account as possible evolutions of the chain is typically named **fork**. Forks can happen with the top, most recent blocks. The further we go down the chain, the less likely they become. However, notice that if a block is included in the chain, its transactions are appended to the ledger and thus remain in the collective memory of the network. If a block is initially considered part of the chain but then gets replaced by another sub-chain, its transactions become non-existing all of a sudden (unless, of course, the new chain still retains that very block). Owing to this, it is recommended to wait for six blocks to be appended in Bitcoin before considering the transaction as finalised.

How to determine the fork to include in the blockchain in a way that eventually all nodes opt for the same one? The mechanism underneath this choice is the basis of consensus. With Proof of Work, the preference leans towards the block at the head of the sub-chain with the highest amount of work put in the mining – which translates with good approximation to the longest sub-chain. Every block received by a node brings with itself the hash-based link to the previous one. If the node observes that the new block does not appear to have the predecessor in the chain, it keeps the new block aside and waits until the predecessor is delivered from the network. Notice that the same process could occur with the predecessor's predecessors, and so on, until a convergence point is found. At that moment, the node can decide which branch to take as the main chain. In Ethereum, the sub-chain that accumulates the highest weight of attestations is chosen.

We remark that this choice is made by all nodes and recall that all nodes are in charge of individually verifying that the block they receive is correct, contains valid transactions, and is consistent with the remainder of the history of the blockchain. For practical reasons,

https://doi.org/10.1017/9781009411783.003 Published online by Cambridge University Press

though, some nodes may not store the whole ledger locally, but only a section of their interest. These nodes are named *light nodes*. A typical example of light node is a *wallet*, that is the software that account holders use to keep track of their funds in cryptocurrencies.

This aspect offers food for thought about the nature of blockchain as a platform (Voshmgir 2019). It is politically decentralised because no entity controls the network. It is architecturally distributed because the information is held and managed by all nodes in the system. However, it is logically centralised because each entity has its own copy of the ledger, in a state that tends to be unanimously agreed.

Transaction Model and Balance Model

Not only the consensus mechanisms distinguish Bitcoin and Ethereum. Another key specificity lies in the paradigm with which the two blockchain platforms handle the transfer of value among accounts. Ethereum adopts a *balance model*: accounts are associated with their current amounts of Wei's, which they can spend by adding a value to the transaction. If account 0xE owns, for example, three ETH, its owner can sign a transaction with which they send 1.5 ETH to account 0xF. Therefore, the nodes in the network should keep track of the current balance of every account to verify whether they can spend the declared amount in a transaction or not. A metaphor for the balance model used in Ethereum is that of the bank transfer: akin to Ethereum transactions, bank transfers typically report the account coordinates of the sender, the account coordinates of the redeemer, the transferred amount, the transaction fee, and a unique identifying number for the transaction. When executed, they determine the subtraction of the indicated amount and the fee from the sender's account and the addition of the amount in the recipient's account. The transaction fee goes to the issuing bank. Notice that in blockchain platforms, banks do not constitute the underlying organisation guaranteeing for the safety and security of the transfers and accounts. The whole network does, thus network nodes (specifically, the block proposers in Ethereum) are rewarded the transaction fees for their efforts.

Bitcoin, instead, uses a *transaction model*. Suppose that account 0xB had received a transaction worth 5 BTC, another one of 6 BTC, and a third one of 1 BTC in the past. The balance amounts to 12 BTC in total, which corresponds to the sum of the denominations of its Unspent Transaction Outputs (UTXOs). Imagine that the owner of account 0xB

https://doi.org/10.1017/9781009411783.003 Published online by Cambridge University Press

wants to send 11.5 BTC to 0xC. To do so, the owner should sign the three UTXOs we mentioned, setting 0xC as the redeemer, and wrap them into a new transaction. UTXOs cannot be split into sub-units. To get the difference in return, the new transaction should include a new transaction unit that transfers 0.5 BTC (or less) from 0xC back to 0xB. If the change amounts to less than 0.5 BTC (say, 0.4 BTC), the difference becomes the *transaction fee* to the benefit of the miner (0.1 BTC in this case). In this setting, the balance of every account is computable by summing up all the UTXOs that belong to it. Considering that account addresses mark every transaction in which they occur as sender or redeemer, UTXOs can be tracked for all their lifetime. As a good metaphor for UTXOs, we can consider transferable cheques that are not redeemed.

Smart Contracts

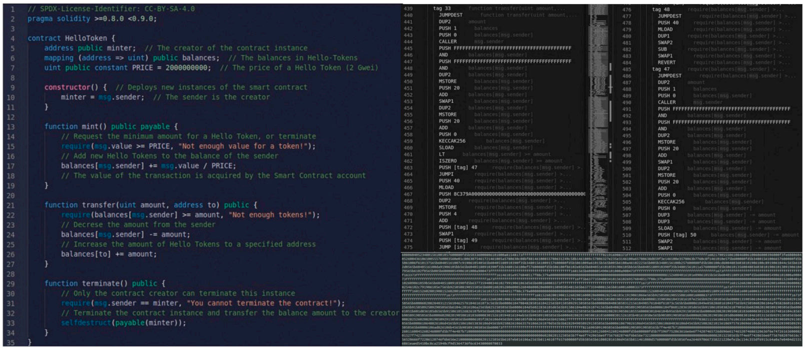
A second generation of blockchain arose when, from the intuition of Vitalik Buterin, the focus shifted from the concept of blockchain as a distributed system for the exchange of electronic money to the concept of blockchain as a */programmable distributed environment/* (Buterin 2014). The notion through which this conceptual leap became possible is the *smart contract*.

The smart contract is a program run by the blockchain platform. Programmers typically use a coding language such as Solidity,[6] which is later automatically turned into a set of low-level operations executable by computerised systems. Figure 1.5 shows the same smart contract written in Solidity (left) and turned into the so-called operation codes, or opcodes for short (top right in the figure), for execution. At the bottom-right corner of the figure, we can see the direct transposition of the instruction codes into binary codes that the delegated component of Ethereum named Ethereum Virtual Machine (EVM) executes. One of the crucial characteristics of smart contracts is that their code, and only their code, fully determines how their status evolves ('*The code is the law*').

Smart contracts, despite their name, are not necessarily linked to a binding contract between counterparts. Of course, they can *also* represent contracts in the most commonly adopted sense. However,

[6] See *Solidity*. Available online: www.solidity.readthedocs.io/ (accessed: 20/01/2023).

https://doi.org/10.1017/9781009411783.003 Published online by Cambridge University Press



```
// SPDX-License-Identifier: CC-BY-SA-4.0
pragma solidity >=0.8.0 <0.9.0;

contract HelloToken {
    address public minter;  // The creator of the contract instance
    mapping (address => uint) public balances;  // The balances in Hello-Tokens
    uint public constant PRICE = 2000000000;  // The price of a Hello Token (2 Gwei)

    constructor() {  // Deploys new instances of the smart contract
        minter = msg.sender;  // The sender is the creator
    }

    function mint() public payable {
        // Request the minimum amount for a Hello Token, or terminate
        require(msg.value >= PRICE, "Not enough value for a token!");
        // Add new Hello Tokens to the balance of the sender
        balances[msg.sender] += msg.value / PRICE;
        // The value of the transaction is acquired by the Smart Contract account
    }

    function transfer(uint amount, address to) public {
        require(balances[msg.sender] >= amount, "Not enough tokens!");
        // Decrese the amount from the sender
        balances[msg.sender] -= amount;
        // Increase the amount of Hello Tokens to a specified address
        balances[to] += amount;
    }

    function terminate() public {
        // Only the contract creator can terminate this instance
        require(msg.sender == minter, "You cannot terminate the contract!");
        // Terminate the contract instance and transfer the balance amount to the creator
        selfdestruct(payable(minter));
    }
}
```

```
tag 33      function transfer(uint amount,...
  JUMPDEST      function transfer(uint amount,...
  DUP2      amount
  PUSH 1      balances
  PUSH 0      balances[msg.sender]
  CALLER      msg.sender
  PUSH FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  AND      balances[msg.sender]
  PUSH FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  AND      balances[msg.sender]
  DUP2      balances[msg.sender]
  MSTORE      balances[msg.sender]
  PUSH 20      balances[msg.sender]
  ADD      balances[msg.sender]
  SWAP1      balances[msg.sender]
  DUP2      balances[msg.sender]
  MSTORE      balances[msg.sender]
  PUSH 20      balances[msg.sender]
  ADD      balances[msg.sender]
  PUSH 0      balances[msg.sender]
  KECCAK256      balances[msg.sender]
  SLOAD      balances[msg.sender]
  LT      balances[msg.sender] >= amount
  ISZERO      balances[msg.sender] >= amount
  PUSH [tag] 47      require(balances[msg.sender] >.
  JUMPI      require(balances[msg.sender] >...
  PUSH 40      require(balances[msg.sender] >...
  MLOAD      require(balances[msg.sender] >...
  PUSH 8C379A000000000000000000000000000000000000000000
  DUP2      require(balances[msg.sender] >...
  MSTORE      require(balances[msg.sender] >...
  PUSH 4      require(balances[msg.sender] >...
  ADD      require(balances[msg.sender] >...
  PUSH [tag] 48      require(balances[msg.sender] >..
  SWAP1      require(balances[msg.sender] >...
  PUSH [tag] 49      require(balances[msg.sender] >..
  JUMP [in]      require(balances[msg.sender] >...
```

```
tag 48      require(balances[msg.sender] >...
  JUMPDEST      require(balances[msg.sender] >...
  PUSH 40      require(balances[msg.sender] >...
  MLOAD      require(balances[msg.sender] >...
  DUP1      require(balances[msg.sender] >...
  SWAP2      require(balances[msg.sender] >...
  SUB      require(balances[msg.sender] >...
  SWAP1      require(balances[msg.sender] >...
  REVERT      require(balances[msg.sender] >...
tag 47      require(balances[msg.sender] >...
  JUMPDEST      require(balances[msg.sender] >...
  DUP2      amount
  PUSH 1      balances
  PUSH 0      balances[msg.sender]
  CALLER      msg.sender
  PUSH FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  AND      balances[msg.sender]
  PUSH FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  AND      balances[msg.sender]
  DUP2      balances[msg.sender]
  MSTORE      balances[msg.sender]
  PUSH 20      balances[msg.sender]
  ADD      balances[msg.sender]
  SWAP1      balances[msg.sender]
  DUP2      balances[msg.sender]
  MSTORE      balances[msg.sender]
  PUSH 20      balances[msg.sender]
  ADD      balances[msg.sender]
  PUSH 0      balances[msg.sender]
  KECCAK256      balances[msg.sender]
  PUSH 0      balances[msg.sender]
  DUP3      balances[msg.sender] -= amount
  DUP3      balances[msg.sender] -= amount
  SLOAD      balances[msg.sender] -= amount
  PUSH [tag] 50      balances[msg.sender] -= amount
  SWAP2      balances[msg.sender] -= amount
  SWAP1      balances[msg.sender] -= amount
```

Figure 1.5 The Solidity code of a sample smart contract in Ethereum (Hello Token)
Source: author's elaboration.

the spectrum of possible usages is much larger as, in Ethereum, they can express anything a computer program could do. For example, smart contracts are typically used to manage the life cycle of tokens such as the Hello Token in Figure 1.5. The Hello Token contract offers four operations: (1) the constructor function to deploy new instances of the smart contract and record the account address of the creator; (2) the mint function to buy new tokens at the price of 2 Gwei each; (3) the transfer function to send tokens from the sender account to another account; and (4) the terminate function to cease the operations of the smart contract and transfer the funds from the account of the smart contract to the creator's account. Notice that the way in which the smart contracts are encoded fully and exclusively dictates the way in which they behave and manage the tokens they define. The challenge of creating smart contracts that precisely and consistently represent the intended requirements and purposes is among the core challenges posed by the research agenda of Magazzeni et al. (2017).

Next to the so-called *externally owned accounts* (EOAs), owned by human users or in any case from entities outside the ecosystem of the blockchain platform, we have the smart contract accounts (CAs), that is accounts entirely managed by the aforementioned programs. Notice that the status of a smart contract includes the

https://doi.org/10.1017/9781009411783.003 Published online by Cambridge University Press

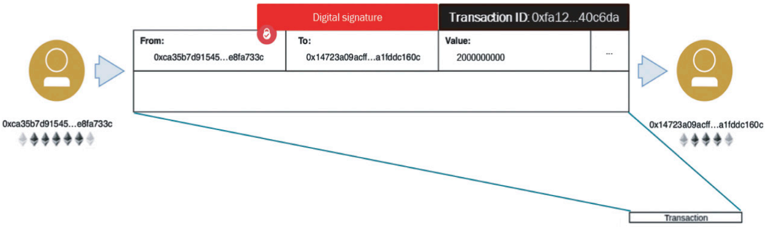

Digital signature
Transaction ID: 0xfa12...40c6da
From:
0xca35b7d91545...e8fa733c
To:
0x14723a09acff...a1fddc160c
Value:
2000000000
...
0xca35b7d91545...e8fa733c
0x14723a09acff...a1fddc160c
Transaction

Figure 1.6 Invocation of a smart contract
Source: author's elaboration.

balance of its own account, then. The code, therefore, determines how the funds therein are to be transferred or kept. However, the smart contracts solely react to calls from other accounts: they do not independently trigger their operations. This entails that they cannot independently take action at some stage on their own free will, so to speak: there always is an invoker that is responsible for triggering the operations.

Smart contracts, in other words, offer functions. Functions are invoked by other accounts (let them be EOAs or CAs) to execute operations. Updating the bank-transfer metaphor we used earlier to support this new paradigm, imagine that the recipient of the transfer is the smart contract, and the message encoded the data necessary to perform the requested operation. When the recipient receives the bank transfer, it automatically starts executing the requested operations based on the payload of the message. Indeed, the means to invoke smart contracts from EOAs is using *transactions*, as illustrated in Figure 1.6 (Box 1.2). As a consequence, all operations are fully tracked, the signee is known, and the outcome too (because the smart contract's code determines it). Notice that the code, although at its low-level representation in terms of bytecode, is publicly known to every node in the network. The reason is, new instances of a smart contract are deployed (i.e., assigned an account and initialised) by means of transactions too. The recipient is a previously unoccupied account address (which will become their own) and the code is in the payload. The payload (i.e., the *code* of the new instance of smart contract just deployed) is part of the transaction, which is going to be stored in the ledger and, therefore, known to every node in the network and immutable.

https://doi.org/10.1017/9781009411783.003 Published online by Cambridge University Press

Box 1.2 Smart contracts

If anything, *immutability*, *non-repudiability,* and *persistence* are good reasons to resort to smart contracts to coordinate the operations of *multi-party processes*, especially in the presence of partial trust among the participants (Mendling et al. 2018). All operations are tracked and cannot be erased, stored together with the smart contract that they testify the interactions with, in a permanent ledger that is replicated over all nodes. Therefore, the automated workflow is known to all participants, it is not going to change, and the ongoing evolution of the running processes can be monitored (Di Ciccio et al. 2022).

The outcome of every invocation is known to every node in the network because virtually every node will execute the triggered operations after they read the invoking transaction in the published block – hence, after the proposer node has actually pre-executed and validated them: since the proposer is responsible for grouping transactions into a block, it must be the first to verify their correctness. The operations of a smart contract function are thus going to be run multiple times across the whole network, with the same input and the same output, akin to a polygraph machine as metaphorically described by Dannen (2017). Therefore, their execution is associated with a price for the invoking account. The paid units go under the name of gas. Every opcode low-level operation (see the listing in Figure 1.5 at the top-right corner) is bound to a specific price in gas. Typically, altering the state of the smart contract and deploying new smart contracts are the most expensive operations, whereas mathematical and hashing operations have a lower cost. A full pricing list can be found in the Ethereum yellow paper (Wood 2014). Deploying a smart contract like the one in Figure 1.5 has a cost of 531,255 units of gas, invoking its mint operation to buy four Hello Tokens requires 43,821 units of gas. Also, the amount of data sent to the smart contract and treated therein are mirrored in additional costs: the more the data, the higher the overall price. Transaction fees in Ethereum are solely based on gas. Transactions with no smart contract invocations require gas, too. Notice that the gas is independent of the value we send. Therefore, to buy four Hello Tokens, we should send eight Gwei to the smart contract backing the token, in addition to the gas expenditure (for further details on the difference between tokens and

https://doi.org/10.1017/9781009411783.003 Published online by Cambridge University Press

cryptocurrencies, see Box 1.3). To make sure that the invocation terminates and no harmful contract drains unlimited funds from the invoking account, the senders set a limit to the maximum amount of gas they are ready to pay (the gas limit).

The price of every unit of gas in units of cryptocurrency (Wei) oscillates depending on two factors, both to be multiplied by the units of gas consumed to run a smart contract's function code: a priority fee, which the invoking account can raise at their will to increase the chances that the proposer will include that transaction in the next block, and a base fee, decided by the protocol and used to counterbalance the use of the network (the more the traffic, the higher the base fee). Notice that the priority fee (times the units of gas) will end up in the account of the proposer, whereas the base fee (multiplied by the units of gas) is going to be burnt. Ethereum, thus, allows for a deflationary mechanism unlike Bitcoin and permits units of cryptocurrency to be not only issued but also removed from the network.

Box 1.3 Token versus cryptocurrency

Notice the difference between a token and a cryptocurrency. Cryptocurrencies like ETH or BTC propel the infrastructure. Their emission (or burning in the case of Ethereum) depends on the protocol and its policies. Cryptocurrencies can be minted or exchanged for fiat money and sold back. This trading will determine their price in the market. Aside from this, there is hardly any other operation that can let users influence the nature of cryptocurrencies. Tokens, instead, are digital entities whose existence is bound to smart contracts supporting them. Their lifecycle, emission, exchange, and deletion are fully controlled by the backing smart contracts. Unlike cryptocurrencies, the rules of the game governing every token are fully determined by the custom code of which the smart contract consists. The infrastructure keeping up smart contracts (and tokens) is propelled by cryptocurrencies, and tokens are acquired from them in exchange for cryptocurrencies. Thus, they lie at different levels of abstraction in what computer scientists call the protocol stack, so to speak: cryptocurrencies are on a layer below contracts. **Smart contracts and tokens** are thus the main instruments to creating new blockchain-based projects for the **automation of services and processes**.

https://doi.org/10.1017/9781009411783.003 Published online by Cambridge University Press

To make an example from the real world, we can acquire points in the context of a fidelity program for customers when we buy a product or service from a given company. We buy products and services with money we can transfer with our credit card from our bank account, or withdraw at an Automated Teller Machine (ATM). The rules determining the market value of the currency of our bank account are not easily customisable by individuals. Also, we cannot typically deposit fidelity points in our bank account. Fidelity programs, however, are fully controlled and manageable by their company. Fidelity points, therefore, are akin to tokens.

The purpose of tokens is typically classified according to their *fungibility*. Tokens are fungible whenever individual units can be mutually substituted as they are indistinguishable. For instance, the Hello Token in Figure 1.5 is fungible. If a token represents an individual entity, with its own characteristics, different to another one and thus not interchangeable, it is *non-fungible*. Tokens representing the ownership of a physical or digital good, an artwork, or a digital identity, are all examples of non-fungible tokens. *Semi-fungible* tokens are fungible until a given expiry date or in case of a change of status like their redeeming, after which they become non-fungible (like tickets for concerts or vouchers).

Blockchain is often compared to the internet. The comparison concerns the current potential and the possible development that is obtained from it. In the beginning, the internet itself was an experimental project that required implementation effort by highly skilled software developers. However, its revolutionary nature has meant that its adoption has spread to the levels we know today. Blockchain and DLTs in general seem to follow this path. In fact, web services can integrate the blockchain. The effect has reduced visibility to the end user, through. Since its inception, the web has been based on a paradigm of clients and servers, with the former requesting services from the front-end (accessible by the user) to the latter, on the back end. In its first generation, the front end was delegated minor operations other than displaying and formatting the content provided by the servers. With the advent of Web 2.0, there has been a migration and outsourcing of operations from the back end to the front-end side, thanks to an increase in the capabilities of desktop PCs, laptops, handheld devices, etc. At the same time, although not evident on the user side, web servers have also delegated operations and data processing

https://doi.org/10.1017/9781009411783.003 Published online by Cambridge University Press

to specific management systems (e.g., on the cloud). The new *Web 3.0* paradigm integrates blockchain platforms on the back-end side, taking advantage of smart contracts for carrying out (part of the) operations though maintaining a front-end that does not impact the user interface. Applications that adopt the Web 3.0 paradigm are commonly known as *Decentralised Applications (DApps)*.

Thus far, we have observed characteristics and traits that are typical of blockchains that are publicly accessible and in which every node can potentially contribute to the validation and publishing of new blocks. However, there are other approaches to DLTs that restrict the aforementioned openness criteria. We briefly examine them next.

Public, Private, Permissioned, and Permissionless

Blockchains are often classified according to two criteria: *transactability* (or visibility) and *consensus* (Bellia et al. 2019). Table 1.1 displays the four categories that stem from these criteria. If a blockchain is *public*, every node can make transactions and view them. If only selected nodes are entitled to this right, it is *private*. If any node in the network can participate in the consensus decision-making process, the blockchain platform is *permissionless*. Instead, if only specific nodes can determine the next status of the blockchain, it is *permissioned*. Bitcoin, for example, is public and permissionless like Ethereum, natively. On the other side of the spectrum, Hyperledger Fabric[7] (Gaur et al. 2020) is an example of a private permissioned blockchain. The LTO network[8] offers a private though permissionless blockchain. EOSIO[9] is an example of a public blockchain platform with a permissioned consensus system.

Choosing the right combination (public or private? Permissioned or permissionless?) is a design choice that is up to the system architect based on the application context and foreseen use. Several decision models have been proposed to determine whether taking the blockchain as a building block is appropriate in the first place and, if so,

[7] See *Hyperledger*. Available online: www.hyperledger.org/use/fabric (accessed: 20/01/2023).

[8] See LTO network. Available online: www.ltonetwork.com/ (accessed: 20/01/2023).

[9] See EOSIO. Available online: www.eos.io/ (accessed: 20/01/2023).

https://doi.org/10.1017/9781009411783.003 Published online by Cambridge University Press

Table 1.1 ***Blockchain types according to their visibility and consensus mechanisms***

		Transactability/visibility	
		Private	*Public*
Consensus	*Permissionless*	**Selected** nodes can transact and view, **every** nodes can participate in consensus	**Every** node can transact and view, participate in consensus
	Permissioned	**Selected** nodes can transact and view, or participate in consensus	**Every** node can transact and view, **selected** nodes participate in consensus

which type of blockchain should be adopted. A renowned example of such guidelines is provided by Wüst and Gervais (2018).

To sum up, blockchains are fit for purpose whenever storing the status of the system is necessary, multiple entities concur to provide information on the evolution of the system while not all of them are trustable, and no trusted third parties are fully available. Otherwise, other solutions, such as distributed or centralised databases, can be more appropriate.

If a blockchain is appropriate for the intents and purposes of the project, the policy on whether making the circle of information providers and validators open or restricted drives the decision on whether the blockchain should be public or private, and permissionless or permissioned, respectively.

Concluding Remarks

This chapter illustrated and discussed the key traits of DLTs and blockchain platforms, with a focus on the technical mechanisms underpinning their guarantees. Transactions, the fundamental building block, allow for the transfer of cryptocurrencies and invocation of smart contracts' functions. Digital signatures enable the authentication of transaction senders. The order of transactions is preserved by the sequential nature of ledgers. The distributed architecture ensures data persistence, as a replica of the ledger is saved virtually on every node. Hashing makes the blockchain robust, as it impedes changes in older transactions without altering the whole sequence of blocks that follow. Mechanisms such as proof-of-work and proof-of-stake ensure that the publishing rights of new blocks are given to nodes that prove their reliability. Consensus algorithms guarantee that the blockchain is eventually consistent in the replicas stored on the network nodes. Smart contracts make blockchain programmable and, among other things, able to support tokens.

Distributed ledger and blockchain technologies are not a panacea for all projects in information technologies and beyond. Their use is suitable for settings where multiple actors cooperate in a regime of partial (or lack of) mutual trust and in the absence of trusted third parties that can authoritatively exert control of the stored information. Entering the details of the core mechanisms and rationale thereof reportedly is a challenging task on its own, and their installation or integration requires the intervention of technically skilled experts. However, they offer a number of critical guarantees by design, including authentication, inviolability, full traceability of data, and robustness, availability, and customisability of service. Arguably, building a novel system that offers *ex novo* all that would be a highly challenging task, let alone in regimes of partial trust between the key actors. Notice that the observations made thus far refer to using distributed ledger and blockchain technologies as an information technology (IT) core infrastructure within larger projects integrated with business processes (Xu et al. 2019). Their potential is enormous and can be fully unleashed when their adoption goes beyond the mere exchange of cryptocurrencies. The next chapters of this book will document success case studies in which endeavours of this sort are carried out in the banking sector.

References

Bellia, M., Kounelis, I., Anderberg, A., Calès, L., Andonova, E., Sobolewski, M. 2019. *Blockchain now and tomorrow: Assessing multidimensional impacts of distributed ledger technologies*. Publications Office of the European Union, Luxembourg: European Commission, Joint Research Centre.

https://doi.org/10.1017/9781009411783.003 Published online by Cambridge University Press

Buterin, V. 2014. Ethereum: 'A next-generation smart contract and decentralized application platform'. White paper. Available online: https://ethereum.org/669c9e2e2027310b6b3cdce6e1c52962/Ethereum_Whitepaper_-_Buterin_2014.pdf

Dannen, C. 2017. *Introducing Ethereum and solidity*. Berkeley: Apress, Vol. 1, 159–160

Di Ciccio, C., Meroni, G., Plebani, P. 2022. 'On the adoption of blockchain for business process monitoring'. *Software and Systems Modeling*, 21(3), 915–937.

Gaur, N., O'Dowd, A., Novotny, P., Desrosiers, L., Ramakrishna, V., Baset, S. A. 2020. *Blockchain with hyperledger fabric: Build decentralized applications using hyperledger fabric 2*. Birmingham, UK: Packt Publishing Ltd.

Magazzeni, D., McBurney, P., Nash, W. 2017. 'Validation and verification of smart contracts: A research agenda'. *Computer*, 50(9), 50–57.

Mendling, J., Weber, et al. 2018. 'Blockchains for business process management-challenges and opportunities'. *ACM Transactions on Management Information Systems*, 9(1), 1–16.

Nakamoto, S. 2008. Bitcoin: 'A peer-to-peer electronic cash system'. Decentralized Business Review, 21260.

Voshmgir, S. 2019. *Token economy: How blockchains and smart contracts revolutionize the economy*. Berlin, Germany: Shermin Voshmgir-BlockchainHub.

Wood, G. (2014). 'Ethereum: A secure decentralised generalised transaction ledger'. *Ethereum project yellow paper*, 151(2014), 1–32.

Wüst, K., Gervais, A. 2018. 'Do you need a blockchain?'. *2018 Crypto Valley Conference on Blockchain Technology*, 45–54.

Xu, X., Weber, I., Staples, M. 2019. *Architecture for blockchain applications*. Berlin, Germany: Springer.

https://doi.org/10.1017/9781009411783.003 Published online by Cambridge University Press