Distributed Ledgers

Design and Regulation of Financial Infrastructure and Payment Systems

The title-level DOI for this work is:

doi:10.7551/mitpress/13382.001.0001

5
Encryption

Cryptocurrencies are e-message systems similar in their basics to M-Pesa but differing in the amount of cryptography used, the diverse methods for validation of transactions, and the public nature of the ledgers.

5.1 Featured Historical Episodes

One might have thought this cryptography component is what makes DLT new, but that is not the case. The use of cryptography goes back at least to the Mesopotamians, who used it as a key part of their economic and messaging systems. The internet's TCP/IP communications protocol is reminiscent of that, where pieces are put into "envelopes" and encased by beginning and ending bits and then disseminated.

5.2 Historical Examples of Encryption

From 7500 to 3500 BC, in Mesopotamia, the code for communication consisted of tokens of some six types and distinct shapes representing particular commodities. Then, around 3500–3100 BC, new complex tokens were covered with lines or dots (figure 5.1a) conferring qualitative and quantitative information (Schmandt-Besserat 2014). Eventually, tokens were

put in clay envelopes (figure 5.1b) as a manifest for shipping goods, sealed so that tampering with the manifest would be evident on arrival, as would theft of cargo, as a check of the actual cargo inventory against the manifest would reveal. Writing on the clay manifest envelopes to convey contents of the message (and cargo) is what gave birth to cuneiform writing (Trubek 2015).

If the sender and receiver of the shipment trusted each other, they could be sure there was no tampering by not-trusted parties in between. Better put, any tampering of the invoice or

Figure 5.1a
Mesopotamia tokens.
Source: Wikipedia.

Figure 5.1b

theft of the shipment that took place in between would be self-evident to the receiver. In effect, it was as if the receiver had a code that could unlock the envelope while others in between could not, apart from the sender that created the envelope code in the first place.

Another historical example: Tally sticks as messages and proof of contract emerged as monies in medieval England and were used for centuries, including as a means of payment (Harford 2017). Tallies were a way of recording debts with a system. Willow sticks recorded the original debt transaction and then were split in half (see figure 5.2).

With a distinctive grain, the two halves would match only each other, providing the requisite proof. The lender's half, called the tally stock, was used as a safe and convenient form of payment (hence the word "stock"). When cashed in, the two halves were checked (hence the word "check"). The original borrower paid back the loan to the third-party holder of the asset upon presentation. Here, the original borrower was trusted to repay while the tally stick provided a trustless record of the original promise to lenders and third parties and a record to the borrower that the holder was presenting the

Figure 5.2
Medieval English tally sticks.
Source: Winchester City Council Museums.

Figure 5.3
A decision to burn the obsolete tally sticks in 1834 nearly destroyed the Palace of Westminster.
Source: Wikimedia Commons.

original claim (no double presentation of debt). Tally sticks came to an odd conclusion (see figure 5.3).[1]

5.3 Contemporary Encryption

This historical discussion leads to key issues in computer science and algorithms. Messages can consist of individual transactions, blocks of transactions, datasets, and other documents. The key to encryption is the one-way function. The underlying message is hashed into a 32-byte (256-bit) message, as in Secure Hash Algorithm SHA-256, and any change at all in the underlying message produces an entirely different hash. One can go from the input data to the hash but not from the hash back to the input data. The hashed output is referred to as a fingerprint. The hash does not reveal the underlying input, and any attempt to tamper produces a different hash, which is easily verified. The underlying message is secure.

A central component is public/private key cryptography. Keys come in pairs—public keys that may be disseminated widely and private keys that are known only to the owner. The generation of such keys depends on cryptographic algorithms based on mathematical problems to produce the one-way functions. Effective security only requires keeping the private key private. In such a system, any person can encrypt a message using the public key, but that encrypted message can only be deciphered with the private key. Roughly, it is known who sent the message but not what the message is. Public key algorithms are fundamental security ingredients ensuring the confidentiality, authenticity, and nonreputability of electronic communications and data storage. They underpin various internet standards.

IBM was installing crypto express cards into its mainframes by 2009. A key distinction now is the perfectly opaque computational systems, which do not allow participants to look inside during computation, versus zero-knowledge proof systems, which do allow participants to look in and find proof that each subset of code ran as intended.

5.4 Validation and Distributed Consensus

The roots of distributed ledgers come from distributed consensus, a concept that has been studied for decades in computer science. The traditional application promotes reliability in distributed computing systems. Narayanan et al. (2016) define a distributed consensus protocol as one in which there are n nodes that have an input value. Some of these nodes are faulty or malicious. A distributed consensus protocol has two properties: It must terminate with all honest nodes in agreement on value, and the value must have been generated by honest nodes. This, again, is the basis of secure multiparty computation.

We come to a clear definition of distributed ledgers as multiple, distrusting organizations that run a protocol to create

an append-only log in which all participants can verify the integrity of entries appended to the log.[2] This definition has an advantage because it does not refer to trusted third parties; otherwise it is ambiguous whether there is one common trusted party or layers of trust. It is sufficient that there be some distrust.

To set the stage for this discussion, we present the aptly named Byzantine Generals Problem, which has its roots in distributed computing. A subset of a group of generals consists of potential traitors, bad nodes that are either malicious or sending error-ridden messages (Lamport, Shostak, and Pease 1982; Robinson 2009). The decision the generals must make is whether to attack or withdraw—that is, approve transactions or not—and this requires consensus. The generals exchange messages with each other. If the number of potential traitors (faults) is known, and all other nodes tell the truth, then, intuitively, cross-checking a sufficient number of messages is sufficient. However, the degree of difficulty is a function of primitive assumptions. If a coordinator is assumed to always send honest messages, then things are easier, as one only needs to check a small sample of other nodes—a bit of centralization. If the coordinator could be faulty, more cross-checks are needed. If nodes somehow cannot lie about what they have heard, that is helpful, though this requires more rounds. If nodes start repeating what they have heard from others, the group may have to abandon the primary coordinator node.

Here, as an overview, are some of the key differences among existing consensus protocols.[3]

Unlike a bank's or telcom's centralized ledger of account balances, validators in a network must achieve consensus. Bitcoin and Ethereum use proof of work (PoW), in which nodes in a network compete with computing power to solve cryptographic math puzzles and reach consensus. Anyone can do this mining; membership is open, and miners can join and/or

leave. Byzantine Fault Toleration (BFT) is actually a property of an algorithm, not the algorithm itself. Sometimes the algorithm is referred to as a Byzantine Agreement. BFT normally means the algorithm is guaranteed to converge or is capable of reaching consensus, even if there are adversarial nodes or if nodes drop from the network. In practical BFT (PBFT) algorithms, this requires "3f+1" replicas to be able to tolerate "f" failing nodes. As PBFT chooses a leader in round-robin fashion, nodes need to agree on a "membership list" of nodes to select from, originally picked by the company that designed the protocol (Curran 2018). When such an authority controls the list, the system is referred to by some as "centralized," no matter how many nodes are approved to operate. Essentially, every node is involved with every transaction to reach an agreed-on critical number, a quorum. Typically a BFT algorithm consists of a system with messages sent back and forth in a voting process, and consensus is achieved when over 66% of the nodes agree. Under proof of stake (PoS), the selected validators that are to suggest the next block for approval are chosen at random followed by multiround voting mechanisms typically based on stake in the system, as with the number of coins held. PoS BFT systems are significantly faster than PoW. Ripple pioneered a decentralized alternative algorithm called Federated Byzantine Agreement (FBA), and Stellar refined it to provide the first provably safe FBA protocol. Each entity decides on others it trusts, a so-called quorum slice. When these slices overlap sufficiently, it is a quorum, necessary to approve transactions. Unlike the earlier Byzantine Agreements, Stellar's FBA is free entry or open membership into validation. Ripple is semi-permissioned and stands between Stellar and the entirely permissioned blockchains of R3 Corda, Hyperledger, Ethereum, and Swift's version of distributed ledgers.

Here are the featured properties that distinguish these various systems. Again, fault tolerance means a protocol can

survive failure of a validator at any point. Safety is a guarantee that something bad will never happen (e.g., no forks or partitions as multiple competing versions of truth), but under safety, if no consensus is reached, an auxiliary process is required to reboot. Liveness (as in availability) means the system is always in operation, even if there are faults or forks, making progress toward some eventual conclusion. PoW favors liveness over safety (forks are possible). The FBA system favors safety, as an accidental fork halts operations until fixed. Other distinctions have to do with latency and transaction speed. Bitcoin with PoW is slow and requires approximately six blocks of transactions groups on ledgers to be confirmed, asymptotically, which takes approximately an hour. For BFT and FBA protocols, with their message passing and voting, transactions are approved every three to five seconds. Asymptotic security means no amount of computing power can overcome consensus. Bitcoin does not have this. BFT and FBA are approved with private keys and ledgers-transaction-asymptotic security is achieved. Finally, validation systems may still be subject to collusion from bad actors. In Bitcoin, there is concern now with potential collusion among miners (as discussed below). In BFT protocols, over 66% of validators have to collude. In an FBA protocol, a complicated web of approval is thought to make collusion virtually impossible.

Among blockchain platforms that allow smart contracts, relevant here in the current discussion of alternative validation systems (and discussed again in chapter 6), are Hyperledger Fabric, Ethereum, Quorum, and Corda.

Hyperledger is modified according to the needs of enterprise. There is no one-size-fits-all. Hyperledger allows multiple consensus algorithms. Quorum, based on Ethereum, uses a peer-to-peer encrypted message exchange for transferring private data to network participants and offers consensus mechanisms that are appropriate for semiprivate consortium chains

with controlled user groups. Finally, Corda is a permissioned ledgers system in which contacting parties name one or several nodes that are responsible for consensus (Sharma 2019).

5.5 Featured Examples

5.5.1 Bitcoin

Transactions in Bitcoin are encoded messages. The public and private keys ensure that no one can transact on someone else's ID, impersonating a node. Because the message or transaction can only be created with the key combination, it is known that the spender wishes to unlock and spend the coin. Plus, this brings commitment to the transaction, so it cannot be undone or reneged on later.

Double-spending would be possible if two messages from a given node were able to spend the same coin. In fact, with internet latency, the problem mentioned earlier, it would be hard to know who the victim is—that is, which transaction came first and should be valid in principle, as time stamps are not necessarily chronological. For Bitcoin, blocks of individual transactions are broadcast to the entire network, or at least to those listening among the community of users. Blocks economize on messages and costs of validation. Anonymous nodes verify blocks of new transactions of which they are aware, transactions that have been accumulating as candidates on individual copies of ledgers over the previous 10 minutes or so. The messages broadcast to the community of users consist of these new series of transactions on the block, concatenated with a randomly chosen number and then hashed into a 256-bit encrypted message. Hashing as noted earlier is a one-way function. To dis-encrypt the hash back into the message, one needs to proceed by trial and error, though solutions are evident once found. In this sense it is a random miner of the code who succeeds in inverting. In fact, a group of miners are all

simultaneously running code to decipher and the miner that finds the solution first finds it essentially randomly, as everyone is attempting to find by trial and error. All of the blocks of ledgers in a chain are linked together, given that the top of each ledger contains the hash of the previous ledger. Cryptography with proof of work by these miners, implemented up to six times, ideally sends the probability of malfunction or fraud asymptotically to zero.

One potential problem is that nodes as bad actors could violate the protocol and propose the latest version of the ledger that they would like to become immutable (e.g., knowingly containing the second of the double-spend transactions while the first was already used to acquire something else). To thwart this, under the Bitcoin system, it is again as if one node were selected at random to certify a current new candidate ledger. There are thus two keys to Bitcoin. One is this certification, which requires time and energy. A proof-of-work algorithm requires a selectable amount of work to find the random number that, when added to the set of transactions, creates the hash. Difficulty is controlled. The discovered random number is then added to the bottom of the block as the proof of puzzle solved, a certification of work done that all can confirm easily. The work, which is costly in its use of electricity and equipment, limits entry into validation. The second key to Bitcoin, and a premise of computer science more generally, is that most nodes are honest, so the de facto randomly selected miner is likely to be honest and following the protocol.

Temporary multiplicity or fraud is possible if another branch containing new blocks is created. But the conventional protocol is that the longest interim chain is considered to be the valid one. To reinforce this, and not incidentally, here Satoshi Nakamoto (2008) put some economics into the design of the computer science protocol. Miners have incentives to mine the longest chain, as they are rewarded in Bitcoin only if the block

of transactions they validate becomes, eventually, part of the immutable history. To repeat in crude terms: Validators now have pecuniary interests in the outcomes they are validating. (We shall come back in this idea in chapter 6.)

Some recent economics literature has provided critiques of Bitcoin.[4] Others argue the cryptocurrency protocol is more robust and malleable than it might seem. First, there could be threats of a double-spending run or extortion if a group of miners acquired 51% of computing power. The industry of miners is in fact quite concentrated, so this has been a concern.

Relatedly, Budish (2018) has argued that the expense of acquiring majority hash power is not an effective deterrent against double-spend attacks. To paraphrase Budish, the amount of computational power devoted to mining must satisfy (1) a zero-profit condition among miners, who engage in competition for the prize associated with adding the next block to the chain, and (2) a limited commitment or rationality constraint that the "stock" computational costs of such an attack must exceed the recurring "flow" payments to miners for running the blockchain, so that they stay in. The second constraint is less binding if (i) the mining technology used to run the blockchain is both scarce and non-repurposable and (ii) any majority attack is a "sabotage" in that it causes a collapse in the economic value of the blockchain. The latter is something close to Nakamoto's original argument.

Biais et al. (2018) model the proof-of-work blockchain protocol as a stochastic game and analyze the equilibrium strategies of rational, strategic miners. Mining the longest chain is a Markov perfect equilibrium, without forking, in line with Nakamoto (2008). But the blockchain protocol is a coordination game, with multiple equilibria. There exist equilibria with forks, leading to orphaned blocks and persistent divergence between chains.

Aronoff (2019) argues that a reorganization of the chain creating a fork will take at most a few hours to carry out and

that the rental cost will be modest. Thus, what seems crucial for Budish's argument is that the flow of mining rewards will still lie above the rental value of investment in hash power. A further critique beyond this bound is that Budish's logic assumes there are no other responses.

> In fact the victim has the option to either accept the reversion to the chain or to carry out its own reorganization and negate the double-spend attack. Likewise, the attacker can either accept the failure of its attempted double-spend or respond by implementing a reorganization to re-instate the double-spend, and so on. (Aronoff 2019, 2)

5.5.2 Ripple

Closely related to the Kenyan environment where we featured transfers of value from city to town are cryptographic e-money systems for the purchase and sale of national currencies, including transfers of value across international borders. It is at present relatively costly and slow to do this transaction through commercial banking systems. Ripple is a for-profit entity that has revolutionized this environment by working with mainstream large financial institutions.

Users of Ripple buy the coin "XRP" and can make payments among each other using cryptographically signed transactions denominated in XRP. But transactions in fiat currencies and other objects without XRP are frequent. Fiat tokens represent fiat monies on the ledgers. Ripple is essentially a payments protocol for fiat money transfers through fiat tokens. XRP has value, but the company reasons that XRP is part of a design to prevent hacking. Flooding servers with innumerable transactions causes the XRP cost of transactions to rise exponentially, hence it is not a cost-effective strategy. Put differently, simply being malicious can be exorbitantly costly.

For XRP-denominated transactions, Ripple can make use of its internal ledger, and XRP can be sent to anyone. But a primary function is an interbank transfer system. Typically,

financial institutions are key gateways trusted by users that hold funds and issue balances on behalf of customers. In that sense there has been limited entry into the Ripple system. A bank is able to lock fiat currency and transact with the associated fiat token on the Ripple network. Specifically, an issuance is a method for an individual account holder on the blockchain to "lock" a particular asset on the blockchain ledger. After an issuance is made it can be sent to other accounts, taking advantage of Ripple's low fees (Schuster 2017).

Trust lines are Ripple's way of securing transactions between individual parties after issuance. Users have to specify the other users they trust and to what amount. Furthermore, when a payment is made between two users that trust each other, the balance of the mutual credit line is adjusted, subject to limits set by each user. The user paying out to a customer in the country of destination is effectively lending value to the originator of the transaction, trusting to get this value back. This is similar to the medieval *hawala* system among merchants and correspondent banks, a popular and informal value-transfer system based not on the movement of cash or on telegraph or computer network wire transfers between banks, but instead on the performance and honor of a huge network of money brokers (known as *hawaladars*) (Wikipedia 2018a).

In order to send assets between users that have not directly established a trust relationship, the protocol tries to find a path between the two users such that each link of the path is between two users who do have a trust relationship, again subject to caps.[5] In principle, if there is not a trust path, then one can use XRP to balance the transaction in the other direction. Likewise, credit lines readjust to earlier levels if there are flows among trusted users going the other way. Outstanding IOUs are on a public ledger of accounts.

All transactions on the Ripple network need to be validated in the sense of agreeing to correctness and timing in

order to prevent double-spending. Each Ripple server connects to a network of peers, relays cryptographically signed transactions, and maintains a local copy of the complete shared global ledger. A Ripple server running in validator mode additionally participates in the consensus process and is a part of an interconnected web of validators, each of whom trusts a specific set of validators not to collude.

A FBA algorithm relies on four rounds of sequential voting, starting at a 50% quorum and reaching 80%. This is termed the Ripple Protocol Consensus Algorithm. It seems from available descriptions that server nodes may have a stake in the transactions they are validating (XRP Ledger 2019).

5.5.3 Stellar, Featuring Entry

The Stellar Development Foundation is a not-for-profit organization that provides greater access and inclusion by connecting people to low-cost financial services. Stellar is open source and a public ledger: It sees expansion into underserved populations as its primary mission. Stellar does not rely on mainstream financial institutions. Individual users are not necessarily large financial institutions such as banks—that is, they are allowed to be nonbanks and small, such as money-transfer operators.

Stellar Consensus Protocol (SCP) (Mazieres 2016) uses a Federated Byzantine Agreement that allows more universal access, akin to the internet, as a way to interact among strangers. In the Stellar protocol for validation, each participant names others it considers important and requires that the majority of these others agree to any batch of transactions. Yet those other important participants do not agree until the participants they consider important also agree. Each transaction requires a majority of nodes designated as "important" by both traders (Ray 2018). The system is thus designed to be open to new entrants who name their own trust network. Unlike Bitcoin, Stellar forms a

decentralized consensus among a group of nodes that are transitively connected to each other by trust.

Stellar, like Ripple, can transfer value across virtually any object (e.g., creating fiat money tokens that are then cashed out). Stellar can also transfer the tokens of others who wish to design their own platform with their own coin, linking to Stellar for value transfers.

Stellar uses anchors and market makers to intermediate the exchange of individual parties. An anchor is typically a highly regarded financial institution—namely, a commercial bank. This part is similar to Ripple. To make an international exchange transaction, for example, a customer makes a deposit with the anchor in the fiat currency of the country of origination, for example, and thus issues an IOU, a debt, to the depositor. This is termed a *base account*. The fiat money is then converted 1–1 to a token-equivalent amount, apart from fees, and these tokens are termed *assets* in a base account. The anchor contacts a market maker—a user who, like broker-dealers in other contexts, posts bid-ask spreads and carries some inventory of a variety of assets. If not holding the fiat token of the country of destination, a second broker-dealer acts as a go-between. An algorithm searches for minimal paths. Potentially finding the optimal path is an NP-hard problem, but typically only two steps are used, at most. The user broker-dealer needs to establish a trust line with the anchor to assure itself the deposit with the originating fiat money is there and that the asset is backed in that sense.

5.5.4 Recent Entrants

HotStuff is a practical Byzantine fault-tolerant protocol that replaces a mesh communication network with a star communication network (see figure 5.4). It will be used by Libra. This means each communication will rely on the leader. The node no longer broadcasts the message to other nodes but sends the

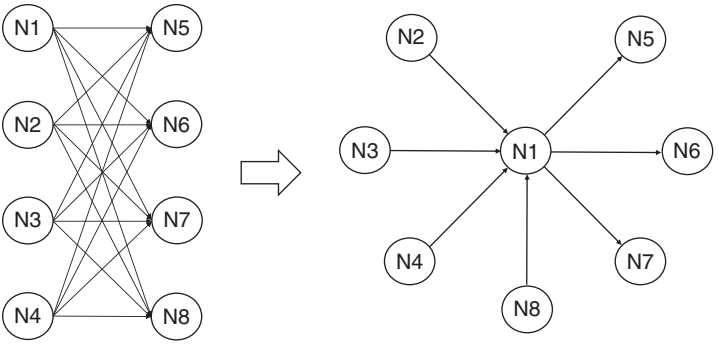

Figure 5.4
Mesh communication networks and star communication networks. The figure on the left shows how nodes *N* are connected to each other. The figure on the right displays a classic star network with a central node.
Source: Author.

message to the leader, which processes it and sends it to other nodes. Thus the communication complexity of the system is greatly reduced. Similar to PBFT, the leader proposes a state transition request and other nodes check its legitimacy after receiving the request.

Algorand uses a simple Byzantine agreement protocol with a leader. It is robust to latency and does not rely on the participants having synchronized clocks. Notably, Algorand takes into account the possibility of malicious leaders:

> When honest messages are delivered within a bounded worst-case delay, agreement is reached in an expected constant number of steps when the elected leader is malicious, and is reached after two steps when the elected leader is honest. (Chen et al. 2018, 1)