Security and reliability of cross-chain exchanges

Léonard Lys



THÈSE DE DOCTORAT

SORBONNE UNIVERSITÉ

Spécialité: Informatique

École doctorale Informatique, Télécommunications et Électronique (Paris)

ED130 — EDITE de Paris

Présentée par

Léonard Lys

Pour l'obtention du grade de docteur

Sujet de la thèse :

Sécurité et fiabilité des échanges inter-blockchain

Security and reliability of cross-chain exchanges

Composition du jury :

Rapporteurs	
Gerard Memmi	Professeur – TeleComParisTech
Stefano Secci	Professeur – CNAM
Examinateurs	
Marc Shapiro	Directeur de recherche émérite – Inria
Hammed Ramdani	Directeur – Palo IT Lyon
Encadrement de la Thèse	
Arthur Micoulet	Encadrant scientifique – Palo IT Paris
Maria Potop-Butucaru	Directrice, Professeure – Sorbonne Université

Thèse soutenue publiquement le *30 juin 2022*

Résumé

Intuitivement, on peut définir une blockchain comme un grand livre de comptes dont chaque titulaire possède une copie intégrale. Un livre de comtpe est un tableau à deux colonnes, qui à un compte A associe un solde X. Effectuer une transaction sur le livre de compte revient à effectuer la demande de transférer $X\$$ du compte A vers un compte B. Dans une blockchain, lorsque le titulaire d'un compte désire réaliser une transaction, au lieu de demander à une banque ou un organisme central d'effectuer le transfert, il demande à l'intégralité des titulaires de "voter" pour déterminer si cette transaction est valide ou non. Si la transaction est considérée valide par une majorité des titulaires, alors ils rajoutent la dite transaction à leur propre copie du livre de compte. De manière plus formelle, une blockchain est un registre distribué, dupliqué sur les noeuds d'un réseau, dont les participants obéissent à un protocole, qui définit les règles de modification du registre.

De part leur caractère distribué, les blockchains tendent à satisfaire certaines propriétés. D'abord l'intégrité, puisque la donnée est répliquée sur plusieurs ordinateurs, il n'y a pas de point de panne unique. Si une des répliques est attaquée ou défaillante, les autres garantissent la sauvegarde du registre. En découle une autre propriété, l'immutabilité. Une fois qu'une transaction a été ajoutée au registre, elle ne peut plus être modifiée ou effacée. Ces propriétés font de la technologie blockchain un candidat idéal pour l'implémentation d'une monnaie électronique pair-à-pair. C'est par ces mots qu'a été introduite en 2008 dans [67] celle qui est considérée comme la première blockchain, Bitcoin.

Depuis, une multitude de blockchains leurs crypto-actifs associés ont émergés, chacun présentant leurs caractéristiques techniques, leurs avantages et inconvénients, sans qu'un standard s'impose dans l'industrie. Sur la même période, la capitalisation de ces crypto-actifs et les volumes d'échange ont connu une augmentation constante malgré quelques mouvements de recul ponctuels, propre à n'importe quel marché.

Le cas d'usage qui génère aujourd'hui le plus de volume, avec environ deux billions de dollars quotidiens en 2021, est l'échange de crypto-actifs. Les utilisateurs se rendent sur des plateformes d'échange ou ils déposent leurs actifs et les échangent contre les actifs des autres utilisateurs. La plateforme d'échange jouant le roule d'intermédiaire de confiance. Bien qu'il propose des avantages, ce système n'est pas satisfaisant en tout point. Premièrement, les plateformes fonctionnent via les commissions qui sont payées par les utilisateurs. Ensuite,

puisqu'elles concentrent dans leurs infrastructures une grande quantité de fonds, elles sont une cible privilégiée pour des personnes malveillantes. Enfin, pour la même raison et puisqu'elles contrôlent les échanges et les prix affichés, elles peuvent artificiellement manipuler le marché.

L'incompatibilité d'une institution centralisée ou d'un tiers de confiance avec la blockchain avait déjà été anticipé au cours de la genèse de Bitcoin. Il apparait donc nécessaire de proposer des protocoles pair-à-pair qui permettent aux utilisateurs d'échanger des crypto-actifs entre plusieurs blockchains, sans intermédiaire de confiance.

Le verrou technologique auquel cette thèse s'adresse est donc l'interopérabilité des blockchains. Chaque blockchain est un environnement indépendant avec son propre réseau, son protocole et ses règles. Elles n'ont pas nécessairement été conçues dans l'optique de s'interopérer. En vérifiant l'historique des transactions et en identifiant l'auteur d'une transaction grâce aux signatures numériques, il est possible de vérifier si une transaction peut ou non être ajoutée à la chaîne. Mais à ce jour, aucun mécanisme n'est prévu pour coordonner des transactions entre plusieurs chaînes afin de procéder à un échange. Un système d'échange de crypto-actifs entre deux chaînes cherche à satisfaire les propriétés suivantes; atomicité, l'échange à lieu intégralement ou pas du tout, sécurité, les participants ne risquent pas de perdre leur crypto-actifs et enfin vivacité, la durée de l'échange doit être limitée dans le temps.

Dans un premier chapitre, nous allons informellement introduire les différent élements techniques qui composent un système blockchain. Nous allons aborder des sujets tel que les structures de donnée, les réseaux distribués et la cryptographie. Ensuite, nous étudierons les protocoles blockchains et donner un cadre d'évaluation des différentes blockchains. Nous présenterons les applications qu'offfent cette technologie et nous finirons ce chapitre par une introduction à l'interopérabilité des blockchains.

Dans un deuxième chapitre nous nous intéresserons au problème des oracles de prix. Ces systèmes permettent d'introduire de la donnée provenant de l'exterieur vers la blockchain. Ce problème est au centre des problématiques d'interopértabilité car ils permettent aux blockchains d'accéder à de la donnée extérieure et donc potentiellement de synchroniser des processus entre plusieurs chaînes. Dans ce document, nous formaliserons le problème d'oracle blockchain et proposerons un protocole qui offre de meilleures garanties que les protocoles existants. Nous prouverons formellement la correction du protocole.

Dans un troisième chapitre nous nous intéresserons au protocole d'atomic cross-chain swap formalisé par Maurice Herlihy. Nous allons analyser les propriétés satisfaites par ce

protocole, puis par une implémentation, nous allons évaluer les performances et les coûts de ce protocole. Nous proposerons ensuite un nouveau protocole permettant de résoudre les limitations du protocole de Herlihy, en faisant usage de processus de vérification inter-chaîne : les relais. Nous proposerons une preuve de correction du protocole ainsi qu'une évaluation théorique de performance.

Abstract

Intuitively, we can define a blockchain as a ledger of accounts of which each holder has a complete copy. A ledger is a two-column table, which associates an account A with a balance X. Performing a transaction on the ledger is equivalent to requesting to transfer $X\$$ from account A to account B. In a blockchain, when an account holder wants to make a transaction, instead of asking a bank or central agency to make the transfer, it asks the entire group of account holders to "vote" on whether the transaction is valid or not. If the transaction is considered valid by a majority of the holders, then they add the said transaction to their copy of the account book. More formally, a blockchain is a distributed ledger, duplicated on the nodes of a network, whose participants obey a protocol, which defines the rules for modifying the ledger.

Because of their distributed nature, blockchains tend to satisfy certain properties. First is integrity, since the data is replicated on several computers, there is no single point of failure. If one of the replicas is attacked or fails, the others guarantee the backup of the ledger. Another property is immutability. Once a transaction has been added to the ledger, it cannot be modified or deleted. These properties make blockchain technologies an ideal candidate for the implementation of a peer-to-peer electronic version of cash. It is in these words that were introduced in 2008 in [67], what is considered the first blockchain, Bitcoin.

Since then, a variety of blockchains and their associated crypto-assets have emerged, each with its technical characteristics, advantages, and disadvantages, without any industry-standard being established. Over the same period, the capitalization of these crypto-assets and the volumes of exchange have increased steadily despite some occasional downturns, typical of any market.

The use case that generates the most volume today, with about two trillion U.S. dollars daily in 2021, is the exchange of crypto assets. Users go to exchange platforms where they deposit their assets and exchange them for the assets of other users. The exchange platform

acts as a trusted intermediary. Although it has its advantages, this system is not satisfactory in every respect. Firstly, the platforms work via commissions that are paid by the users. Secondly, since they concentrate a large number of funds on their infrastructure, they are a prime target for malicious attackers. Finally, for the same reason and since they control the exchange rates and the posted prices, they can artificially manipulate the market.

The incompatibility of a centralized institution or a trusted third party with the blockchain had already been anticipated during the genesis of Bitcoin. It, therefore, appears necessary to propose peer-to-peer protocols that allow users to exchange crypto-assets across several blockchains, without a trusted intermediary.

The technological challenge that this thesis addresses is therefore the interoperability of blockchains. Each blockchain is a self-contained environment with its own network, protocol, and rules. They were not necessarily designed with interoperability in mind. By checking the history of transactions and identifying the author of a transaction thanks to digital signatures, it is possible to verify whether or not a transaction can be added to the chain. But to date, there is no mechanism for coordinating transactions between multiple chains to make a cross-chain exchange. A crypto-asset exchange system between two chains seeks to satisfy the following properties; atomicity, the exchange takes place entirely or not at all, security, the participants do not risk losing their crypto-assets and finally, liveness, the duration of the exchange must be limited in time.

In a first chapter, we will informally introduce the different technical elements that make up a blockchain system. We will cover topics such as data structures, distributed systems and cryptography. Then, we will study blockchain protocols and give an evaluation framework for the different blockchains. We will present the applications that this technology offers and we will end this chapter with an introduction to the interoperability of blockchains.

In a second chapter we will focus on the problem of price oracles. These systems allow the introduction of data from the outside-world into the blockchain. This problem is at the heart of interoperability issues because they allow blockchains to access external data and thus potentially synchronize processes between several chains. In this paper, we will formalize the blockchain oracle problem and propose a protocol that offers better guarantees than existing protocols. We will formally prove the correctness of the protocol.

In a third chapter we will focus on the atomic cross-chain swap protocol formalized by Maurice Herlihy. We will analyze the properties satisfied by this protocol, and then, by an

implementation, we will evaluate the performances and the costs of this protocol. We will then propose a new protocol to solve the limitations of Herlihy's atomic cross-chain swap by using inter-chain verification processes: relays. We will propose a proof of correctness of the protocol as well as a theoretical performance evaluation.

Contents

List of Figures

List of Tables

Chapter 1 Introduction

In this chapter, we are going to introduce some preliminaries on blockchain technologies. In the first section, we will start by independently reviewing the technological bricks that combined makes up a blockchain system. In a following section we are going to explore blockchain protocols in the form of three case studies. Then we are going to give some key elements to evaluate blockchain systems. We will discuss also discuss the applications of blockchain technologies. Finally, we will end up this first chapter by introducing blockchain interoperability, which is the main topic of our contributions.

1.1 Blockchain fundamentals

1.1.1 The digital ledger

At the core of any blockchain system stands the actual ledger of transactions. This is the account book that every node stores a copy of. The ledger is a large two-column table that associates an account A with a balance X. As its name suggests, the ledger is made of blocks that are linked together. Essentially, a block is divided into two sections. The first section is the block header. It contains metadata that gives structure to the chain. It may contain a timestamp a nonce and many different data depending on the blockchain. But most importantly, and this characteristic is common to every blockchain, the block header contains a unique id pointing to the previous block, thus forming the chain. The second section of the block is the block's body. It contains a list of transactions formatted in some way, as well as other information relative to the transactions of the said block. A simplified representation of the ledger is given in Figure 1.1.

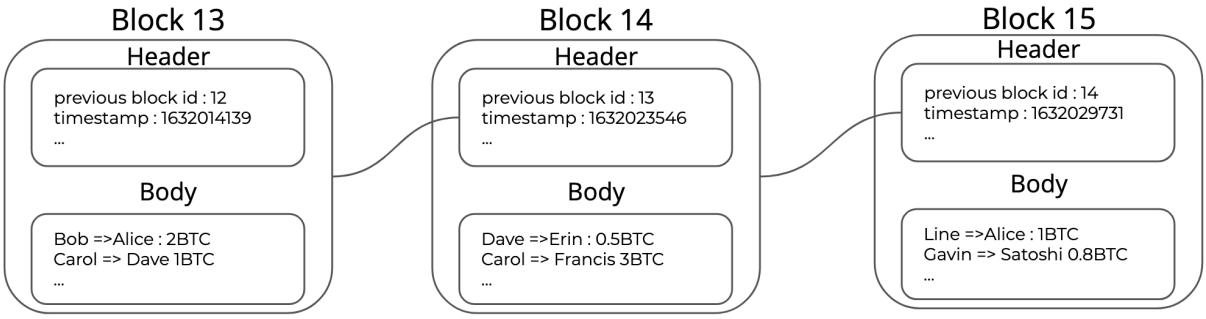


Figure 1.1: Simplified blockchain ledger structure

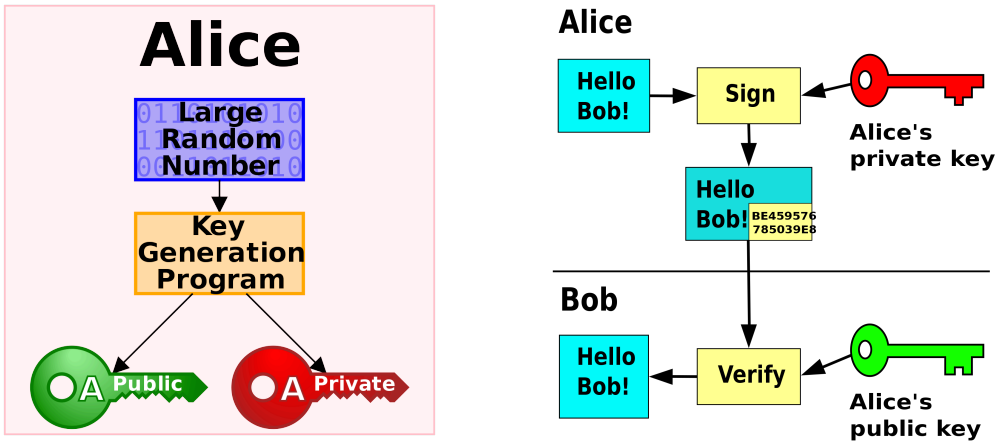
The number of transactions that each block contains, the way those transactions are stored, and the way new blocks are added to the chain varies depending on the type of blockchain. But the common characteristic that makes a system a blockchain system is the fact that each new block contains a reference to the previous one. This characteristic is very powerful for ensuring data integrity. Indeed, the fact that each block contains a reference to the previous one makes it harder to corrupt, because modifying a past block would imply modifying all successive blocks.

1.1.2 Cryptography

Public key cryptography In a blockchain system, to guarantee security, it is necessary that participants can authenticate each other. Indeed, when someone wishes to execute a transaction, one must be sure that the person sending the transaction message is the owner of the assets. To do so, blockchain systems make use of public-key cryptography. Each participant possesses a pair of keys, a public one and a private one. Both are derived from a big random number. It is easy to calculate the public key from the private one, but it is impossible in the other way. A core feature of public-key cryptography is digital signatures: when a participant wishes to send a transaction message and to be identified by the other participants, he uses his private key to sign the message. Then, with the public key of the participant, one can verify that the signature has been created by the private key and thus, that the message is authentic. The safety of this system is guaranteed by the fact that it is almost impossible to forge a fake digital signature without having the corresponding private key.

Hash functions Another cryptographic function that is widely used in blockchain systems is hash algorithms. A hash algorithm is a mathematical function that transforms any given input into a fixed size output. Those functions should satisfy the following properties:

- Deterministic: A given input will always produce the same output.
- One-way: It should be infeasible to calculate the input given the output. The only way to find two inputs that produce the same output should be by brute-force search (i.e. by trying every possible input until finding a collision).
- Large output space: The number of inputs to check during a brute-force search before finding a collision must be high enough for the attack to be infeasible.
- Non-locality: A slightly different input should produce a radically different output.
- Uniformity: The output of a hash function should be uniformly distributed over the set of



(a) The public and private keys are derived from a large random number. The public key can be calculated from the private key but not the other way around

(b) Alice signs a message with her private key and send it to Bob. With Alice's public key, Bob can verify the authenticity of the message

Figure 1.2: Public key cryptography

possible output values

In blockchain systems, hash functions are used to protect data integrity: the intuition is that a hash function allows producing a unique digital fingerprint of any chunk of data. As explained earlier, each block contains in its header the unique id of the previous block. This unique identifier is the hash of the previous block. So if an attacker tries to modify an old block, its hash will be radically different. Thus any modification of the chain is easily detectable. The same principle is also applied to transaction storage inside the block body, through the use of Merkle trees. This will be further discussed in the following section. Finally, hash functions are used in some blockchains as Proof-of-Work. This will be further discussed later.

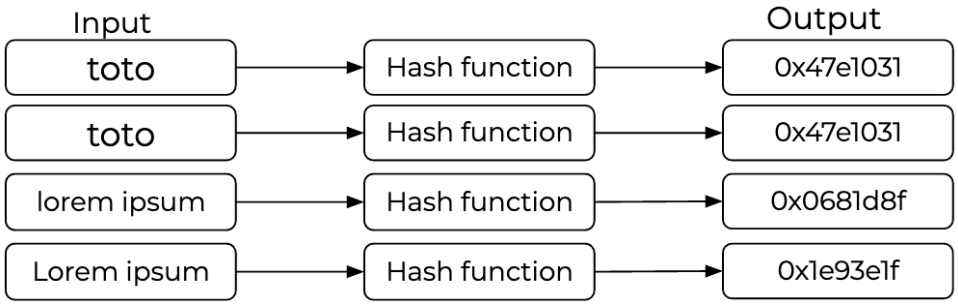


Figure 1.3: Illustration of an hash algorithm properties

1.1.3 Brief introduction to Bitcoin cryptography

To put in perspective how cryptographic functions are used in blockchain systems, let us have a brief introduction to Bitcoin's cryptography. In Bitcoin the ownership of a bitcoin [1] is established through digital keys, addresses, and digital signatures [6]. Participants in the system are identified by what is called their Bitcoin invoice address or simply address. It is an identifier of 26 to 35 alphanumeric characters that is derived from the user's public key. More precisely it is the result of the public key being hashed by two different hash functions. This is presented in Figure 1.4.

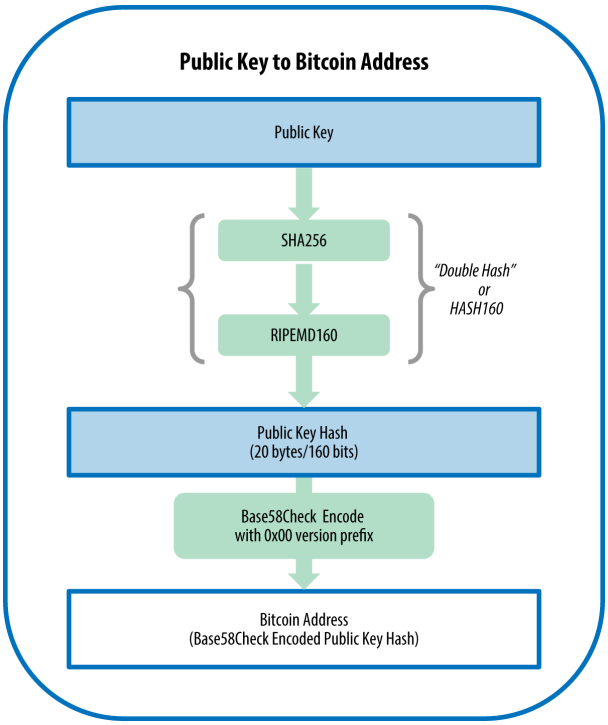


Figure 1.4: How Bitcoin addresses are derived from public keys

In order to receive a Bitcoin payment, one can simply specify the address to the payer. He can then send a transaction to this specific address. To spend the funds contained in this address, one must provide a digital signature made with the private key from which the address is derived from. Thus in Bitcoin, the ownership of a coin narrows down to the ability to produce a digital signature corresponding to the receiving address.

The digital keys are not meant to be stored on the network, but instead, they are generated offline by the user and stored on a file, database or even piece of paper referred to as a wallet. The wallet software manages the keys and is responsible for generating digital keys, calculating

[1] Bitcoin with a capital B refers to the whole protocol and networks' name whereas bitcoin with a lowercase b refers to a single unit of asset of the Bitcoin blockchain

new addresses, signing messages, etc. To sum up, the private key is a pseudo-random number generated by the wallet software. The public key is derived from the private key but the public key cannot be calculated from the public key. The public key can be taught of as an account number while the private key would be the PIN. Invoice addresses are derived from the public key, but in the same manner, public key cannot be calculated from addresses. Figure 1.5 presents a simplified algorithm of Bitcoin addresses generation.

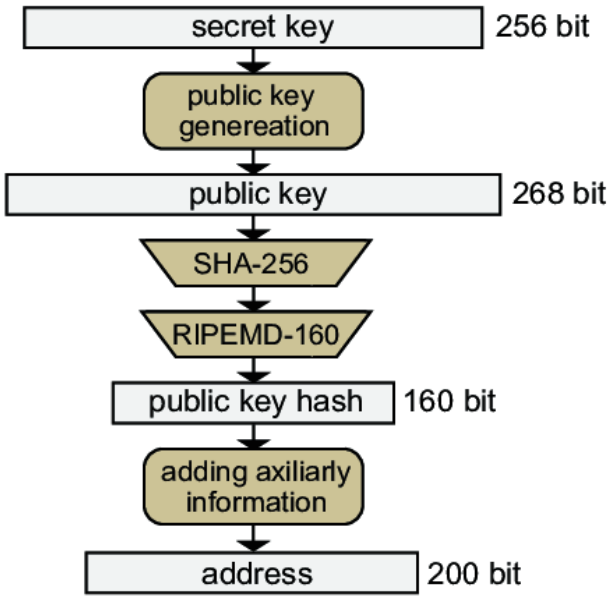


Figure 1.5: Basic algorithm of the Bitcoin address generation

1.1.4 Merkle trees

A Merkle tree is a binary tree where each leaf is the hash of a data block and each node is the hash of its concatenated children. It was first proposed in 1979 by Ralph Merkle [65]. The main application of Merkle trees in blockchain systems is for transaction storage inside a block. To build a Merkle tree out of a list of transactions, we first hash each transaction. Then transaction's hashes are paired up, concatenated, and hashed again. The process is then repeated over the obtained hashes until we reach the root of the tree. This root called Merkle root or hash root is then stored in the header of each block as presented in Figure 1.7. As block hashes allow for integrity checking of the chain, Merkle trees do the same thing at the block level. They allow efficient and secure storage of transaction data as well as efficient and simple integrity checks. Let's assume you want to check if a particular transaction tx_3 has been altered or modified. Let's assume there are n transactions in the tree. We first check whether the hash

of $tx3$ and $tx4$ gives us the correct hash $h34$. Then, cascading upwards, we keep checking the corresponding hash values all the way up to the root. If the original information is indeed valid and has not been changed, we should end up with exactly the same Merkle root. Instead of verifying each individual transaction, which is a complexity of $\mathcal{O}(n)$, you only need to check for the transaction's branch, which results in a complexity of $\mathcal{O}(\log_2 n)$.

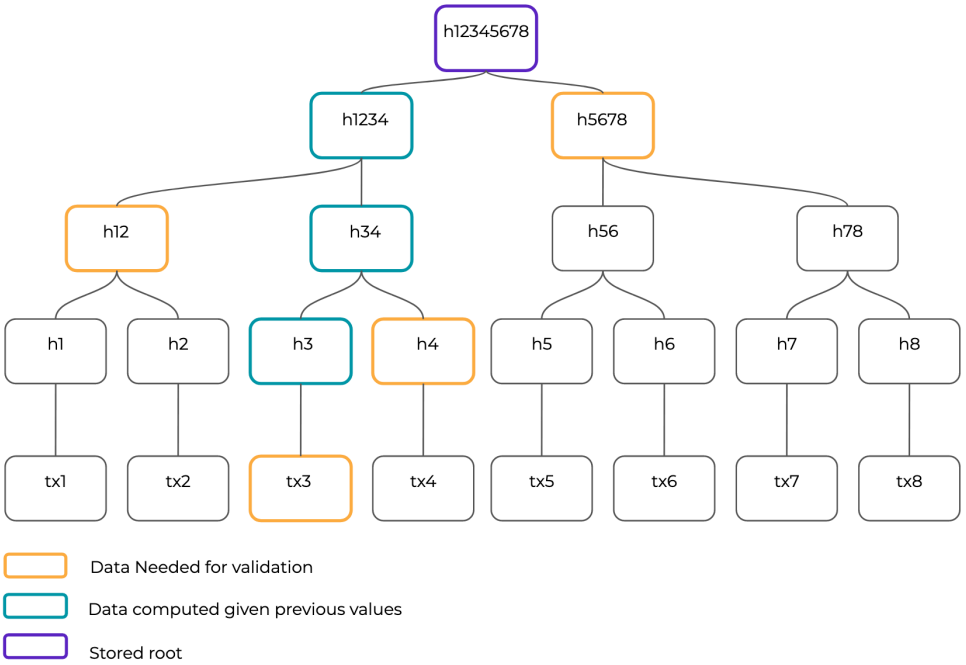


Figure 1.6: An example of which data is needed and computed to check the integrity of a transaction in a merkle tree

Another feature provided by Merkle trees is light clients. The size of a blockchain and thus its storage requirement is ever-growing. In the meantime, the cost of storing data is not necessarily decreasing as fast as the blockchain is growing. At the time of writing, the total weight of the Bitcoin blockchain is more than 350 gigabytes, which is more than many personal computers can store. Light nodes provide part of a solution to this problem, and they mainly rely on Merkle trees. Light nodes are blockchain nodes that don't possess a full history of the blockchain. Instead, they only store the block headers of each block. Since they have the Merkle root of each block, they can verify if a transaction has been included or not. They still rely on full nodes to transmit transactions to the network but they can check if a transaction has been included in a block. In Bitcoin, this is called Simplified payment verification [84]. Ethereum [93] implements an enhanced version of Merkle trees called Merkle Patricia tries [59], which provides the same functionalities over a key-value store. This feature decrements the storage needed for running a blockchain node by 99.9%, which allows for low-resources devices such

as IoT devices to run a node.

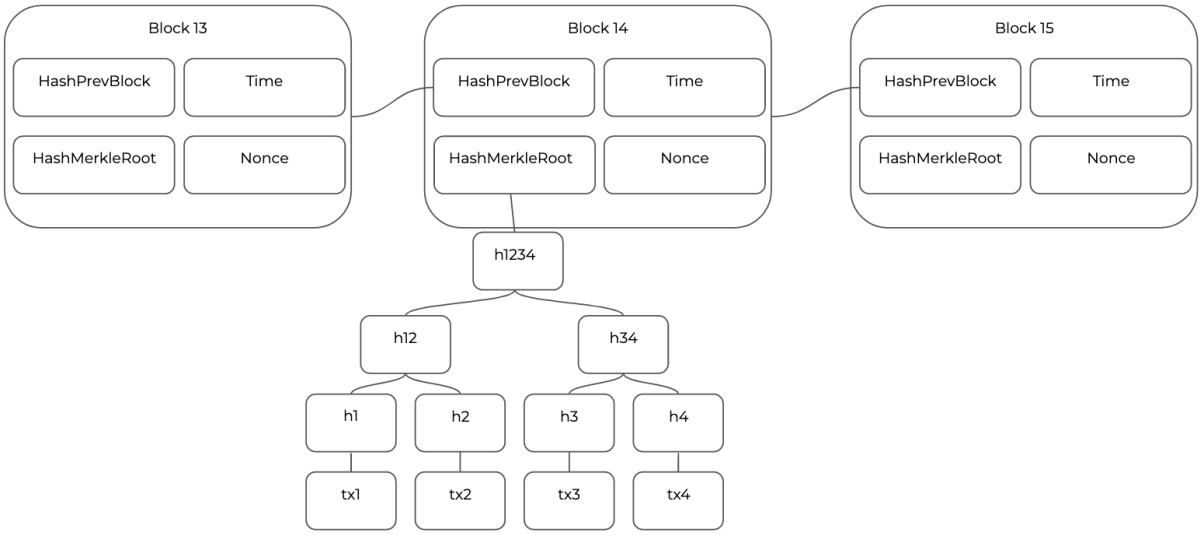


Figure 1.7: A slightly more accurate representation of a blockchain ledger structure

1.1.5 Distributed system

As introduced above, a blockchain is a ledger that is replicated and distributed across a large number of computers. These computers communicate with each other via a peer-to-peer network. The computers in the network work together for a common goal; to maintain the blockchain. This is what we call a distributed system. More formally, a distributed system is a system where several computational units (or nodes), with their own local memory, communicate with each other to achieve some task [62]. A desired property of a distributed system is that it should appear to the user as a homogeneous entity (as if it was a single computer). There are many examples of distributed systems we can cite; the internet, an email service, or the telephone network, but also the control system of an airplane or a ticket reservation system, etc. In the case of blockchains, being a distributed system brings several advantages. First, there is less chance that the system will suffer a single point of failure. Even if a computer of the system is faulty or crashes, the information is not lost because it is replicated on other computers, and the system can continue operating. Another advantage is that the users of the system do not have to trust a central institution. This reduces the power that such institutions may have and thus prevents potential abuse of power or attempts at censorship. To sum up, distributed systems tend to be more democratic, reliable, and transparent than centralized systems.

Unfortunately, although they offer advantages, distributed systems are inherently much more complex to design, implement and operate. Indeed, when developing a distributed

system, one must consider several characteristics and issues that are specific to them.

In this section, we will review the main characteristics and concepts associated with distributed systems. Then in the following section, we will define the consensus problem and see how different blockchain systems try to resolve it.

Concurrency A characteristic that is to be considered when designing distributed systems is concurrency. Each computer (or process) in the network is executing operations at the same time and independently from the others. In addition, those operations may have some influence over other processes' operations. For example, in the case of a blockchain, two different processes may be trying to publish a new block at the same time. For the system to keep working besides concurrent tasks being executed, it is necessary to have some coordination between processes.

Lack global clock Another characteristic of distributed systems is that it is very difficult to implement a global clock to synchronize events. Inside a single computer, the processor's operations are paced by a clock signal. The clock beats at a certain frequency which determines the rate at which the processor executes instructions. A single processing unit is said to be a synchronous system. On the other hand, in a distributed system, where computers are spatially separated, it is very difficult to synchronize all the clocks. Even if one achieves to synchronize all clocks in a system, after some time, the clocks will begin to differ due to a phenomenon known as clock drift. Thus, in most real-world distributed systems, we cannot rely on the clock to determine the sequence in which events happened, they are said to be asynchronous. In a 1978 paper [55], Leslie Lamport shows that by remembering that (1) messages are sent before they are received and that (2) each computer has a sequence of events, we can deduce whether one event happens before another. However, the resulting order is only a partial ordering of the events. Partial ordering only defines the order between certain key events that depend on each other. For example in a blockchain system, partial ordering would guarantee the order between two transactions concerning the same account. But it is not necessary to know the orders of transactions from two distinct and unrelated accounts. Whereas total ordering would guarantee the exact order between all the transactions.

Asynchronous communication Another aspect that often forces us to consider distributed systems as asynchronous is the communication model. Computers in distributed systems communicate through communication channels. Different protocols can be used for message

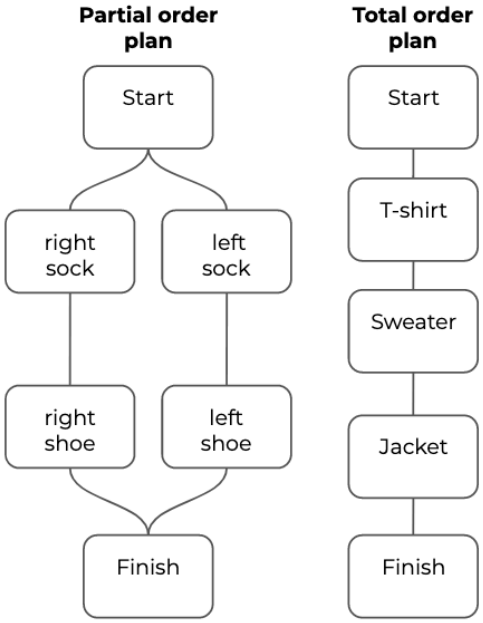


Figure 1.8: Illustration of the concept of ordering and concurrency. In the partial-order plan, the only condition is for the sock to be put on before the shoe. But we do not need to know the exact order between the left and right foot events to achieve the goal of putting shoes on. On the other hand, in the totally ordered plan, we need a totally ordered set of events to achieve the goal of getting dressed.

passing. For example, most blockchains systems communicate through TCP, but other systems might use HTTP, FTP, POP, etc. A common problem with real-world communication protocols is that they might fail. In most real-world applications, the network might lose, delay, modify or duplicate messages. In the asynchronous communication model, it is assumed that there is no upper bound on how long a message will take to be received. On the other hand, the synchronous communication model assumes that there is a known upper bound on message transmission time and that all clocks in the system are perfectly synchronized. Because they lack a global clock and rely on unreliable communication channels, blockchain systems often assume an asynchronous communication model.

Besides lost messages, asynchronous communication also implies that a node can not make the difference between a delayed message and a crashed node.

Failures of components In real-world applications, computers can crash, information can be lost or modified and in the case of an open network (anyone can participate) like most blockchains, we can have malicious attackers trying to corrupt the system. When designing a distributed system those facts have to be taken into consideration. We talk about "fault-tolerant

distributed systems". Whether it be a computer crash, a lost message, or a design problem (a bug), those events are called faults. A fault is anything that deviates from what is expected. A fault will produce unexpected results which might lead to an error. If the error is not taken care of and propagates inside the system this results in a failure. A system is said to be fault-tolerant when a fault does not lead to the failure of the system because the system can recover from those faults. Faults can be roughly divided into two categories:

- Fail-silent/Fail-stop: a component fails and does not send messages or it sends messages that allow the other components to detect that it is faulty. It will either produce no output or some sort of output that indicates that the component has failed.
- Byzantine: the worst faults, a fault presenting different symptoms to different observers [29]. The faulty component can inconsistently appear both failed and functioning to the other component of the system. While it is faulty, the component continues to run and behaves inconsistently, producing different outputs to different components.

A system that is tolerant to byzantine faults is said to be byzantine fault-tolerant (BFT). A Byzantine fault-tolerant system should make a cohesive, uniform decision, even if there are some corrupt elements present in the network.

1.2 Blockchain protocols

In previous section we have seen how the blockchain as a data store structure were designed. We have seen that nodes communicate in a network and thus form a distributed systems. The network might be unreliable but they can authenticate each other thanks to digital signatures. In this section, we are going to review how blockchain systems actually add blocks to the chain; those are the blockchain protocols.

A blockchain system can be seen as a replicated state machine where the ledger of transactions represents the state, and the transactions represent state transitions, as presented in Figure 1.9. It is a replicated state machine because each node of the blockchain system possesses a copy of the ledger (the state), and each node somehow intends to apply the correct state transitions to its own copy of the ledger. The design goal of a blockchain is for all copies of the ledger to be identical.

In a blockchain system transactions are not added one by one, but instead, they are added by blocks of transactions. Thus the goal here is for the nodes in the blockchain system to decide

Figure 1.9: Representation of a blockchain as a replicated state transition system

and agree on the value of the next block. Broadly speaking, during one protocol execution (deciding on one block) we assume three types of processes:

- Leaders or proposers are the processes that will propose the new block to be decided upon and eventually added to the chain.
- Voters are the processes that listen to the proposal of the leader and vote to accept or reject the proposed block.
- Learners or clients are processes that don't participate in the voting process itself but learn the final values that were decided. They are the ones sending the transactions, they are the "average user".

In following executions, processes may switch roles, for example, a process that was a voter at block index n can become a leader at block index $n + 1$.

Learners or clients are constantly sending transactions messages over the network. Those transactions will only be valid once they are added to a block. Thus leaders and voters store the pending transactions in their memory until they can insert them inside a block. Then the process generally consists of four steps:

- Elect: Processes elect a single process that will be the leader for the current execution.
- Propose: The leader builds a block out of the pending transactions and broadcasts it to the voters
- Vote: The voters vote on the block proposed by the leader and send their vote to the other voters
- Decide: Once they received enough votes from other voters, they decide whether or not the block is valid and should be added to the chain. If the number of votes "for" is above some threshold, they decide on that block, add it to their copy of the chain and move on to the next block. If they did not receive enough "for" votes, the block is rejected and the process starts over.

We are going to review proof-of-work (PoW) and proof-of-stake (PoS), which are the two

most common protocols used in blockchain systems today.

1.2.1 Proof-of-Work (PoW)

Overview Proof-of-Work, which stands at the core of Bitcoin and Ethereum, was first introduced in 1993 by Dwork et al. in [31] as a way to deter DDOS attacks.

As explained in the previous section, they are essentially three roles in a blockchain protocol, leader, voter, and learner. With proof-of-work, leaders and voters are both called "miners". The miners are the ones responsible for creating the new blocks, while learners send transactions and keep their copy of the chain up-to-date. All the participants in the network communicate through peer-to-peer communication channels and are identified by their public keys. The miners maintain a full copy of the blockchain.

The miners are constantly listening to the network. They listen to transaction messages sent by the learners and store them inside their memory. This forms a pool of unconfirmed transactions (transactions that are not in a block yet) called a mempool. It is to be noted that each miner has its own mempool, and the list of pending transactions in the mempool can be different from one miner to the other.

In a nutshell, there are essentially four steps to a blockchain protocol; elect, propose, vote, and decide. With proof-of-work, the elect and propose steps are combined together in a kind of race. Instead of electing a leader and then letting the leader propose a block, all the miners are concurrently trying to find a block that meets some criteria. The first one to resolve this cryptographic challenge will be considered the leader, and the block he found will be the proposed block.

The cryptographic challenge for the miners is to build a block whose hash value is inferior to some target number. As introduced in Section 1.1.2, a hash function can take anything as an input and will produce a fixed size output. As the output is a number, it can be compared to another number: the target number. A block is made out of a list of transactions, a timestamp, the previous block's hash value, and a nonce. Recall that a property of hash functions is that a small change in the input will produce a radically different output. Thus by changing anything in what the block contains will radically change its hash value. Miners vary the parameters until they find a block whose hash value is inferior to the target number. They make variations on the list of transactions, on the nonce, and on the timestamp until they find a valid block. The first miner to find a valid block (a block whose hash value is less than the target) wins the challenge

and gets elected as the leader. He then broadcast the block to the network.

This target number is derived from what is called the difficulty. The difficulty is an expression of how hard it is to find a valid block. The target number t corresponding to some difficulty d is generally obtained with $t = c/d$ with c being a very large constant number. The higher the difficulty is, the lower is the target number and thus, the more trials it requires before finding a block whose hash is inferior to the target number.

At the time of writing, the Bitcoin blockchain difficulty was so that the probability of finding a valid block over a single trial is $1.7e - 23$.

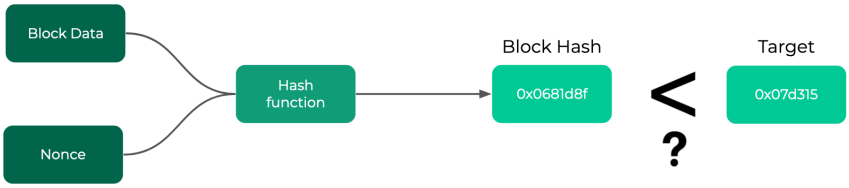


Figure 1.10: How block hashes are built and compared to the target number in order to satisfy the proof-of-work

As we can see, the elect and propose steps are combined, as the first to propose is elected as the leader. Then, on receiving the proposed block, the voters verify the proof-of-work, i.e. that the hash of the block is indeed inferior to the target number. They also check the format of the transactions, that the transactions are legal (sender has enough funds) and that the transaction's signatures are valid. If the block passes all those checks, then voters will add the block to their own copy of the ledger and continue by trying to build a block on top of this one. If on the contrary, the proposed block doesn't pass the checks, then they keep searching for a valid block until they find one.

Obviously, the more computing power a miner has, the more trials he can make over a given period of time, and the more chances he has to be the first one to find a valid block. Thus, in PoW blockchains, the probability of being elected as the leader is proportional to the computing power a miner has.

Adjusting the difficulty In real-world applications of blockchain systems, we wish to guarantee that users can send transactions anytime. This is referred to as the liveness of the system. In order to guarantee some kind of liveness, it is important to ensure that new blocks will be added to the chain and that those blocks will arrive at some expected frequency. The frequency at which new blocks are created in blockchain systems is known as block time. In proof-of-work in particular, the difficulty target number has a great influence on block time.

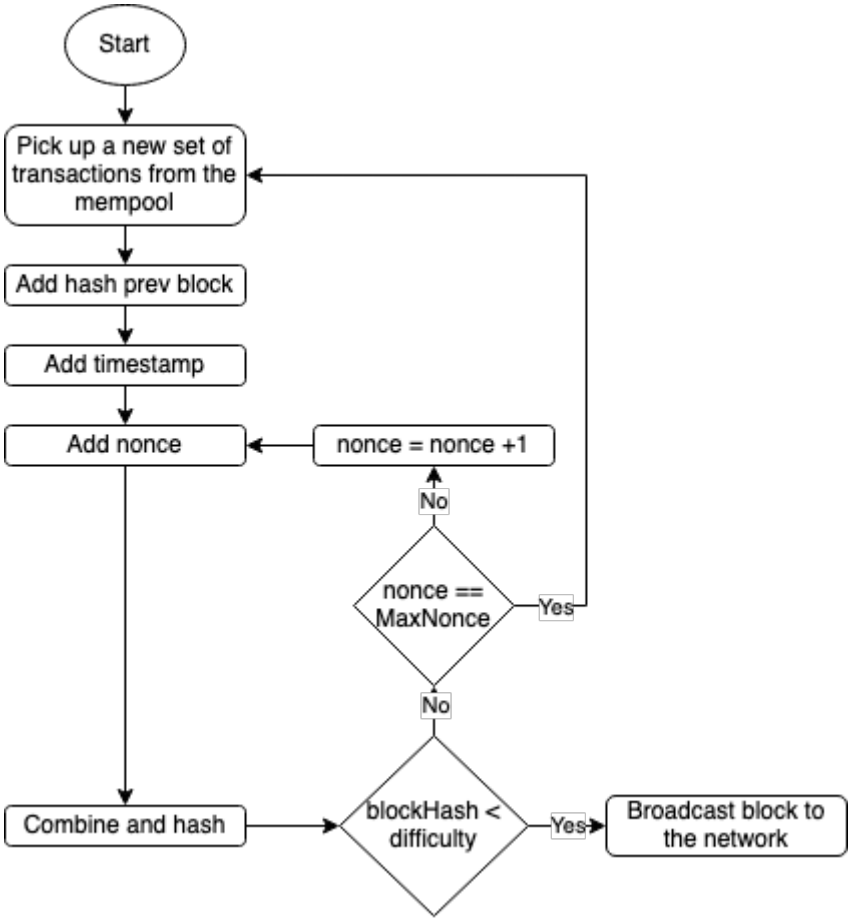


Figure 1.11: Flowchart of how miners try to find a valid block in proof-of-work

As presented in Section 1.1.2, a hash function should be uniform, i.e. the output of a hash function should be uniformly distributed over the set of possible outputs. Thus for a set of possible outputs $A = \{x \in \mathbb{N} | x \leq n\}$ and a target difficulty number δ, the probability of finding a block whose hash value h is less than the difficulty target should be $P(h \leq \delta) = \delta/n$. Therefore, by adjusting the target difficulty number, we modify the probability of finding a valid block. The lower the difficulty target number is, the higher the mean number of trials before finding a valid block will be. In proof-of-work-based blockchain systems, the total combined computing power of all the miners is called the hash rate and is expressed in hashes per second. Given a constant hash rate, adjusting the difficulty is equivalent to adjusting the mean frequency at which miners will find new valid blocks.

The creator of the Bitcoin blockchain wished that the block-time, e.i., the frequency at which new blocks was around ten minutes. This number was chosen arbitrarily as a trade-off between the first confirmation time and the amount of work wasted due to forks. To keep that number relatively stable, a mechanism was implemented. The mechanism had to adjust the difficulty relative to the fact that the total computing power of the network (the hash rate) was

not constant. Indeed, miners can at any moment, join or leave the system. Thus it was decided that the difficulty, or in its developed form, the target number, will be adjusted every 2016 block, considering how fast the last 2016 blocks were mined. We can see in Figure 1.12 how the difficulty follows the variations of the hash rate.

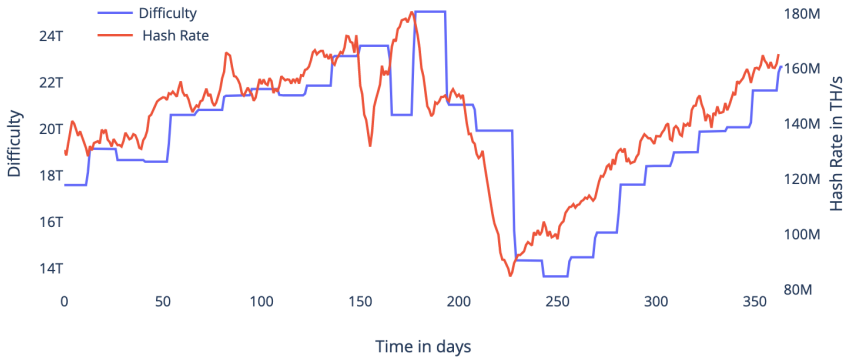


Figure 1.12: Chart of Bitcoin difficulty versus hash rate over a one year period. https://www.blockchain.com/charts/difficulty accessed in February 2022

Other implementations of proof-of-work use different mechanisms to adjust the difficulty. This is over the scope of this document, but the key point to remember is that by adjusting the difficulty, we can increase or decrease the probability of finding a block over a single trial.

Termination and network partition By adjusting the difficulty, it is possible to adjust the mean rate at which miners will produce new blocks but the event of finding a single block is probabilistic. Indeed, while the probability p of finding a valid block over a single trial is known, there is no guarantee that a block will be found over $1/p$ trials. Thus in proof-of-work blockchains, we cannot rely on block time or expect that a transaction is final once sent to the miners.

Moreover, two miners may find a valid block at the same time. If two miners find a valid block at the same index and within a time period that is less than the propagation of a message over the whole network, then two valid blocks will exist in the system. As the communication is asynchronous, some subset A of the network will have received block b_i first and would have added it to their copy of the ledger. In the meantime, another subset of the network A' will have received and added block b'_i. In this case, we talk about forks. The probability of a fork happening is generally low. With Bitcoin, for example, it was estimated in [27] that the probability of a fork is about 1.69%.

In most proof-of-work blockchains, like in Bitcoin and Ethereum [67] [17], to resolve network partitions, the longest chain rule is applied. While the partitions A and A' have a

different state of the chain, the miners continue trying to find a new block. Miners of partition A try to build a block on top of b_i and miners of partition A' on top of b'_i. The first partition that comes off with a new block at index $i+1$ broadcasts it to the whole network. The protocol wants that honest miners chose the chain with the most work, i.e. the longest chain. Thus honest miners of the "shortest chain" will abandon their block and continue with the blocks of the other partition. In our example, if we assume that partition A came out with a block b_{i+1} before partition A', then honest miners of partition A' will drop block b'_i and continue mining over block b_{i+1}. It is possible that after a fork another fork happens, when for example a two blocks b_{i+1} and b'_{i+1} are found on top of b_i. The probability of such an event is low, less than the square of the probability of a single fork happening. However, in such events, the longest chain rule still applies.

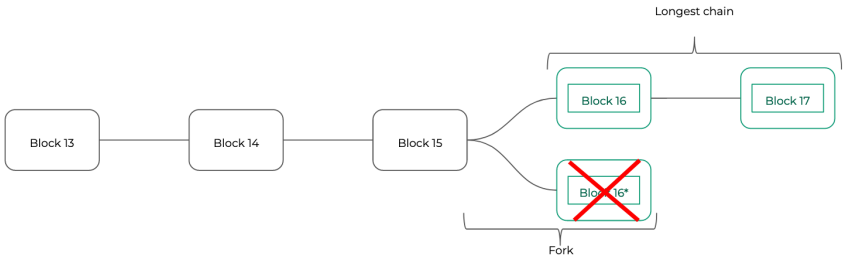


Figure 1.13: Representation of the longest chain rule

A fork is generally an event that we wish to avoid because it can lead to security issues. Indeed, b_i and b'_i may include different transactions and even conflicting transactions. Let us assume some party wishes to buy some instantaneously delivered digital goods with cryptocurrency. Let us assume that the checkout transaction is included in block b_i but not in block b'_i. If the seller is in the partition that chose block b_i, then he will have the proof that he has been paid and will deliver the digital good. At this point, everything is fine for both the buyer and the seller. But later on, when the network reunites in a single partition, block b_i may be abandoned in favor of block b'_i. The chain now has a different history where the checkout transaction has never been sent. The seller ends up losing his good without his payoff.

To prevent such risks, most merchants require several "confirmations" before considering a payment final. What we call confirmations, is the number of blocks that have been added to the chain after the block containing the transaction of interest. For example with Bitcoin, most merchants consider that three confirmation is safe enough. Indeed, the more blocks are added to the chain after a certain block, the safer it is to assume that a block will not be abandoned. This is because, as new blocks are added to the chain, more work is necessary to build a longer

chain and thus revert the block of interest.

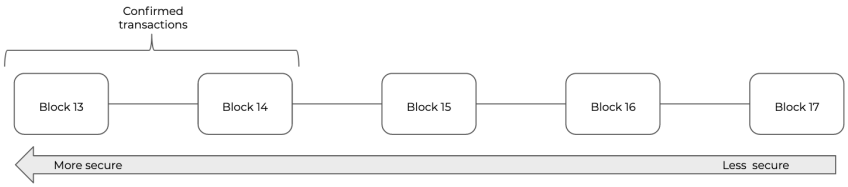


Figure 1.14: The higher the depth of a block is, the more work is required to revert it, the safer it is

Proof-of-work economics In most proof-of-work blockchains miners receive compensation for their work. In general, the protocol specifies that the miner who finds a new block is entitled to add a transaction at the beginning of the block that goes from nowhere to his wallet. That is how crypto-currency is created in Bitcoin for example. Besides that, miners often receive transaction fees that have been paid by the users. Indeed, by paying transaction fees along with a transaction, a user maximizes the probability of a miner picking up its transaction in priority when building a block.

In proof-of-work systems such as Bitcoin, the incentive model has encouraged miners to buy processing units that are more and more powerful. The idea is that behind each unit of crypto-currency created, there is an amount of time, money, and effort invested, giving the crypto-currency some intrinsic value. However, those "mining rigs", besides their cost and environmental impact have a huge energy consumption. At the time of writing, it is estimated that Bitcoin consumes 0.55% of the global electricity production [41]. In 2021 mining a single Bitcoin consumed as much energy as twelves households yearly. The implication of such energy consumption in a world where we can't ignore environmental concerns is beside the scope of this thesis, however, it is to be remembered that we might need alternatives to proof-of-work for global-scale application of blockchain technologies.

Proof-of-work performance Another limiting aspect of PoW is its relatively weak performance compared to centralized systems or other distributed blockchain protocols. At the time of writing, Bitcoin was capable of 5 transactions per second while visa achieves 1700, and other blockchain systems claim much more. Proof-of-work was indeed the first generation of blockchain protocols and has shown poor response to the increasing load of users. Scalability solutions are being developed mainly through the use of payment channels, which we will discuss later on. However, the industry is rapidly shifting to new blockchain protocols that

achieve a better throughput in terms of transaction per second, while consuming less electricity. Ethereum for example, the second-largest blockchain in terms of total market capitalization is transitioning from PoW to proof-of-stake.

1.2.2 Proof-of-Stake (PoS)

Proof-of-stake (or PoS) has been proposed as an alternative to the energy-intensive proof-of-work. The intuition behind proof-of-stake is that, instead of securing the blockchain with computational power, we use assets that are locked in custody. The Sybil protection mechanism is not secured with computational power but with the stake. In proof-of-work, the probability of being elected as the leader is proportional to the computational power of the miner. In proof-of-stake, the probability of being elected as the leader is proportional to how much money, or stake, has been placed in custody by the voter. In PoS, you don't "mine" blocks, rather the participants propose blocks when they are elected to do so. They get a reward for proposing valid blocks, and they get their stake, or part of it taken as a punishment when they propose a block that gets rejected. Voters are here to observe and vote for the proposed blocks. Then, like with PoW, PoS needs some kind of chain selection rule. At this point, PoS protocols fall under two main categories; chain-based PoS and Byzantine Fault Tolerant (BFT) style PoS.

In BFT-style proof of stake, validators are randomly assigned the right to propose blocks, but agreeing on which block is canonical is done through a multi-round process where every validator sends a vote for some specific block during each round, and at the end of the process all (honest and online) validators permanently agree on whether or not any given block is part of the chain. Note that blocks may still be chained together; the key difference is agreement over a block can come within one block, and does not depend on the length or size of the chain after it. This is refereed as finality but we will come to that later.

In chain-based proof of stake, the algorithm pseudo-randomly selects a validator during each time slot (eg. every period of 10 seconds might be a time slot) and assigns that validator the right to create a single block, and this block must point to some previous block (normally the block at the end of the previously longest chain), and so over time, most blocks converge into a single constantly growing chain.

In fact we say that BFT-style proof-of-stake solves a problem called the blockchain consensus problem. We will discuss that on later sections.

Under those two categories exist a plethora of PoS "flavors", and examining each one of

them is behind the scope of this document. However, to better understand the functioning and problematics of PoS we will proceed by case studying one major protocol of each category; Tendermint for BFT-style PoS and Ouroborous for chain-based PoS.

1.2.2.1 Chain based Proof-of-stake case study: Ouroboros

As a case study for chained-based proof-of-stake protocols, we chose Ouroborus. Ouroboros is the engine of the blockchain Cardano. It was first introduced in 2017 by Kiayias in [48]. It is at the time of writing the 6th top cryptocurrency in terms of total market capitalization.

Leader election process In Orouborous [48], time is divided into epochs (5 days) and epochs divided into slots (20 seconds). One and only one block is to be produced at a certain slot. To participate in the process, voters that are called "stakeholders" place some stake in custody. Then they synchronize their clock with the network as well as the blockchain's full history.

At the first slot of a new epoch e, the first step of the protocol is to elect a leader. To do so, each stakeholder will produce a random value and compare it to a number. Much like in proof-of-work, if the random value is less than this threshold number, the stakeholder is elected as the leader. The difference is that each process only has one shot at producing a random value, while in PoW, he had as many trials as his computing power would allow.

To produce the random value, the stakeholder uses a Verifiable Random Function (or VRF). A Verifiable Random Function is a deterministic function that takes as input some arbitrary seed and a private key, and that produces some random value. It is a one-way function, i.e., it is impossible to calculate the key or the seed from the output value. However, along with the random value, a VRF function produces a certificate that proves the correct execution of the random function. The specific VRF used in ouroboros is a weighted VRF. It takes into account the stake distribution for the frequency of a stakeholder election to be proportional to its stake (Fairness property).

Thus at the first slot of a new epoch e, stakeholders trigger the VRF with three parameters: (1) the stake distribution of the last block of epoch $e-2$, (2) the random seeds left by previous validators in blocks of epoch $e-1$ and (3) their private key. From this VRF execution, they obtain a random value v and a certificate of execution π. If v is below the threshold, they are elected to produce a block for this slot.

As the election process is probabilistic there can be no leader for a slot or several leaders

for a single slot. Indeed, the randomness of the function, while guaranteeing fairness over a large number of trials, does not guarantee that a single slot leader will be elected or that several can be elected at the same time. These odds tend to elect a single leader at a time, but the case where no leader is elected or several are elected is to be taken into account. This will be further discussed later on.

Block production If the stakeholder has found a random value v that is below the threshold, he is elected to be a leader. Inside the block, the leader includes the pending transactions, the random value v[2] and the proof certificate π. Before broadcasting the block, the slot leader generates a new secret key. The public key remains the same, but the secret key is updated in every step and the old key is erased. It is impossible to forge old signatures with new keys. And it is also impossible to derive previous keys from new ones. Finally, the slot leader broadcast the new block to the network.

Forks and long-range attacks As explained earlier, the leader election process in Orouborous is probabilistic. It is possible that no leader gets elected to produce a block in some slot which in this case, we just wait for a new slot. It is also possible that two or more leaders get elected. In this case, participants in the protocol will receive several valid versions of the chain from the leaders. To avoid those forks, a chain selection rule is applied: like in PoW, the longest chain rule. This is what makes Orouborous a chain-based PoS protocol. However, there needs to be a little variation from the PoW-style longest chain rule. In proof-of-work, each block represents a certain amount of work and thus it is very difficult to find several valid malicious blocks. In PoS, the cost of producing blocks is not important, allowing for producing a long alternative chain to try to manipulate the network. This is referred to as a long-range attack. This attack is made possible by the fact there is no extensive computation to produce a valid block in PoS.

Because of the threat of long-range attacks, the longest chain rule has been slightly modified. Instead of choosing the longest chain in terms of the number of blocks, honest nodes are invited to choose the longest chain in terms of transaction density. Indeed, it is very difficult for a malicious attacker to forge as many transactions as the honest majority.

[2]This random value will be used in further leader election. The protocol feeds of itself as a source of randomness hence the name Ouroboros, the snake that eats its own tail.

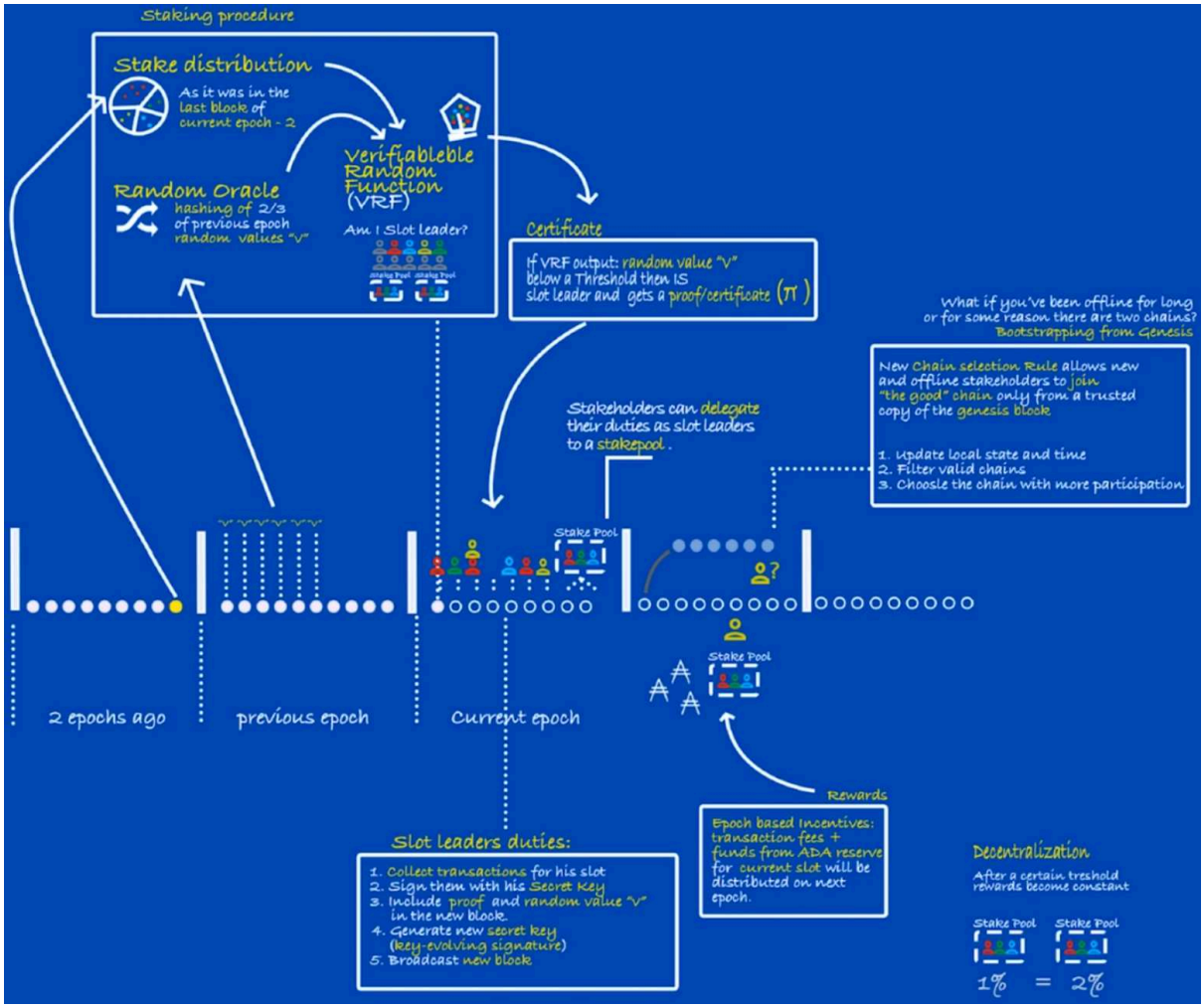


Figure 1.15: Sum up of the Orouborous protocol[57]

Stake delegation A single individual may not have enough stake and thus, voting power to ever see his random value below the threshold. At the time of writing someone owning 160$ worth of Cardano's crypto-currency is expected to be elected once every fifteen years. To allow small stakeholders to participate in the system, and to maximize the profit of big stakeholders, an individual can delegate their duties to a stakepool. The stakepool will increase its chances of being elected and redistributes the potential gains fairly among pool members.

1.2.3 Conclusion on chain based blockchain protocols

In this Section, we presented proof-of-work and proof-of-stake blockchain protocols. We have seen that the primary role of PoW and PoS was to select the process that will produce the next block. We have also seen through our case studies of Bitcoin and Ouroboros that two valid blocks can be found at the same time resulting in a partition of the network recognizing one state of the chain while another partition recognizes another. To resolve those forks, chain-based

protocols such as the one we studied use the longest chain rule. The network remains divided until a longer or denser chain is found. But the uniqueness of the chain is not guaranteed. This concept is known as finality. In blockchain systems, the finality refers to the affirmation that once added to the chain, a block and the transactions it contains will not be revoked. In other words, once a transaction has been included in a block and added to the chain, it will be engraved to the ledger forever. Bitcoin and Ouroboros do not satisfy finality because, as explained earlier, a fork can happen, eventually resulting in a block being revoked. Finality is a very desirable property for users as, when they provide a service or a good in exchange for a transaction, they do not expect it to be changed or reverted. When designing a blockchain protocol, finality is a crucial factor, but sometimes, finality has to be given up for availability.

We consider two types of finality; probabilistic finality and instant finality. In probabilistic finality, there is no guarantee that a block is final, however, the probability of a block being reverted reduces with new blocks being added to the chain. Bitcoin [67] is a probabilistic finality chain. Most merchant wait for 6 blocks before considering a transaction final. Instant finality guarantees that once added to the chain a transaction is final. In general consistency favoring systems provide finality, while availability favoring systems provide probabilistic finality.

Blockchain protocols work in practice by introducing waiting periods. After some time or number of blocks, the probability of the block of interest being revoked is very low and thus the transaction safe. However chain based protocol do not guarantee chain uniqueness or finality. Some other blockchain protocols satisfy the finality property. They solve a problem well known in the distributed systems literature as the consensus problem. In the following Section we will introduce the consensus problem and study the case of a blockchain protocol that solves it.

1.2.4 The byzantine consensus problem

The proof-of-work and proof-of-stake protocols that we have just reviewed try to solve a problem that is very similar to a well known problem in the academic literature. This problem, introduced as soon as 1983 by Fisher et al, in [32] is known as the consensus problem. Its blockchain version is introduced by Gramoli et al, in[37]. We will present this problem in this section but first let us introduce some context.

The Byzantine Generals problem In 1981 Leslie Lamport formalized a problem known as the Byzantine generals problem [56]. This problem can be summarized by the following

analogy: A general and his troops are besieging an enemy camp. The general's troops are divided into groups, each under the command of a lieutenant. The general has to issue a very important order to his lieutenants: the time at which the assault on the enemy camp must be launched. If all the troops of the general launch the assault at the same time, then victory is guaranteed. However, if they do not attack at the same time, they risk being caught off guard by the enemy. The general's troops are too far apart to communicate directly. To exchange messages, they have to send a messenger that can be intercepted by the enemy. To complicate the problem, the lieutenants may be traitors who desire the enemy's victory.

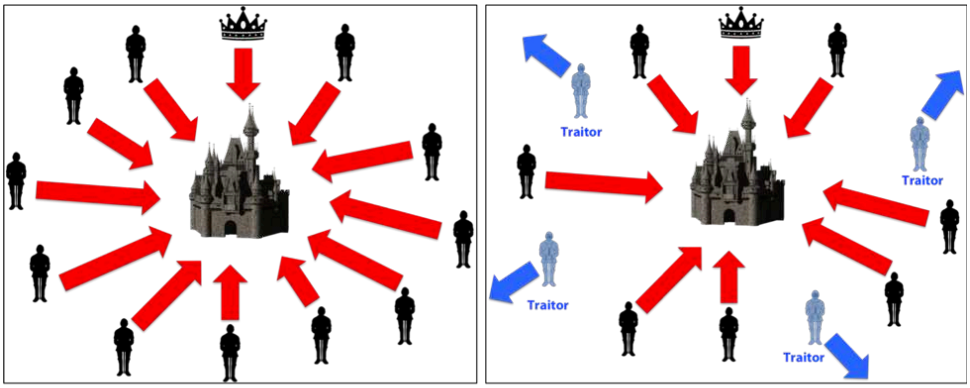


Figure 1.16: Byzantine Generals Problem representation

This problem serves as an analogy to what one can expect in a distributed system. If we were applying by analogy with blockchain systems, the time of the assault would represent the value of the new block to be added to the chain, the lieutenants and the general would represent the nodes of the network, and the messengers represent the fact that messages can get lost or corrupted in a distributed system. Finally, the traitors represent the fact that some actors in the system might be byzantine and act maliciously in their interests, rather than in the interest of the whole system. This problem is schematized in 1.16.

The blockchain consensus problem As stated in [37], in the context of a blockchain, the consensus problem can be expressed as follows:

The blockchain consensus problem: To answer the blockchain consensus problem, at a given block index, a blockchain system must satisfy the following properties:

- **Agreement:** No two correct nodes decide a different block.
- **Validity:** The decided block is a block that was proposed by one process.

- **Termination:** All correct processes eventually decide on some block.

A protocol solving the consensus problem has to guarantee that blocks are totally ordered, hence preventing concurrent appended blocks containing conflicting transactions. When designing such a protocol, one must keep in mind the characteristics of distributed systems we explained in the previous sections: (1) Some faults, byzantine or not, can occur to computers in the system (2) the network is not reliable and messages may fail to deliver, be delayed, or be out of order and finally, (3) there is no global clock to help determine the order of events.

The FLP impossibility In 1985, Fischer Lynch and Paterson [33] have proved that the consensus problem was impossible to solve in an asynchronous environment with one faulty process. This result is also known as the FLP impossibility theorem. This result also states that out of the three following properties; agreement, validity, and termination, it was only possible for an algorithm to satisfy two of them at the same time, given an asynchronous environment and failing processes.

But while this problem has proven to be impossible to solve in these particular conditions (asynchronous environment and byzantine processes), the FLP impossibility theorem does not state that consensus cannot be achieved "in practice". Indeed consensus can be achieved by making assumptions about the model. For example, the RAFT algorithm [74] solves the consensus problem assuming pseudo synchronous communication. They make use of timeouts to overcome the fact that in the asynchronous model, there is no upper bound on message delay. Moreover, it is also possible to implement consensus in real-world applications by weakening the properties of the consensus problem. Bitcoin [67] for example does not guarantee the agreement property. In Bitcoin, two correct nodes may decide on a different block. However the probability of this event happening is low, and there is a mechanism that incentivizes correct nodes to eventually agree on the same block.

In the following sections, we will see different approaches to solving the blockchain consensus problem "in practice".

The CAP theorem Another interesting result in the literature on distributed systems is the CAP theorem. In [14], Brewer shows that in any shard data system, it is only possible to achieve two of the three following properties:

- **Consistency**: A total order must exist on all operations such that each operation looks as if it were completed at a single instance. For distributed shared memory this means (among other things) that all read operations that occur after a write operation completes must return the value of this (or a later) write operation.
- **Availability**: Every request received by a non-failing node must result in a response. This means any algorithm used by the service must eventually terminate.
- **Partition tolerance**: The network is allowed to lose arbitrarily many messages sent from one node to another.

As blockchain systems can be assimilated to a shard data system, they fall under the rules of the CAP theorem. Ultimately blockchain systems will have to forfeit one of the three properties. When considering large scale, public blockchain systems in real world environments, partition tolerance is often not to be be forfeited as we cannot guarantee a reliable network. Thus blockchain systems are said to favour availability or consistency. Bitcoin [67] for example favors availability while BFT-style consensus such as Tendermint [51] favors consistency.

1.2.4.1 BFT style Proof-of-stake case study: Tendermint (Cosmos)

We chose Tendermint as a case study for Byzantine fault-tolerant style proof-of-stake because it is one of the earliest and most widely used PoS protocols. It was proposed by Jae Kwon [51] in 2014. Tendermint core, which we will be studying in this section is technically not a blockchain. It is a BTF-style consensus engine on top of which blockchain applications can be built. The main blockchain that is built on top of it is Cosmos [52], however, the Tendermint core consensus engine is open source and anyone can build a blockchain application on top of it.

Leader election process With Tendermint, the voters are called the validators. Whereas in proof-of-work any miner can join or leave the network anytime, Tendermint uses a different approach. The set of validators is curated by some external process and validators are expected to be online [88]. In PoW systems, there can be as many miners as we can imagine. It is estimated that there are is one million individual Bitcoin miners. On the contrary, BFT-style PoS like Tendermint require a reduced and known set of validators. At the time of writing, the main blockchain built on top of Tendermint, Cosmos [52], has a hard cap of 300 validators. This approach of having a fixed and curated set of validators has been chosen because of the

time complexity of the voting process. This will be further discussed later, but keep in mind that there is a fixed and known set of validator V of size n. The validators are identified by public-key cryptography and they communicate through P2P communication channels.

Time is divided into rounds, and at each round, a validator is elected to be the leader and to propose a block. The leader election step works like a weighted round-based mechanism. Each validator is associated with an accumulated priority number noted $A(i)$. At each round, the highest-ranked validator is elected to be the block proposer (the leader). The round-based ranking functions as follows: The voting power of a validator i is noted $VP(i)$. His voting power is equal to his share of the stake, so the more stake a validator has locked in custody, the more voting power he has. The total voting power of the set of validators is noted P. When a new round is initialized, the priority number of each validator is increased by $VP(i)$. Thus at each new round, $A(i) = A(i) + VP(i)$. If this results in validator i having the highest priority number of all validators, then he is the leader for this round and can propose a block. Once he has proposed a block, his priority number is decreased by the total voting power of the set, this is $A(i) = A(i) - P$.

This election process satisfies (1) determinism and (2) fairness properties [79]. This means that (1) at some round e, all correct processes agree on who is the leader and that (2) the frequency at which a validator is elected as a leader is proportional to its voting power.

Voting and decision process The voting process of Tendermint core is derived from the DLS [30] algorithm. The full protocol is described in [15].

Processes communicate through three types of messages, PROPOSAL, PREVOTE, and PRECOMMIT. The process consists of three steps. At each step, validators await for $2/3$ of the voting power of the validators to pronounce for or against the proposed block. The unfolding of the process is as follows: Once all correct processes have agreed on which of them is the leader at round r, the leader proposes a block. He does so by broadcasting a PROPOSAL message that contains the proposed block. On receiving this message, each validator evaluates the proposed block. If they think that the block is valid, they broadcast a PREVOTE message to the network containing the id of the block. If they think that the block is not valid, they broadcast a PREVOTE message containing nil instead of the block id. Validators listen to the network and await PREVOTE messages from the other validators. They store it in their memory until the sum of the voting power of the miners that sent matching PREVOTE messages is above $2/3$

of the total voting power P. If the weighted $2/3$ of the miners have sent a PREVOTE<blockId> message, then they proceed to send PRECOMMIT<blockId>. If $2/3$ sent PREVOTE<nil> then they proceed to PRECOMMIT<nil>. If $2/3$ sent PRECOMMIT<blockId> messages, then the block is committed, added to the chain and they proceed to a new block. Otherwise, if $2/3$ voted PRECOMMIT<nil>, the block is abandoned and they start a new round with a new leader.

A flowchart of the process is presented in Figure 1.17.

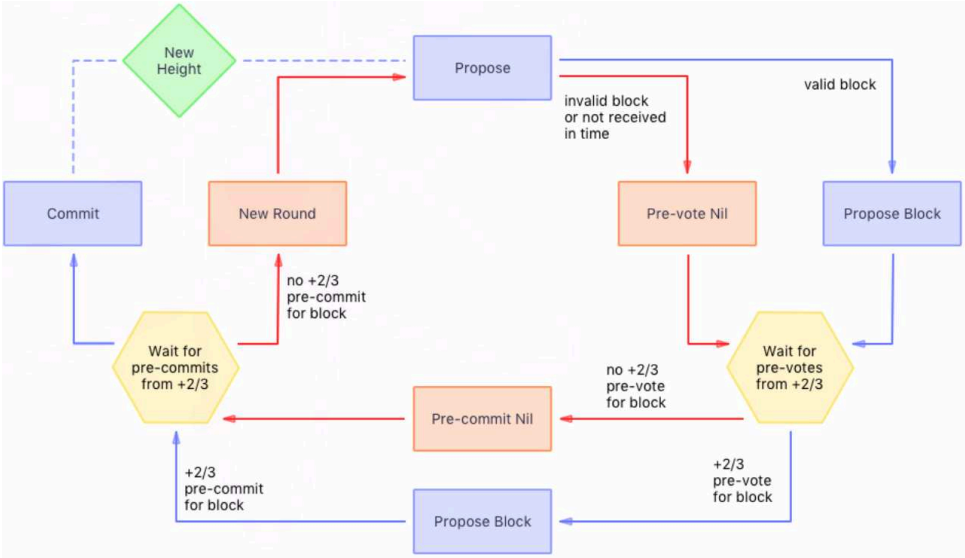


Figure 1.17: Flow chart of the Tendermint core voting process

During the voting process, if one or several validators fail to broadcast the messages they are expected to, they could threaten the liveness of the protocol. Indeed, if for example, the leader does not broadcast a PROPOSAL message, then the process would be blocked. To address this potential threat, the Tendermint consensus engine makes use of timeouts. At each step of a round, the validators set up a timer. If the timer expires before the condition he was set to is met, then the process will abort the round by voting nil or by requesting a new round.

Tendermint consensus properties As stated by the FLP impossibility theorem, it is impossible for a system to guarantee agreement, validity, and termination in an asynchronous environment with Byzantine processes. The Tendermint consensus engine is no exception to the rule. Indeed, Tendermint assumes partially synchronous communication. They assume that if a process sends a message at time t, then all correct processes will have delivered the message at $t + \Delta$. They assume that there is an upper bound on communication delays.

Moreover, the Tendermint consensus engine only satisfies the consensus properties if at least $2/3$ of the validators, in terms of voting power, are honest nodes.

However, in a partially synchronous environment and with at least $2/3$ honest validators, Tendermint achieves consensus with near-instant finality.

Another property of the Tendermint protocol is the relatively low number of validators the algorithm can handle. As explained earlier, Cosmos has a hard cap of 300 validators. This is very low compared to the estimated one million Bitcoin individual miners. This is due to the message complexity of the voting process of the Tendermint consensus protocol. Indeed, it has been shown in [4] that the message complexity of Tendermint was $\mathcal{O}(n^3)$. This means that the number of messages exchanged between validators increases exponentially with the number of validators, inducing latency.

Stake delegation A single individual may not have enough stake and thus, voting power to ever see his priority number on the top of the list. Indeed, bigger stakeholders have much more chances of being elected. In order to allow small stakers to participate in the system, and to maximize the profit of big stakeholders, an individual can delegate its voting power to a staking pool. The stake pool will increase its chances of being elected and redistributes the potential gains fairly among pool members.

1.2.5 Conclusion on blockchain protocols

Proof-of-work and Nakamoto consensus were the first trials at a practical distributed consensus. PoW as its core is a Sybil protection mechanism. Combined with a chain selection rule, it practically solves the distributed consensus problem. However, in systems like Bitcoin and Ethereum that make use of PoW, we can not state that they achieve distributed consensus. Indeed, the termination property of the distributed consensus problem is never totally satisfied. PoW systems are tolerant to at most 50% computing power of Byzantine process. By design, PoW is computation-intensive and thus consumes a lot of electricity.

Proof-of-stake has been proposed as an alternative to the energy-intensive PoW. PoS function similarly, but instead of using computing power as a mechanism to deter Sybil attacks, it uses stake held in custody, which can be taken away as a punishment if the validator misbehave. Some PoS systems are based on historic research on Byzantine Fault Tolerance. BFT-style PoS blockchain achieves distributed consensus with near-instant finality. However, these kinds of protocols are being criticized for their centralization. Indeed, because of the message complexity of the voting process, the number of validators must be reduced compared to PoW systems.

In an effort to guarantee some sort of decentralization, a mechanism named stake delegation allows participants to allocate their stake to a certain validator. In general, those systems are only tolerant to 30% of byzantine processes. Chain-based PoS does not achieve instant finality and does not satisfy the termination property of distributed consensus. However, they can tolerate up to 50% of Byzantine processes.

Table 1.1 sums up a few blockchain protocols, their properties and performances.

Blockchain	Leader election	Decision	Fault tolerance	Throughput (tx/second)	Confirmation latency
Bitcoin	PoW	Longest-chain rule	50% computing power	Sub-ten	10m-60m
Ethereum	PoW	Longest-chain rule	50% computing power	Tens	6m-10m
Tendermint (cosmos)	PoS-based round-robin	BFT	31% stake	Thousands	1s-3s
Orouborous (cardano)	PoS-based committee election	Longest-chain rule	50% stake	Hundreds	1m-10m

Table 1.1: Comparison of different blockchain protocols. Some figures have been found in [95]

Termination and network partition in proof of work systems As explained in Section 1.2.4, the FLP impossibility theorem states that out of the three properties of the consensus problem, an asynchronous distributed system with faulty processes can only satisfy two at a time. Proof-of-work-based blockchains do not formally satisfy the termination property.

Considering the CAP theorem [14], a blockchain system must chose at most two between consistency, availability and partitions tolerance. Proof-of-work-based blockchain systems generally forfeit the consistency property.

Proof-of-work is not a consensus protocol Technically, proof-of-work is not a consensus protocol. It is a Sybil protection mechanism and a leader election protocol [25]. A Sybil attack [16] wherein a reputation system is when an attacker forges several fake identities to compromise a disproportionate share of the system. In a blockchain system, a Sybil attack would consist in forging a lot of voter identities and votes for a block of your choice [2]. In PoW, the chances of being elected as the block producer are proportional to the computing power a miner has. Yet,

it is impossible to fake having a lot of computing power, because this claim requests a proof; the proof-of-work. So the first role of PoW is to prevent Sybil attacks.

The second role of PoW is to be a leader election algorithm. The elect step of the consensus algorithm. Proof-of-work achieves this too, by setting up a cryptographic race between miners and selecting the first one as the leader.

The reason why proof-of-work is not a consensus protocol is that, once the leader has been fairly[3] selected and has proposed a block, termination is not reached. Indeed, another block might have been found resulting in a fork. Even with the longest chain rule is applied, finality is never reached. The probability of a block being removed decreases with new blocks being added to the chain. But the probability never reaches zero. Thus Bitcoin-style PoW does not satisfy the termination property and cannot be considered a consensus algorithm. In the literature, it is often referred to as the "Nakamoto consensus", because of the author of the Bitcoin white paper.

1.3 Blockchain evaluation

1.3.1 A blockchain for each use case

When the first blockchain systems arose, at the beginning of the last decade, enthusiasts were expecting that one blockchain would take the lead and become the one and only digital cash system, that would replace central banking [63]. At that time, the main candidate was Bitcoin, which represented more than 80% of the total cryptocurrency market capitalization. Ten years, and a few blockchain systems later, Bitcoin dominance has been divided by two and new systems with different properties have been deployed. The reason for this is that Bitcoin can not yet fulfill all the use cases a digital version of cash can generate. For example, Bitcoin can only achieve around seven transactions per second, and transactions are confirmed in about one hour. Those kinds of performances are not satisfying for an in-store payment solution. However, as the number of Bitcoins to be mined is limited, it does not suffer inflation and its price tends to go up. That makes Bitcoin a reasonable store of value as gold is. Some other blockchain systems present different properties that can make a fit for different use cases. Cosmos [52] for example achieves near-instant finality, high transaction throughput, and low transaction fees,

[3]Fairly meaning that its chances of being elected were proportional to its computing power

which makes it a good candidate for in-store payments. The Ripple blockchain [83], controls who can participate in the network and specialized in inter-banks settlements. Monero [71] specializes in private and untraceable transactions, which makes it a good candidate for a censorship resilient system.

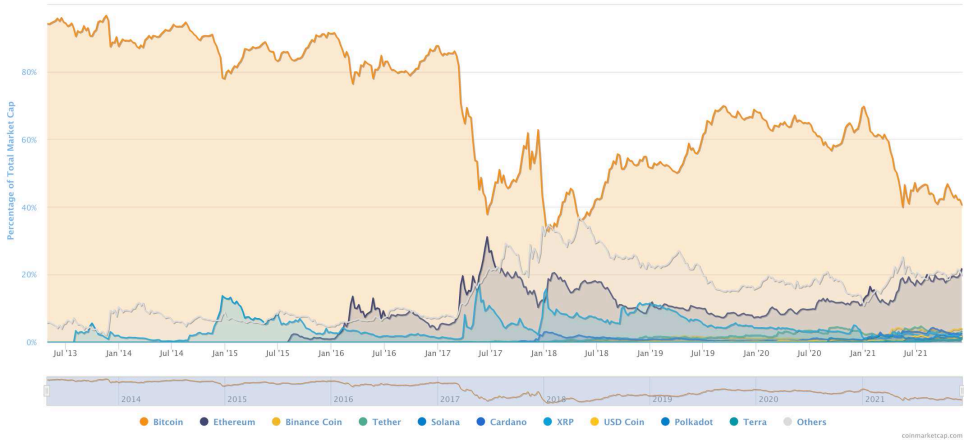


Figure 1.18: Major cryptocurrencies market capitalization compared

1.3.2 The blockchain scalability trilemma

As we can see, different blockchain systems satisfy different properties that make them correspond to different use cases. In [18], Vitalik Buterin, the co-founder of Ethereum, defined three properties that blockchain systems tend to achieve; scalability, security, and decentralization. He defines those properties as follows:

- **Scalability** is the ability of a blockchain system to process more transactions per second when more nodes join the network.
- **Security** corresponds to the ability of a blockchain system to resist a large number of Byzantine processes trying to attack it. This can be assimilated to the fault tolerance parameter as presented in Table 1.1.
- **Decentralization** refers to the fact that a blockchain runs without the need for big centralized actors. Typically, a decentralized blockchain network can be joined by anyone with a consumer laptop.

Buterin defines the problem as the blockchain scalability trilemma. Much like the CAP theorem, the blockchain scalability trilemma states that by favoring two of the three properties, any blockchain system ultimately makes concessions on the third property. Thus each type of blockchain can be placed in one of the arcs of the triangle presented in Figure 1.19. Bitcoin for

example is tolerant to 50% of byzantine processes. It is also decentralized, as anyone can join the network with consumer laptops. However, Bitcoin's transaction throughput is limited and is not correlated with the number of participants in the network. It is not scalable. Thus Bitcoin can be placed in the arc that connects the secure vertex and the decentralized vertex.

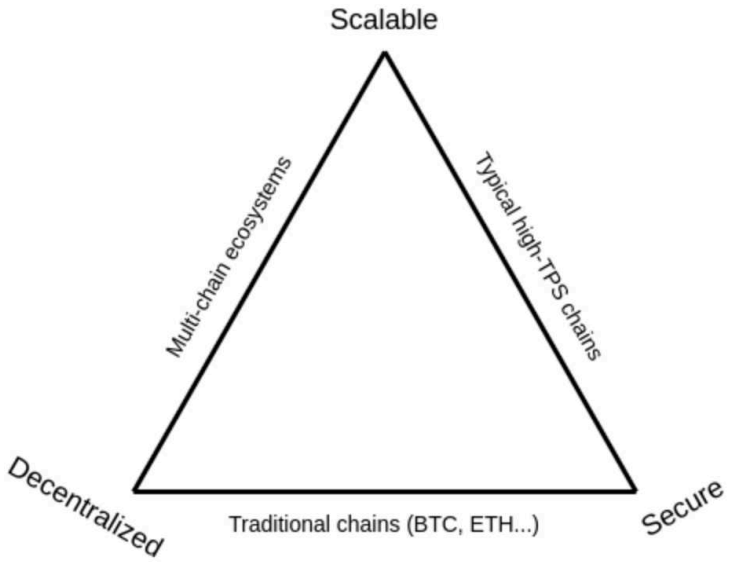


Figure 1.19: Buterin's blockchain scalability trilemma. From [18]

Each blockchain can be placed somewhere on this triangle. Tendermint based blockchains such as cosmos [52] are scalable. It achieves a great number of transactions per second and is quite tolerant to 31% of Byzantine processes in terms of stake. However as explained earlier, the number of validators is limited, and joining the validator set is an expensive and long process. Thus cosmos would be on the arc connecting the scalable and secure vertex.

Note that unlike the CAP theorem [14] or the FLP impossibility theorem [33], the blockchain scalability trilemma is not a theorem and has not been proved. It is an observation that has been made over the state of the art of existing blockchain systems. It is more of a framework to evaluate blockchain systems than a proven result.

Finding a blockchain protocol that could satisfy all three properties may be possible and is currently a very active field of research[18, 38, 61, 91]. The most promising projects focus on a technique named sharding. Sharding is a way of partitioning the blockchain to spread out the computational and storage workload across the network. Each node isn't responsible for processing the entire network's transactional load, but instead, each node only maintains information related to its partition or shard. Sharding is over the scope of this paper but it is to be noted that the scalability trilemma is not an impossibility result but an observation.

1.4 Application layer

1.4.1 Smart contracts

High level overview Early adopted blockchains such as Bitcoin only supported simple use cases like token transfers. The following generation of blockchains introduced the concept of smart contracts; they are Turing complete scripts hosted and executed on the blockchain through the same transaction system. Smart contracts are programs that are stored on the blockchain and executed by the validators. Those programs have their own memory and can perform arbitrary computation. They inherit the properties of blockchain systems; their state, inputs, outputs, and execution is publicly readable and auditable. Smart contracts have been popularized by the Ethereum framework [93] in 2014. They have now found many applications in fields that require decentralization such as voting, gaming, crowdfunding, accounting, insurance, finance, etc. The intuition behind smart contracts is that since a blockchain is at its core, a sort of data store, why not store programs in it. As the logic of the program exists in the same environment as the crypto-assets, they can natively interact with them, allowing for a contract to transfer, store and receive crypto-assets.

Figure 1.20 presents an example of a smart contract written in Solidity language. Solidity was originally developed by core contributors of the Ethereum platform. It is now very popular among smart contract developers and is used in several blockchains platforms besides Ethereum.

This contract acts as a time-locked vault. It could be used for example, as a child saving account, that the parents fill up and that can only be spent by the child when he reaches majority. Once deployed to a blockchain, anyone will be able to send funds to the contract through the $setupVault$ function. It is required that the person setting up the vault provides a beneficiary and a number of blocks corresponding to the locking period. The funds will be then locked inside the contract until the beneficiary claims them. The beneficiary is identified by its blockchain address, and the lock time is expressed in a number of blocks. Once the number of new blocks has been produced, the beneficiary can call the $unlockVault$ function, triggering an asset transfer from the contract to the beneficiary. As in most blockchains, the block production frequency, or block time, has a constant mean, it is possible to predict approximately when a particular block will be produced. For example in Bitcoin, the block time is ten minutes. Thus if the person setting up the vault provides 720 as the $lockTime$ parameter, then the beneficiary will

```solidity
pragma solidity >=0.7.0 <0.9.0;
/**
 * @title Vault
 * @dev Implements a crypto-currency vault
 */
contract ClockLockedVault {

    struct Vault{
        address beneficiary;
        uint amount;
        uint lockAtBlockNumber;
        uint lockedUntilBlockNumber;
    }

    mapping(address => Vault) public vaults;

    /**
     * @dev Sets up a vault for the beneficiary
     * @param _beneficiary address of the beneficiary of the vault
     * @param _lockTime number of blocks before the vault can be oppened
     */
    function setupVault(address _beneficiary, uint _lockTime) public payable {
        Vault storage v = vaults[_beneficiary];
        v.beneficiary = _beneficiary;
        v.lockAtBlockNumber = block.number;
        v.lockedUntilBlockNumber = block.number + _lockTime;
        v.amount = msg.value;
    }

    /**
     * @dev Unlocks and empties a vault
     */
    function unlockVault() public {
        // Verify that the sender owns a vault and that the vault is not empty
        require (vaults[msg.sender].amount>0);
        // Verify that vault lock time has expired
        require (block.number >= vaults[msg.sender].lockedUntilBlockNumber);
        //Empty vault
        vaults[msg.sender].amount = 0;
        // Transfer the content of the vault to beneficiary
        payable(msg.sender).transfer(vaults[msg.sender].amount);
    }
}
```

Figure 1.20: Example of a vault smart contract. This contract allows one to lock some crypto-assets inside the contract for a beneficiary until a certain block height is reached.

be able to unlock in approximately five days, because there are $720 * blockTime_{bitcoin} = 7200$ minutes in five days.

There exist several ways to implement smart contracts; scripting like in Bitcoin [67], virtual machines like in Ethereum [17] and containerization like in Hyperledger [5]. As in this document, we will only be discussing virtual machines and scripting, we will only cover those two ways of implementing smart contracts. We will do so by studying the case of Ethereum with the Ethereum Virtual Machine and Bitcoin with Bitcoin SCRIPT.

The case of Bitcoin Script Bitcoin Script is the programming language used in Bitcoin to process transactions. It is not considered a smart contract programming language because it is intentionally not Turing complete. It only allows for performing simple programs such as multi-signature wallets or time locks. Bitcoin Script is a stack-based language where instructions, known as OP_CODE for operation code, are executed from left to right.

Before considering the case of Bitcoin Script, let us introduce the Bitcoin UTXO model. The Bitcoin blockchain does not have an account-based system as Ethereum has. Rather than keeping track of accounts and balances, Bitcoin maintains a list of unspent transaction outputs or UTXOs. UTXOs are indivisible chunks of Bitcoin currency locked to a specific owner, recorded on the blockchain, and recognized as currency units by the entire network. By keeping track of all unspent tokens, we can virtually build a list of accounts and their corresponding balances. The account balance of a user is calculated by its blockchain client application. The blockchain client application, also known as a wallet, calculates the balance of the user by parsing the blockchain and aggregating all UTXO belonging to that user. More precisely, that user's public key.

Each Bitcoin transaction contains two scripts, an unlock script and a lock script. The unlock script is supposed to match the desired UTXO while the lock script contains what is expected from others in the future to use that output. In Bitcoin, the ownership of a token can be narrowed down to owning the keys that can unlock a UTXO.

The simplest transaction we can dissect is a Pay to Public Key script or P2PK. This type of script locks an output to a public key. Then the owner of the private key corresponding to this public key will be able in the future to unlock this output. In other words, a P2PK is a transaction where you transfer the ownership of a token to another public key. It is the simplest transfer.

Figure 1.21 presents the execution of a P2PK script. The P2PK script is the unlock script of a particular UTXO. The script only contains two things: a public key and the OP_CHECKSIG operation code (stage 1 of Figure 1.21). In order to spend the UTXO, it is required to provide a signature corresponding to the public key (stage 2). Then the script will put the signature and the public key inside the stack (stages 3 and 4). The OP_CHECKSIG operation is applied to the two variables that are in the stack (stages 5, 6, and 7). If the signature matches the public key, the OP_CHECKSIG operation places 1 inside the stack (stage 8), meaning that the script finished successfully. The transaction should also contain a lock script that provides further

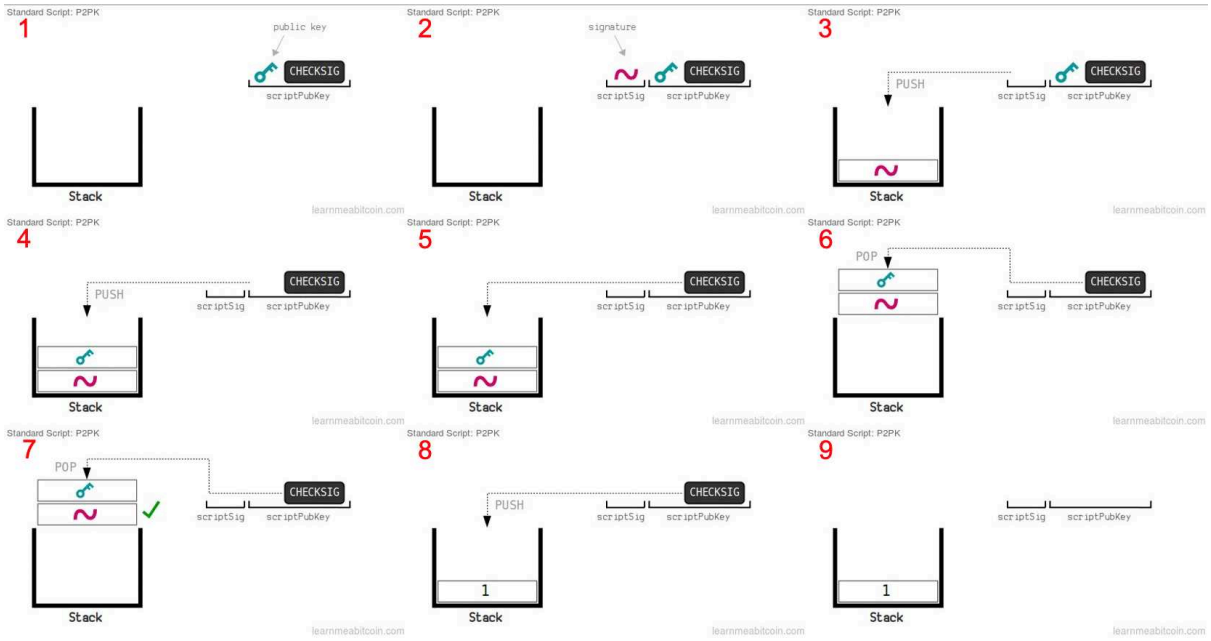


Figure 1.21: Pay to public key (P2PK) script execution in Bitcoin

instructions to spend the newly generated UTXO.

P2PK script is the simplest script in Bitcoin. However, it is possible to build more complex applications with the available operation codes. For example, it is possible to build a lock script that will only allow spending the UTXO after a certain number of blocks have been created. Or it is possible to create a lock script that instead of one signature, requires multiple signatures from multiple public keys. This would result in what we call a multisignature scheme. The script presented in Figure 1.22 for example, will require that at least two of three public keys provide a signature to unlock the output. Bitcoin Script applications will be further discussed in Chapter 3.

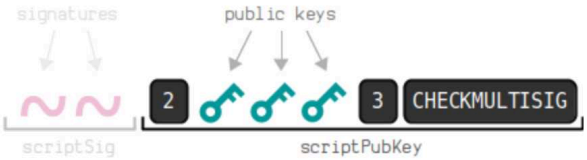


Figure 1.22: Pay to public Multisig (P2MS) in Bitcoin

The case of Ethereum and the EVM While languages like Bitcoin SCRIPT provide some functionalities, they are still limited. Bitcoin SCRIPT is not Turing complete and does not provide the ability to make loops. Ethereum [17] popularised smart contracts by providing a new approach to smart contracts that will be executed in another environment, the Ethereum Virtual Machine or EVM.

Unlike Bitcoin which is UTXO-based, Ethereum is account-based. The Ethereum blockchain keeps track of each account and its balance, much like a bank account. There exist two types of accounts in Ethereum; Externally Owned Account or EOA, which are controlled by a user via private keys and contract accounts. These accounts are controlled by a smart contract code. The contract accounts do not have private keys and thus can not initiate transactions. However, when triggered by an EOA, they can transfer tokens or call another contract.

A smart contract is a type of account that possesses a memory and some code. Through transactions, the code can be executed resulting in a new state.

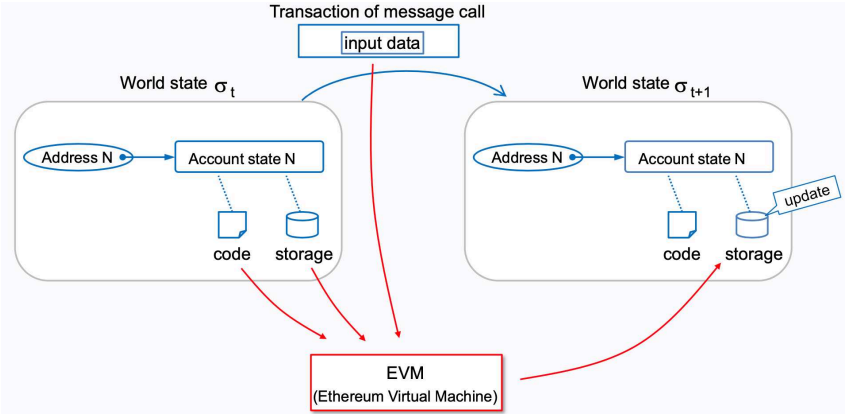


Figure 1.23: State transition of an Ethereum smart contract with the Ethereum Virtual Machine (EVM)

Some blockchain systems have designed their own programming language and execution environment for smart contracts while others leverage existing technologies. Ethereum has its own smart contract programming language called Solidity. Solidity is a Turing complete language that is then compiled to Bytecode. Via a transaction, the compiled binary or Bytecode is deployed to a contract account and executed by the Ethereum Virtual Machine. During the compilation, an Application Binary Interface or ABI is generated. This interface references the contract methods, how inputs should be formatted, and how to interpret the outputs. The memory of an Ethereum smart contract is persistent. Its data and variable can be saved and used through several executions.

The Ethereum Virtual Machine exists as one single entity maintained by thousands of connected computers running an Ethereum client. Indeed, computation is executed by miners when creating new blocks. They get rewarded for this through a mechanism based on gas. Gas refers to the unit that measures the amount of computational effort required to execute specific operations on the Ethereum network. Each operation has a fixed cost in terms of gas that has to be paid by a function caller for the miners to execute the required computation.

To incentive miners to include their transaction in a block rapidly, a function caller can increase the gas price they pay, as the cost of a function call is $cost = gas_{cost} * gas_{price}$. Miners tend to prioritize transactions that have a high gasp price. Table 1.2 presents a sample of gas cost for a few EVM operations.

Operation	Gas cost	Description
ADD	3	Addition operation
MUL	5	Multiplication operation
SUB	3	Subtraction operation
SLOAD	200	Load word from storage
DIFFICULTY	2	Get current block difficulty

Table 1.2: Sample of gas cost in Ethereum. Taken from Ethereum's yellow paper [93]

Bitcoin Script versus EVM analysis Compared to what the EVM allows in terms of computation, Bitcoin Script offers very few possibilities. Indeed, Bitcoin Script is by design, not Turing complete. It does not allow to perform programmatic loops or script reusing. This was intended by the Bitcoin creators because (1) a Turing complete language could end up in an infinite loop and (2) because if scripts could be reused, then an adversary could plan replay attacks (replay the message sent to a network by an attacker, which was earlier sent by an authorized user). Ethereum and the EVM have solved both of these problems with their account-based approach and the gas system. The account base system allows linking every account with a transaction count named the nonce. For every new transaction sent by an account, its nonce is incremented, thus enforcing that the transaction is unique and cannot be replayed. The gas system solves the infinite loop problem. Indeed it is trivial to prove that an infinite loop is impossible because it will result in infinite cost. Moreover, there is a hard cap on the maximum computation a transaction can result in; the "gas limit". In conclusion, I would argue that virtual machine based execution in blockchain systems is superior in terms of the possibilities it offers to developers. That is why Bitcoin is considered a first-generation blockchain while Ethereum is considered a second-generation blockchain. In this framework of evaluation, third-generation blockchains offer the same properties as second-generation ones, but while doing so, they are less electricity and resource-intensive.

Conclusion on smart contracts The semantic of the word smart contract is misleading because they are neither smart nor contracts. Technically a smart contract is just a program. However, when deployed to a blockchain, the compiled code of this program is publicly available

and thus auditable. The execution of the program and the inputs and outputs are visible to anyone and thus auditable. This is why those programs are called smart contracts; when using a smart contract, the user agrees on a set of rules, this is, the smart contract's code, and the rule will be enforced during execution because it will be verified by the network. These properties of smart contracts make them a very good tool for decentralized applications or Dapps, which will be discussed in the following section.

1.4.2 Decentralized applications (Dapps)

Overview A decentralized application or Dapp is a web application that instead of using a centralized server, is deployed over a decentralized network. Tor the onion router [28] [81] for example, is an application that runs on a peer-to-peer network. It implements the onion-routing technique which consists in encapsulating a message in several layers of encryption to achieve anonymous communication over a network. Stremio and Popcorn Time are torrent clients that also rely on a peer-to-peer network to store and serve video content. They implement the BitTorrent protocol [58], which enables users to distribute data and electronic files over the Internet in a decentralized manner.

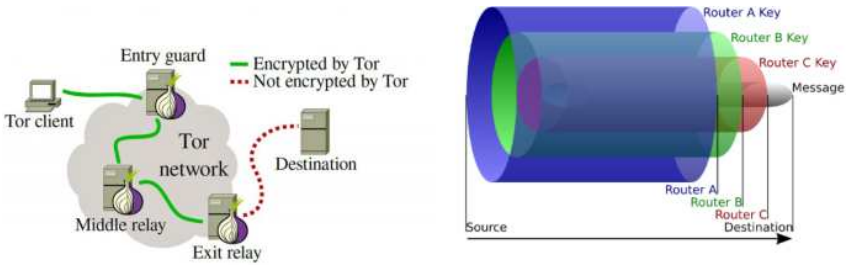


Figure 1.24: The onion-routing technique encryption representation and network topology

In the context of blockchain, a Dapp is an application that somehow relies on a smart contract to store data and execute computation. In general, Dapps source code is open-source and publicly available. The fact that Dapps are deployed to a distributed environment makes them resilient to failures and censorship. For example, we could think of a distributed version of Twitter where tweets would be stored on a smart contract deployed to some blockchain. As transactions on the blockchain are immutable, it would be impossible, even for the author, to delete a previously posted message.

Use cases of Dapps range from video content serving to financial applications, social media platforms, gaming, etc. Thanks to web browser wallet extensions such as metamask [1], users can interact with a smart contract directly from their web browser.

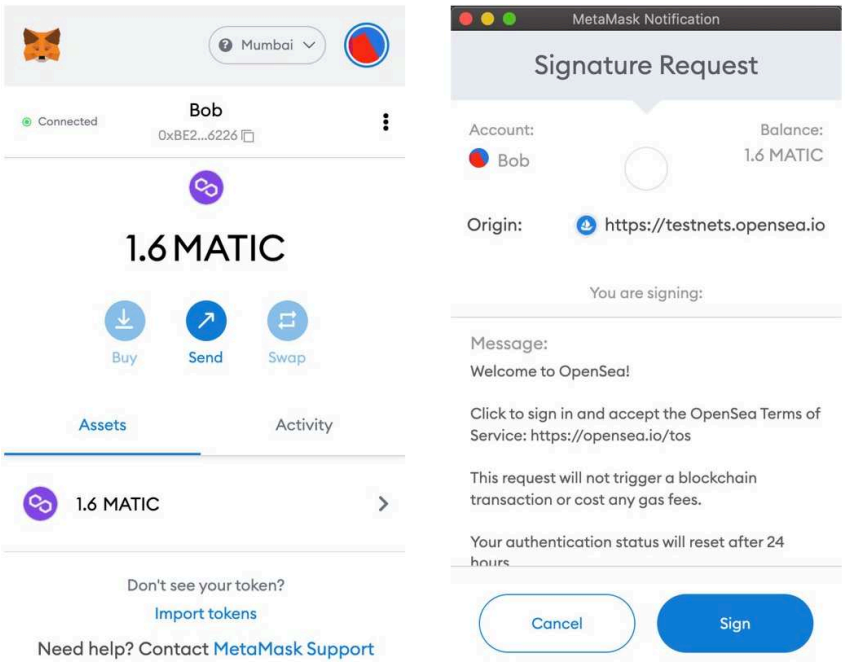


Figure 1.25: The metamask wallet web-browser extension. The screenshot on the left shows the metamask wallet extension. It shows that the wallet's current balance is 1.6 MATIC. MATIC is the native token of a blockchain named Polygon. Under the name of the wallet, "Bob" is written the address of the account. The picture on the right shows a metamask pop-up request for signature. When interacting with a Dapp, the application front-end will generate transaction messages for the user. Before sending the transaction, the wallet will request the user to sign the transaction message.

The Dapp that crashed Ethereum: CryptoKitties CryptoKitties [45], deployed in 2017, is known to be one of the first, and most popular at the time, gaming Dapps on the Ethereum blockchain. This game allowed players to purchase, collect, breed, and sell virtual cats. When a user buys a kitty, a smart contract will assign it random attributes, creating a clearly recognizable but unique avatar. The avatar can have different backgrounds, skin colors, clothes, and expressions. The game consisted of five smart contracts; a core contract that recorded all "kitties" attributes and owner information, an auction and offer contract that allowed users to put their kitties token to auction, a contract that allowed users to rent their kitties, and finally a contract that was responsible for generating the "genes" or attributes of newborn kitties. Users would pay with Ethereum native currency ether to purchase their kitties tokens.

The core contract is a token contract. A token contract is a program that keeps track of token ownership and manages the creation and transactions of those tokens. In the case of Cryptokitties, the tokens contained the metadata that defined the "genes" or attributes of the kitties.

The game was released in October 2017 and quickly gained in popularity. Some tokens were

sold for more than a hundred thousand U.S. dollars worth of Ether as soon as December 2017. It has been shown in [45] that at the beginning of December 2017, CryptoKitties accounted for more than 10% of the transactions of the Ethereum main network, causing network congestion. This event has shown that Ethereum was not yet capable of hosting large-scale applications. In the meantime, it has also shown that there was a real public attraction to such decentralized applications.

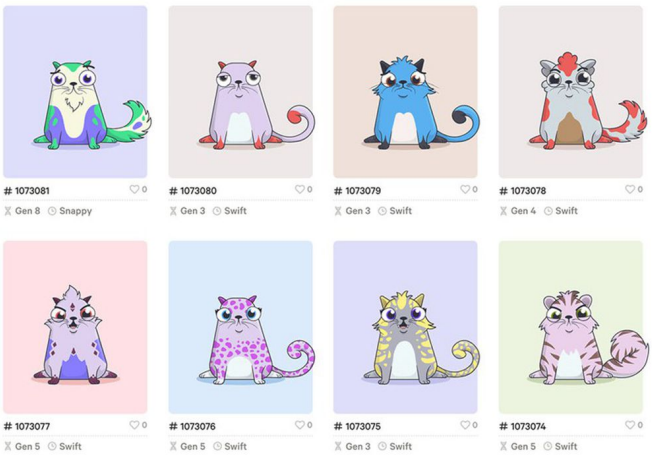


Figure 1.26: Catalogue of the CryptoKitties Decentralized application

Decentralized applications storage As smart contract programming languages offer the possibility for arbitrary computation, thus virtually any application could be built on top of a smart contract. Applications such as social media, video streaming, or newspapers could be built on top of a smart contract. However, storage and computation on the blockchain are expensive because validators or miners have to be paid for their work. Thus large files cannot be hosted on blockchains. Moreover, some data does not need to be replicated over every node. If we take the example of BitTorrent, each file doesn't need to be replicated over each node for the application to function. Thus, many Dapps are not fully hosted on a blockchain. Rather, some key components and computation are on a blockchain's smart contract while other components are hosted by decentralized services that are not blockchains but are very much compatible with them. IPFS [10] for example is a file-sharing system that can be leveraged to store and share large files more efficiently. It relies on cryptographic hashes that can easily be stored on a blockchain [85]. Systems such as IPFS are often used by smart contract-based decentralized applications in order to store large files or to serve their front-ends.

In Figure 1.27 we have an example of a decentralized application that uses IPFS as a storage

system and a smart contract has a tempered proof-of-ownership. This Dapp can be used by photographers for copyright purposes. The photographer uploads the picture to the Dapp (1). Then the Dapp uploads the picture to IPFS. The file is stored on IPFS and its address is the file's hash (2). Then the application publishes the file's hash along with the photographer's copyright information to a smart contract (4). With this application, a photographer can prove that he is the original author of the picture with tamperproof. This Dapp uses both a smart contract on a blockchain and a distributed storage. Storing the full picture file to the blockchain would be unnecessary and too expensive.

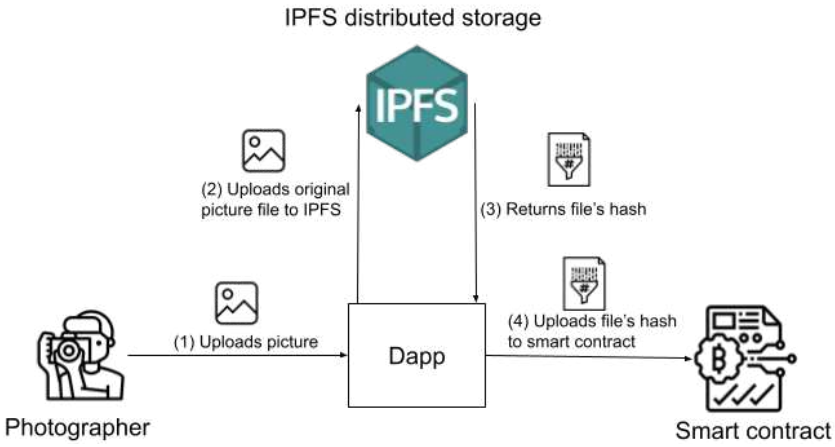


Figure 1.27: Example of a Dapp used for picture copyrighting. The original image's file is uploaded and stored to IPFS. The file's hash, which is its address in the IPFS protocol is then uploaded to a smart contract on a blockchain.

Tokenization While we have seen that blockchains are not very good at storing large chunks of data, they are a very good candidate for another use case; tokenization. Tokenization is a process during which some sort of asset is stored on a smart contract under the form of a token. Tokens are owned by some blockchain address. The token contract keeps track of each token, its attributes, and its owner. The owner can transfer a token to another address just like a regular crypto-currency. Tokens can be fungible, meaning that all tokens are identical, or non-fungible, which means that they contain attributes or metadata that makes them unique. Cryptokitties are an example of a non-fungible token. Indeed, each Cryptokitty token possesses different attributes that define the genes of the kitty. An example of a fungible token would be Tether [92] that tokenizes U.S. dollars. Each token created (or minted) by the Tether organization is backed by a U.S. dollar stored in a real-world reserve. Thus on cryptocurrency markets, Tether tokens are exchanged at a value close to one U.S. dollar.

Tokenization can be applied to a variety of assets. We already saw gaming artifacts and fiat currency but it can also be applied to bonds, securities, real estate, etc. Token contract standard interfaces have been developed in the industry in order to have homogeneous interfaces. Thus tokens representing a variety of assets can be exchanged on trading platforms.

Figure 1.28 presents a simplified version of the most used token contract on the Ethereum blockchain, the ERC20. At line 3 the variable $\_balances$ is the data structure that keeps track of all token holders and their balances. The $balanceOf$ function at line 20 allows checking a particular account's balance. At line 25, the $transfer$ function transfers $amount$ tokens from the transaction sender's account to a recipient.

```
1   pragma solidity ^0.8.0;
2   contract ERC20  {
3       mapping(address => uint256) private _balances; // List of accounts and their balances
4       uint256 private _totalSupply;
5       string private _name;
6       string private _symbol;
7
8       //Sets the values for {name} and {symbol}.
9       constructor(string memory name_, string memory symbol_) {
10          _name = name_;
11          _symbol = symbol_;
12      }
13
14      //Returns the amount of tokens in existence.
15      function totalSupply() public view returns (uint256) {
16          return _totalSupply;
17      }
18
19      //Returns the amount of tokens owned by `account`.
20      function balanceOf(address account) public view returns (uint256) {
21          return _balances[account];
22      }
23
24      //Moves `amount` tokens from the caller's account to `recipient`.
25      function transfer(address recipient, uint256 amount) public returns (bool) {
26          require(_balances[msg.sender] >= amount);
27          _balances[msg.sender] -= amount;
28          _balances[recipient] += amount;
29          return true;
30      }
31
32      //Creates `amount` tokens and assigns them to `account`, increasing the total supply.
33      function _mint(address account, uint256 amount) public {
34          _totalSupply += amount;
35          _balances[account] += amount;
36      }
37  }
```

Figure 1.28: Simplified version of the ERC20 token smart contract standard written in Solidity.

These kinds of tokens have allowed people to raise funds during a process called Initial Coin Offering or ICO. Much like a classical IPO, an ICO is a fundraising during which a company sells tokens to investors in order to finance its development. The fund raised can be used to hire people, develop a product, rent offices, etc. Depending on the project, the token issued during the ICO can then be used as a utility, i.e. to pay for the company's services, or

as a security, a share of the company that can be traded and exchanged. ICOs gained a lot of traction at the end of 2017, raising an estimated total of $20 Billion this year. However, lots of projects were over-financed, wrongly advertised, misleading, or even pure fraud. Most of the projects that raised funds during this period don't exist anymore.

In some cases, the token represents voting power for the company's decisions. Much like in a regular company where stake represents voting power at the board meeting. In the case of a company that raised funds and issued tokens during an ICO, the token represents voting power. We talk here about Decentralized Anonymous Organisations or DAOs. DAOs function like a regular company; when the company has to make a decision like choosing to finance a project or not, buy an estate, or re-brand a product, they submit the decision to the stakeholders that will vote for or against the resolution. In a DAO, the token represents the stake, and the vote is executed in a smart contract. The more tokens a person owns, the more voting power he has. MakerDAO [64] for example is a company that uses this kind of on-chain voting governance. This company issues a stable coin i.e. a coin that value is backed by a government's central currency like U.S. dollars. The stable coin issued by MakeDAO is called the DAI token. They also issued the MakerDAO governance token which represents voting power during on-chain governance decisions. Figure 1.29 is a screenshot of the MakerDAO governance website. We can see a proposal that is submitted to vote. Holders of the MakerDAO token can choose to support this proposition by baking it with their voting power. Here the proposition "Parameter Changes, Switching MKR Vesting Source" is supported by ten bakers, totaling 74,810 MKR (the MakerDAO token symbol). If this represents more than half of the total voting power then the proposition will be adopted. On the top right of Figure 1.29 we can see a button "Connect wallet". This is because stakeholders submit their vote through a smart contract on-chain.

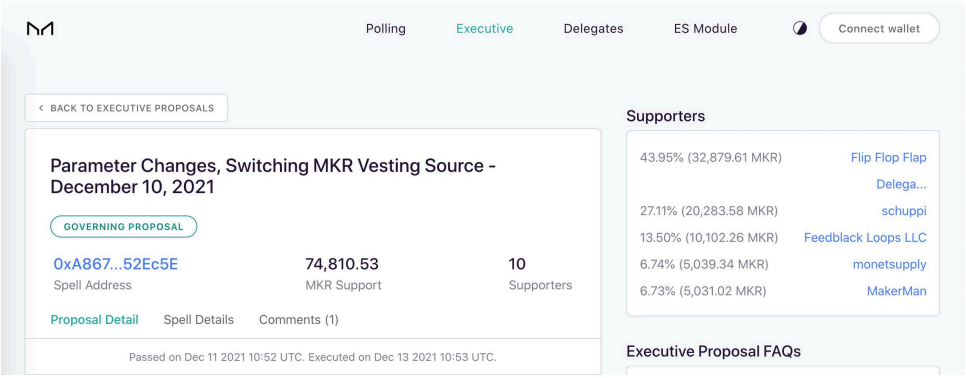


Figure 1.29: The governance platform of MakerDAO

1.4.3 DeFi protocols

While ICOs were the hot topic a few years ago, the word that generates the most traction today is DeFi for Decentralized Finance. Under the term Defi are grouped a set of applications that mimic classic financial instruments but in a distributed version on the blockchain. Financial products and services offered by DeFi applications range from interest-earning accounts to trading or landing or any financial instrument that can be implemented through a smart contract. In this section, we are going to review some notorious DeFi projects that answer several financial use cases.

Stable coin: MakerDAO's DAI token While crypto-assets popularity and usability are increasing every day, they are not yet to be used in every payment. Most governments do not accept crypto-asset as payment for taxes or legal fees. Very few real estate owners would accept crypto-assets as a payment for rent or sale, and most businesses do not accept crypto-assets. Even in the crypto-assets ecosystem, the reference value is still the fiat currencies such as the euro or the U.S. dollar. Thus there is a strong need for a crypto-asset whose value is backed by some fiat currency. These kinds of assets are called stable coins.

MakerDAO [64] that issues the DAI token is considered a DeFi application. Indeed, the issuance of the DAI token is fully distributed. The DAI is an ERC20 crypto-asset that attempts to maintain a stable 1:1 value with the U.S. dollar. But rather than having a real-world reserve in U.S. dollars, the DAI token maintains its value by locking other crypto-assets in contracts as collateral. Thus no centralized classical financial institution is involved in the issuance or backing of the DAI token. DAI tokens value is guaranteed by smart contracts called CDPs for Collateralized Debt positions. In order to obtain DAI tokens, users deposit their crypto-assets inside the CDP contract as collateral. CDPs can be thought of as secure vaults for storing the aforementioned collateral. To account for the volatility in the crypto collateral, i.e. changes in the exchange rate between the U.S. dollar and the collateral, DAI is often over-collateralized, meaning that the deposit amount required is typically higher than the value of DAI.

For example, users must spend $200 in Ether in order to receive $100 DAI, which is meant to account for the potential decrease in the value of Ether. As a result, if Ether depreciates by 25% compared to the U.S. dollar, the $100 in DAI would still be safely collateralized by $150 in Ether. The ratio between what is locked in the CDP and the amount of DAI tokens that were withdrawn is called the collateral ratio. Figure 1.30 presents the steps the user goes through to

issue some DAI tokens.

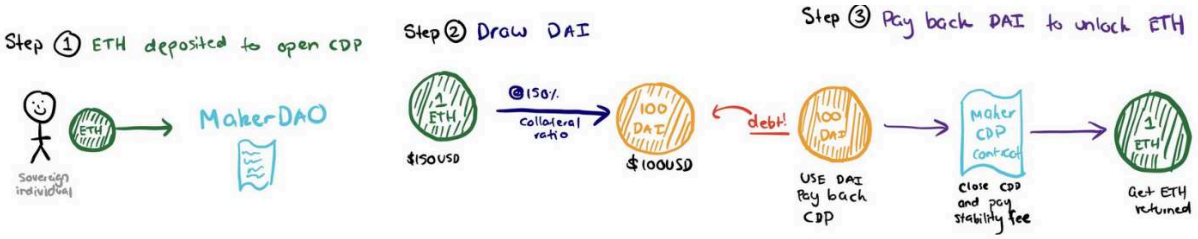


Figure 1.30: How to create a CDP with Ethers in order to issue DAI tokens

To guarantee that each DAI token is backed by enough collateral, the collateral ratio must stay above some predefined trigger, for example, 150%. If a CDP's collateral ratio falls under the above-mentioned trigger, the CDP can be liquidated by a third party.

Let's say a user has deposited one Ether that is at that time worth $150 inside a CDP. Let's assume he has issued 100DAI from this CDP. Its collateral ratio is 150% which is above the liquidation trigger. Now let's assume that the value of Ether falls by 10% to $135. The collateral ratio of the CDP is now equal to 135% which is below the liquidation ratio. At this point the CDP is open to liquidation, meaning that any third party can pay back your debt by buying back the collateral. In our example, a third party would have to spend 100DAI (the amount our user has issued from the CDP) to buy back the collateral. The third party has an incentive to do so because he would only spend $100 worth of DAI to buy the collateral that is still worth $135 of Ethers. Thus to prevent liquidation, the user should keep its collateral ratio above the 150% trigger. To do so he can either increase the amount of collateral in the CDP or return some DAI tokens.

Thanks to those mechanisms, makerDAO has been able to maintain the value of the DAI token very close to $1. At the time of writing, the total market capitalization of the DAI token was over $9 billions. They have been able to do so in a decentralized manner, where no centralized financial institution is involved. However, there is a very important software brick of the system that has not been discussed, this is, how is the exchange rate between the collateral and the U.S. dollar obtained? Indeed, as we have seen, for the system to function, it is necessary to input the exchange rate between the collateral and the U.S. dollar inside the smart contract. The exchange rate between the collateral and the U.S. dollar determines how much DAI can be issued, what is the collateral ratio and if a CDP is open or not for liquidation. To obtain the exchange rate between the collateral and the U.S. dollar, makerDAO makes use of what is called price oracles.

Price Oracle: Chainlink's price feeds In order to secure DeFi protocols such as the DAI token, it is necessary to know the exchange rate between some crypto-assets. As explained in the previous paragraph, the exchange rate between assets is used to calculate the collateral ratio which is a key parameter in the safety of DeFi protocols. We can not rely on end-users to input the prices because they could input wrong to their benefits. For example, with the DAI token, if the user was to input the exchange rate between Ethers and dollars, he could input a higher price than the real price to issue more DAI than what is collateral allows. Thus, to prevent such attacks, DeFi protocols make use of what we call price oracles. Price oracle is responsible for providing reliable price feeds to DeFi smart contracts. Before issuing DAI tokens, for example, the CDP contract requests the exchange rate from a price oracle. The returned price will be used to calculate how many DAI tokens the user will be able to issue.

One of the most famous distributed price oracle is Chainlink's off-chain reporting protocol [13].

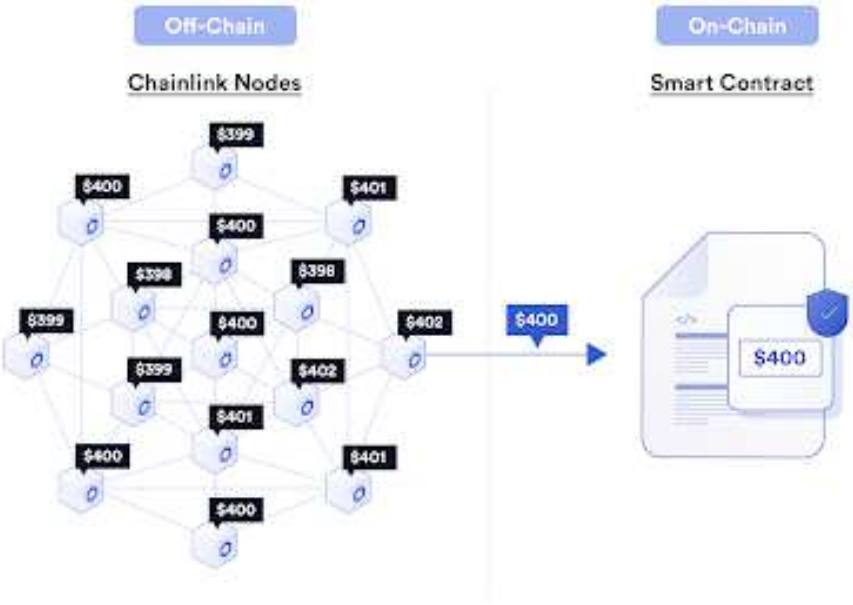


Figure 1.31: The chainlink's Off-Chain reporting protocol

Figure 1.31 presents chainlink's off-chain reporting protocol. This protocol relies on a network of nodes or oracle nodes that makes observations on real-world data. Chainlink's OCR can technically be used to observe any real-world variable such as the current temperature, the result of a football match, or the price of an asset. In this example, we will focus on prices. The oracle contract is a smart contract deployed to some blockchain. When a user, a smart contract, or a DeFi service needs to obtain the price of some asset, he makes a request to the

oracle contract. He also pays a little fee to the contract. The oracle contract keeps track of a list of oracle nodes and their public keys. When a price has been requested, a leader among the list of oracle nodes is elected. Each individual oracle node will gather the requested data from independent data sources. In the case of a price, the data source is in general a crypto-asset trading platform. Then, the oracle nodes sign their observations and send them back to the leader. Once the leader has received enough observations, he produces a report containing each individual observation and publishes it to the oracle contract. Signatures are then verified and the oracle contract returns the median of the observations to the entity that requested the price. The oracle nodes that have produced signed observations then receive their share of the fee paid by the requesting entity. Thanks to this protocol, chainlink's price feeds secure billions of dollars worth of assets in the DeFi ecosystem.

Other implementations of blockchain oracles exist, we will go into more detail in the following chapters. Yet it is to be noted that imputing off-chain data to a chain is a challenging problem because blockchains are self-contained environments that don't have natural access to the exterior. The verification process that is applied to transactions can not be applied to off-chain data and thus when off-chain data is needed, we must rely on distributed oracles or centralized third parties. When designing a DeFi service, it must be taken into consideration that oracles can fail or provide the wrong data. In February 2020, because of a human error, the XAG/USD chainlink price feed (silver versus U.S. dollars) registered silver at $1600 a once instead of $18. This resulted in $37000 worth of assets stolen by users.

Liquidity pools: Uniswap Another application of DeFi service that has gained in popularity during the past few years is liquidity pools. A liquidity pool is a smart contract within which several crypto-assets are locked in. They allow for several applications such as automated market-making (AMM), borrow-lend protocols, yield farming, synthetic assets, on-chain insurance, blockchain gaming, etc. The principle behind a liquidity pool is quite simple; users called liquidity providers add an equal value of two tokens in a pool to create a market. In exchange for providing their funds, they earn trading fees from the trades that happen in their pool, proportionally to their share of the total liquidity. The other users are the traders, they exchange one asset of the pool for another. Traders pay a little fee to the pool to exchange their assets. This fee is used to pay the interests of the liquidity providers.

Liquidity pools can be seen as an alternative to centralized trading platforms. In a central-

ized trading platform, the order book keeps track of all open positions and a matching engine is responsible for matching buy and sell positions. However, in the context of distributed applications, each interaction with an order book would imply paying gas fees to the network. When executing a trade on a liquidity pool, there is no need for a counterparty in the traditional sense. Instead, youre executing the trade against the liquidity pool. For the buyer to buy, there doesnt need to be a seller at that particular moment, only sufficient funds liquidity in the pool.

Figure 1.32 presents how liquidity providers and traders interact with the uniswap liquidity pool.

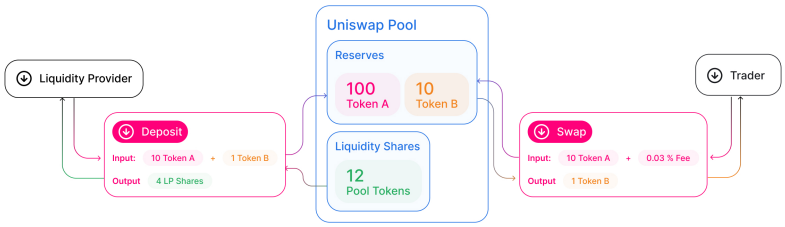


Figure 1.32: How the uniswap liquidity pool works

Uniswap is an automated liquidity protocol powered by a constant product formula and implemented in a system of smart contracts on the Ethereum blockchain. It obviates the need for trusted intermediaries, prioritizing decentralization, censorship resistance, and security.

Each Uniswap smart contract, or pair, manages a liquidity pool made up of reserves of two ERC-20 tokens.

Anyone can become a liquidity provider for a pool by depositing an equivalent value of each underlying token in return for pool tokens.

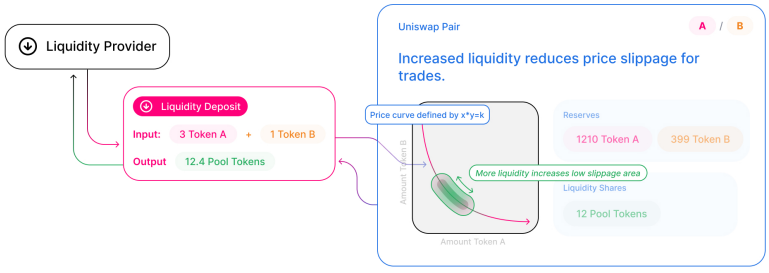


Figure 1.33: The uniswap constant product formula

Pairs act as automated market makers, standing ready to accept one token for the other as long as the constant product formula is preserved. This formula, most simply expressed as $x * y = k$, as presented in Figure1.33, states that trades must not change the product k of a pairs

reserve balances (x and y). Because k remains unchanged from the reference frame of a trade, it is often referred to as the invariant.

In practice, Uniswap applies a 0.30% fee to trades, which is added to reserves. As a result, each trade increases k. This functions as a payout to liquidity providers, which is realized when they burn their pool tokens to withdraw their portion of total reserves.

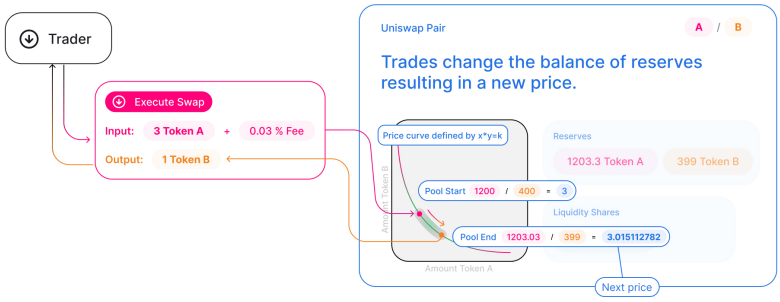


Figure 1.34: How prices evolve thanks to uniswap's constant product formula

Because the relative price of the two-pair assets can only be changed through trading, divergences between the pool price and external prices create arbitrage opportunities. This mechanism ensures that pool prices always trend toward the market-clearing price. Such a mechanism removes the need for a price oracle.

A lot of similar liquidity pools have appeared during the past few years, generating a lot of public traction. Companies like SushiSwap and ParaSwap have provided similar tools and contracts to provide liquidity to the DeFi ecosystem. While they allow for decentralized trading of crypto-assets, they do so only inside a single blockchain. For example, traders cannot exchange Bitcoin for Ethereum through a liquidity pool. They can only exchange ERC20 tokens whose values are pegged to Bitcoin and Ethereum. Liquidity pools don't allow for asset portability across blockchains.

1.5 Blockchain interoperability

1.5.1 Blockchain interoperability definitions

The AFUL, the french association for open source software users defines interoperability as "a characteristic of a product or system, whose interfaces are completely understood, to work with other products or systems, at present or in the future, in either implementation or access, without any restrictions". Thus, in the case of a blockchain system, blockchain interoperability

could be simply defined as the ability for blockchain systems to work with other systems, possibly other blockchain systems. However, in this definition, the term "work" is not clear and does not specifies which kind of interaction those systems will have.

While there is no clear consensus on the definition of blockchain interoperability in the academic literature, some papers give us a more precise definition.

Let us review some definitions found in the academic literature to have a more precise idea of what blockchain interoperability is:

In [9], Belchior et al. extend the common interoperability definition by adding the concept of cross-blockchain transactions. They define blockchain interoperability as "the ability of a source blockchain to change the state of a target blockchain (or vice versa), enabled by cross-chain or cross-blockchain transactions, spanning across a composition of homogeneous and heterogeneous blockchain systems".

In [90], Wang et al. define interoperability as "the ability to correctly conduct assets transferring and recording among a composition of homogeneous and heterogeneous blockchain systems, without compromising the legacy design philosophy of each blockchain system". We can see here that this definition focuses on the portability of assets across several blockchains.

Finally, the most formal definition that we found from Lafourcade et al. in [54] states that "a blockchain A that is interoperable with blockchain B accepts transactions because given the current state of A and Bs ledgers, the transaction does not violate As rules. Furthermore, if the rules for said transaction only imply conditions on As ledger, then the transaction does not require B to be valid, and as such does not make use of the interoperability. So an interoperable transaction on A must be dependent on the state of Bs ledger.

While those definitions share some concepts such as heterogeneous environments and transactions that span multiple blockchains we can see that there is no clear consensus on what blockchain interoperability really is. The first definition states that an interoperable blockchain should be able to change another chain's state while the second definition states that interoperability narrows down to the ability to transfer assets from one chain to the other. Finally, the third definition says that interoperability occurs when the validity of a transaction in a chain A is dependent on the state of a chain B.

This research area has recently gained a lot of interest, both from the industry and academics. In our opinion, blockchain interoperability is a very hot yet very immature topic, which explains why there is no consensus over the description of blockchain interoperability.

Thus, Rather than providing our own definition, we will introduce the subject by looking at the applications and use cases that require blockchain systems to "work" together.

1.5.2 Blockchain interoperability applications

Trading crypto-assets across chains The first and foremost application that requires some sort of blockchain interoperability is of course cross-chain asset exchanges. We will further discuss in more detail why this application is the one that focuses the most effort both from the academia and the industry, but what is to be noted for the moment is that in 2021, the crypto-currency trading industry has generated an average daily volume of about two trillions of U.S. dollars.

Exchanging crypto-assets across chains, for example, trading bitcoins for ethers is a very challenging problem. Blockchains are self-contained environments that have no natural access to the outside world. Each blockchain system has its own set of rules and agreement protocol for block creation and validation. This set of rules and the blockchain protocol itself can vary a lot between one chain and another. In most systems, nothing has been specially designed to access and verify outside world data, such as another blockchain's transactions. Thus it is difficult to set up an exchange protocol between chains because blockchains were not designed to read another chain's state. Moreover, actors in the system, such as traders, have conflicting interests and can potentially act maliciously toward each other. This issue is emphasized by the pseudonymous character of blockchain systems.

As of today, because of the above-mentioned problem, most crypto-asset exchanges take place on centralized platforms that act as trusted intermediaries. They hold the funds in custody and handle the transaction for a fee. However, for reasons that will be discussed later, we think that decentralized solutions are preferable in many ways to the centralized model.

Improving performance and scalability Recently, the most widely used smart contract platform, Ethereum, has seen its transaction fees greatly increase. As presented in Figure 1.35, while the average transaction costed less $0.1 in January 2020, it costs at the time of writing, an average of $30. This is mainly due to network congestion; the number of users and applications wanting to perform transactions is increasing while the amount of transactions per second the network is capable to achieve is constant. By simple market effect, the cost of executing a transaction becomes higher.

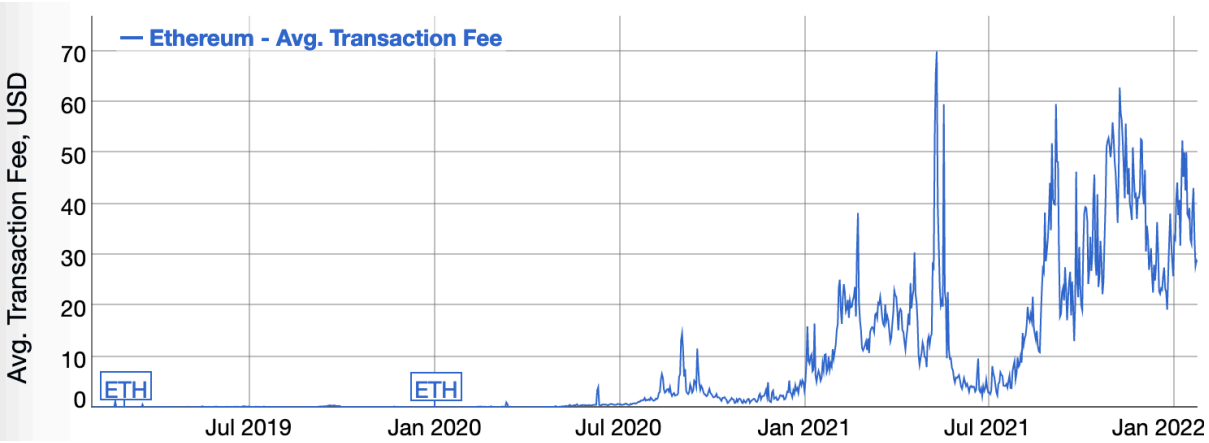


Figure 1.35: Ethereum average transaction fees in U.S. dollars. Three years to date.

We discussed earlier that solving this problem was one of the biggest challenges blockchain technologies have to face. A large part of both the industry and the academia are focusing their efforts on this topic. Several solutions are being explored. Switching to a different blockchain consensus protocol such as proof-of-stake is one of them. But in addition to this, some suggest that we could use what are called side-chains and/or relays.

In Layman's terms, a side-chain is a "sub-chain" that is plugged into a "main-chain" and within which assets are transferable. The side-chain is a blockchain with its own network of nodes, rules, and blockchain consensus protocol. Just like with a regular blockchain, validators produce blocks that contain transactions, etc. The idea is that, because the assets are transferable from one chain to the other, the side-chain can "off-load" the main-chain's network congestion [60]. The main chain does not need to be aware of each transaction of the side-chain. Rather it only needs to know the output of a batch of transactions. Thus, transactions on the side-chain can be batched together to take less space when they are synchronized with the main-chain. To transfer an asset from the main-chain to the side-chain, one must lock the asset inside a contract on the main chain. Then with the proof that the asset has been locked on the main-chain, the user can request a smart contract on the side-chain to issue him, an equal amount of asset in the currency of the side-chain. In reverse, the asset can be transferred from the side-chain to the main-chain in a similar process. The user presents a proof to the main-chain that he has locked or burned the tokens of the side-chain, and gets back its locked assets of the main chain. The concept is often described in the literature as relaying and blockchain relays. This will be discussed in later sections, but synthetically a relay is a contract on chain B that receives a block from chain A submitted by individual processes called relayers. The relay contract on chain B implements the verification algorithm of blockchain A. Thus when submitted a transaction

of chain A, the relay on chain B can check if this transaction is actually a valid transaction of chain A or not. This cross-chain verification process allows the other blockchain to issue or lock assets and thus keeping the monetary volume constant.

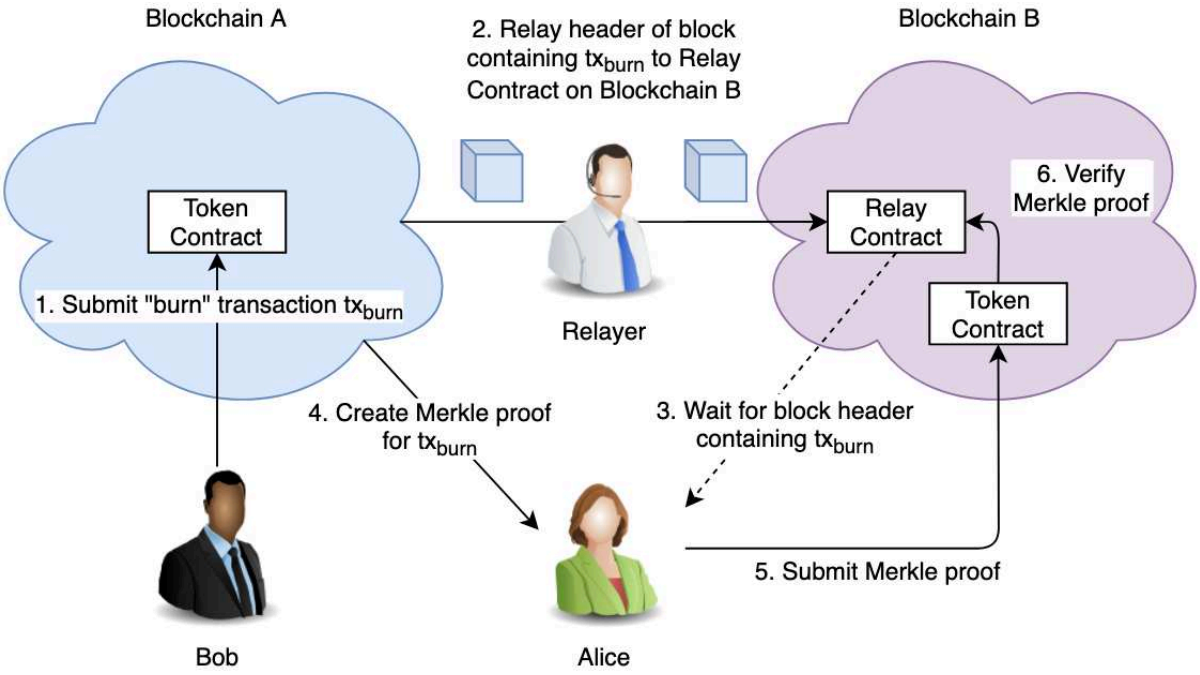


Figure 1.36: Cross-blockchain Token Transfer with Blockchain Relay [34].

Because assets can be transferred from the main-chain to the side-chain and vice versa, each chain now only has to care about transactions and blocks that happen on its own chain. Of course, both the side-chain and the main-chain need to record the blocks of the other chain, but only the block header tanks to Simple Payment Verification or SPV (see Section 1.1.4. And because block headers are much smaller than full blocks (80 bytes versus 10 MB for Bitcoin), the total transaction throughput of the whole system that consists of the main and the side chains is almost equal to the original transaction throughput of the main chain plus the transaction throughput of the side-chain. The process is presented in Figure 1.36.

The term side-chain is subject to debate [89]. Indeed the naming involves that there is a main chain and a side chain. However, we could consider that both chains are side-chains of the other. Thus instead of saying that B is a side chain of A we prefer to say that chain A has a relay of chain B.

Inputting data from the outside world Another application of blockchain interoperability relies upon what is called cross-chain oracles. A cross-chain oracle is an abstraction that allows to input outside world data, possibly from another chain, to a blockchain. We saw in the previous

chapter that some blockchain applications required crypto-currency market prices to be inputted into a smart contract. But this application can be extended to other types of data. We could for example think of an identity oracle. This identity oracle hosted on blockchain A would be able to provide identity proofs to another blockchain. Thus some smart contract on blockchain B could be programmed to allow certain actions from a user, only if it has received identity proof from the oracle on blockchain A. To guarantee the veracity of the data that is inputed, a sort of agreement or averaging protocol is applied to observed data. The principle is to collect data from as many sources as possible to ensure that it reflects the real value "in average". The chainlink's Off-chain Reporting [13] specifies such a system.

1.5.3 Blockchain interoperability use cases

From our observations, we think that there are three main applications to blockchain interoperability. First, we can swap assets across chains by synchronizing transactions. Then we can improve scalability with side-chains enabled by relays. By making assets portable from one chain to the other, we can increase the total network throughput. Finally, we can input data from off-chain sources with blockchain oracles.

To put these notions in context we will review in this section some projects that make use of blockchain interoperability.

Polkadot's multi-chain network In 2016, Gavin Wood et al., one of Ethereum's co-founders presented Polkadot in [94]. Polkadot is a framework that aims at connecting blockchain networks. The idea is similar to the idea of side-chains; a main-chain called the relay-chain validates a number of sub-chains called the parachains, for parallelized-chains. Due to their parallel nature, they can parallelize transaction processing and achieve scalability of the Polkadot system.

The relay-chain prime role is to validate the parachains. Thus it supports limited functionalities, for example, it does not implement smart contracts.

Each parachain appears to the relay-chain as a globally-coherent application-specific dynamic data structure. Individually, the parachains can be seen and used as an independent chain.

Polkadot places no constraints over what parachains can do besides that they must be able to generate a proof that can be validated by the validators assigned to the parachain. This

framework called "Substrate" guarantees cross-language support with WebAssembly, a light client, and off-chain workers, allowing for integration with other technologies. Polkadot can even work with public networks such as Ethereum thanks to bridges.

Parachains that were built following this framework can communicate with each other via the Cross-chain Message Passing Protocol (XCMP), a queuing communication mechanism based on a Merkle tree.

The whole protocol is quite complex but to put it up in simple terms; the relay-chain is a Proof-of-Stake blockchain with a set of validators that have staked DOT tokens (the currency of Polkadot). Each parachain is registered to a slot on the relay-chain. The number of available slots is limited to around 100 [75]. The parachains function independently and the relay-chain does not need to know every transaction that has happened on every parachain. Processes from the parachains called collators are responsible for regularly submitting the parachain state transition proof to the validators. Validators will then build relay-chain blocks that are made out of parachain state transition proofs.

With this framework, Polkadot claims to achieve scalability and cross-chain communication with heterogeneous blockchains. They also say that each parachain is secured by the relay-chain and thus achieves the same levels of security.

Chainlink's off-chain reporting Chainlink Off-chain reporting protocol is an oracle protocol for blockchains. While its main application is to provide price feeds to smart contracts, it can be used for any type of data. It can be used to collect exchange rates between crypto-assets but also meteorological data, the outcome of a sporting event, etc.

In [13], Breidenbach et al. describe "Chainlink off-chain reporting protocol, a new, more scalable, version of the protocol driving Chainlinks data feeds". Chainlink already had a protocol for providing data feeds to smart contracts. However, the old protocol required several transactions on-chain, which became more and more constraining with the raise of transaction fees on the Ethereum network. Moreover, they aim at designing a framework that is more blockchain agnostic, meaning that it could be plugged into any blockchain network.

The system consists of n nodes called oracles that communicate through a network and are identified by their network endpoints and authenticated by their cryptographic keys. The set of oracles P is fixed and their public key is registered in the oracle smart contract C. The goal of the protocol is to repeatedly submit reports to C containing signed observations of data from

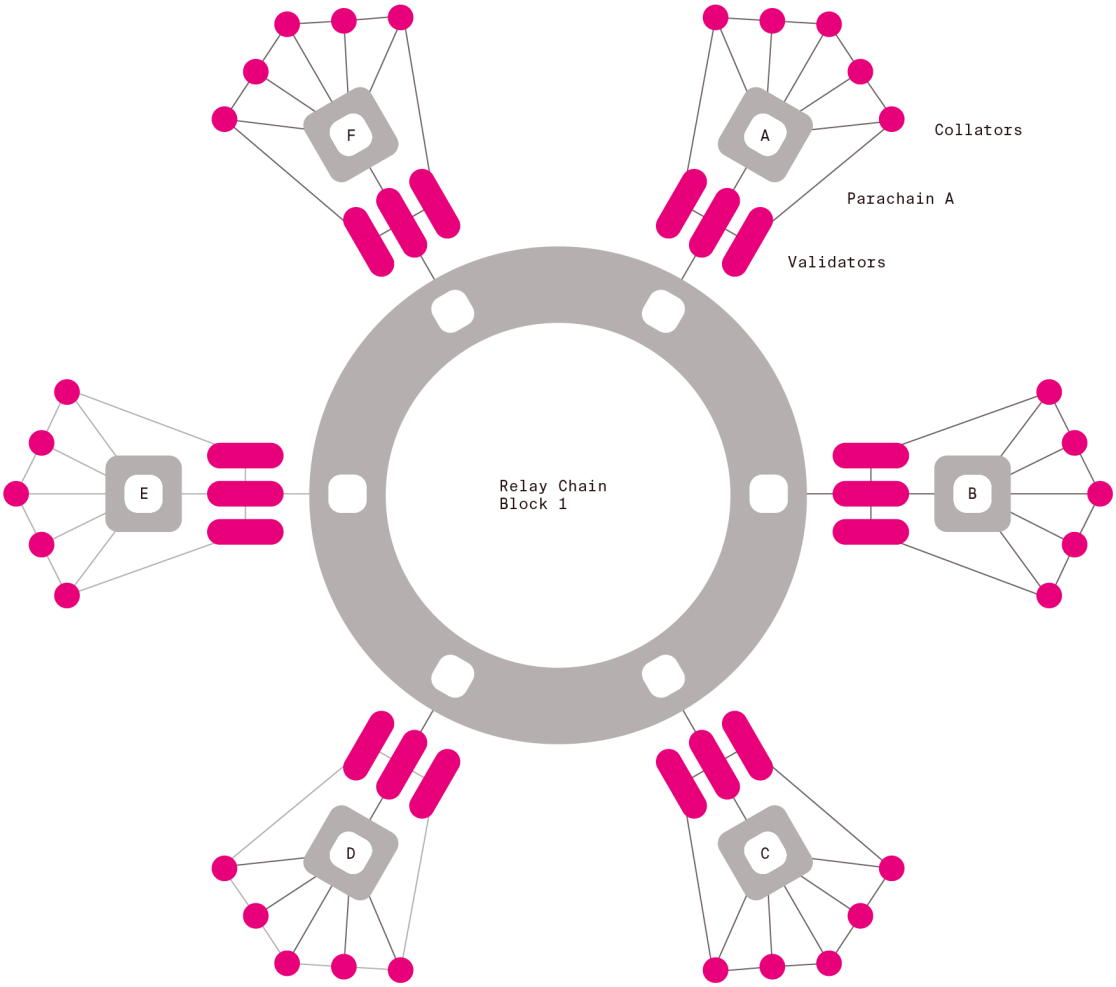


Figure 1.37: Visual representation of the polkadot network.

the oracles. The time is divided into epochs, and at each epoch, a leader is elected to produce the report. Oracles are connected to a data source such as a market's API and they submit their signed observations to the current leader. The leader is responsible for producing the report containing the said signed observations and submitting it to the oracle contract. The protocol is derived from PBFT consensus such as [21]. However, it does not ensure consensus and relies on C for resolving ambiguities that may occur due to transitions across epochs.

The timing model is aligned with the partially synchronous model as in [30], meaning that the network is asynchronous until some time called Global Stabilization Time or GST, after which the network behaves synchronously. It is to be noted that the protocol does not guarantee liveness outside synchronous periods.

1.6 Conclusions

In this chapter we briefly introduced the concept of blockchain by reviewing each element that makes up a blockchain system. We have seen that a blockchain is a digital ledger of transactions arranged in blocks, each block pointing to the previous one, and thus forming a chain of blocks. This digital ledger is replicated over an unreliable network of nodes whom identify and authenticate each other thanks to public key cryptography. These nodes exchange transaction messages that will eventually be included in a block that will be appended to the ledger. To be included in a block, transaction messages must satisfy a set of rules specific to each blockchain system. We then discussed the actual execution, by reviewing some blockchain protocols such as proof-of-work and chain based proof-of-stake. The main goal of those blockchain protocols is for the nodes to decide what the next block will be. To do so, chain-based protocol run a pseudo lottery where a process is designated to produce the next block. The difference between proof-of-stake and proof-of-work is that the chances of being elected as the block producer depends on stake allocated respectively on computing power. While they work in practice chain-based protocol do not guarantee finality. Indeed, we have seen that more than one process can be elected as block producer at the same time resulting in the network being divided in several partition that do not agree on the state of the chain. Chain-based protocols overcome this issue by awaiting for one of the state of the chain to be longer or denser than the other(s). But some other protocols called Byzantine Fault Tolerant style blockchain protocol achieve instant finality. They address a problem well known is the literature and introduced in this Chapter; the byzantine consensus problem. We introduced an informal framework for blockchain evaluation called the blockchain trillema. It states that blockchain systems tends to favor two out of the following three properties: scalability, security and decentralization. We discussed smart contracts, which are programs executed on the blockchain that open up for a wide range of applications. From gaming with CryptoKitties to financial applications such as decentralized trading of assets. This last use case, trading assets inside a blockchain and across blockchain is a topic that attracts a lots of interest both from the industry and the academia. As of today, the use case that generates the most volume in the crypo-asset ecosystem is crypto-asset trading. But in order to exchange assets across chain, some level of interoperability, or cross-chain communication, is required. Thus we briefly introduced the main topic of this document, blockchain interoperability. According to the most formal definition we could find

in the literature by Lafourcade et al. in [54], blockchain interoperability is only possible by having a "2 in 1" system where a child blockchain is somehow contained in a parent chain. In our opinion, this definition is too narrow, and exclude some interoperable systems that can synchronize events and transaction across chains, without a "2 in 1" system. If one can reliably read off-chain data from within the chain, then he can synchronize transactions between several chains without a "2 in 1" system. In the next chapter, we are going to introduce blockchain oracles, which are systems adressing this problem.

Chapter 2 Decentralized permission-less price Oracles

2.1 Abstract

Blockchain oracles are systems that connect blockchains with the outside world by interfacing with external data providers. They provide decentralized applications with the external information needed for smart contract execution. In this paper, we focus on decentralized price oracles, which are distributed systems that provide exchange rates of digital assets to smart contracts. They are the cornerstone of the safety of some decentralized finance applications such as stable coins or lending protocols. They consist of a network of nodes called oracles that gather information from off-chain sources such as an exchange market's API and feed it to smart contracts. Among the desired properties of a price oracle system are low latency, availability, and low operating cost. Moreover, they should overcome constraints such as having diverse data sources which is known as the freeloading problem or Byzantine failures.

In this chapter, we define the distributed price oracle problem and present PoWacle, the first asynchronous decentralized oracle protocol that copes with Byzantine behavior.

2.2 Introduction

Decentralized finance (DeFi) is a term that emerged during the past few years to describe financial instruments that do not rely on centralized intermediaries such as brokerages exchanges or banks. To implement those instruments, DeFi protocols make use of smart contracts hosted on blockchain systems. Those smart contracts are programs that implement the logic of classical financial instruments. A wide range of applications is already in production, from interest-earning saving accounts to lending protocols to synthetic assets or trading platforms, etc. This industry is quickly gaining in popularity both in terms of the number of users and market capitalization.

In order to function, a lot of those DeFi protocols make use of what is called blockchain oracles and more specifically blockchain price oracles. A price oracle is a system that provides

exchange rates or prices to DeFi protocol's smart contracts. They gather data from off-chain sources, such as an API, and feed it to a smart contract on-chain. For example, a popular DeFi application consists in issuing a number of tokens to a user in exchange for collateral that will be locked in a smart contract, until the user pays back his debt. Obviously, for the process to be fair, it is necessary to know the current exchange rate between the token issued and the token locked as collateral. This is where prices oracles come into the picture.

Price oracles can be split into two categories, centralized and decentralized. A centralized oracle relies on the observations of a single trusted entity while decentralized oracles gather information from several sources. In this paper, we focus on decentralized ones, as we consider that DeFi protocols should not rely on a single trusted entity.

2.3 Related works

Although decentralized price oracles have a central role in designing DeFi applications, there is very little academic literature that addresses fault-tolerant decentralized oracles. To the best of our knowledge, the only academic work addressing this problem is [13]. However, several non-academic reports propose ad hoc solutions practical solutions. In the line of non-academic work, the most interesting contributions are the band protocol [8] and DOS network [68].

2.3.1 Band protocol

Band protocol [8] is a public blockchain network that allows users to query off-chain APIs. It is built on top of the Cosmos-SDK and uses the Tendermint consensus engine to reach instant finality. The oracles in the band protocol are selected pseudo-randomly to produce observations that they gathered from off-chain data sources. Much like in a proof-of-stake-based chain, they have tokens staked on the chain and their chances of being elected to produce an observation are proportional to their share of the total stake. When an oracle produces an observation, this observation is published to the bandchain, and a proof is generated. This proof can be later used by smart contracts on other blockchains to verify the existence of the data as well as to decode and retrieve the result stored. This process requires a bridge contract on the targeted blockchain to interpret the proof. While Band protocol's approach is interesting, we think that it lakes interoperability concerns. Indeed a bridge contract has to be implemented for each new

integration. Our proposal is integrated by design as it leverages the target blockchain's keys.

2.3.2 DOS network

Another non-academic work is DOS network [68]. The DOS network leverages verifiable random functions, distributed key generation, and threshold signature scheme to implement their Decentralized Oracle Service. The system is made of a network of oracle nodes connected to off-chain data sources. The time is divided into rounds and for each round, a group of nodes is randomly selected to provide the observations. Each member of the group is given a share of a distributed private key. Members of the group exchange messages containing their signed observations until one of them has received enough signatures to generate the group signature. This node will be responsible for publishing the report containing the group's observations and group signature. The smart contract will then verify the signature and execute the payout. The idea proposed by this scheme is interesting however it has a major drawback. This approach seems very interesting in terms of properties it guarantees as well as in terms of performance. But in our opinion, it suffers a major drawback. The probability that two or more members of the group are able to construct the group signature at the same time is high. This will result in several nodes publishing the same reports simultaneously. Although this can be resolved on-chain by a proxy that only accepts the first response, the cost of the transaction is permanently lost for the following reporters. Our approach based on proof-of-work can be used in asynchronous settings it allows us to better sample the probability of finding a valid report.

2.3.3 Chainlink's Off-chain reporting protocol

In [13], Chainlink presents the "Off-chain Reporting Protocol", an oracle system designed, among other goals, to minimize transaction fees. Transaction fees for some blockchains have become quite prohibitive, motivating the need for such systems. The system consists of n Oracles that exchange messages through off-chain communication channels. The Oracles and their respective public keys are listed in a smart contract C. Time is divided into epochs, and a leader taken from the list of oracles is associated with each epoch. Epochs are mapped to leaders through a deterministic function. A simple modulus that can be calculated by anyone in the network. The Oracles make observations (such as price observations), sign them with their private keys, and submit them to the leader. When he received a sufficient amount of

observations, the leader builds a report that lists them all, as well as the signatures, and submits it to a transmission protocol. Finally the transmission protocol hands out the report to the smart contract C. A new epoch associated with a new leader starts whenever the oracles think that the current leader does not perform correctly. While this protocol shows good resilience and low transaction fees, they assume a partially synchronous model. Formally, they assume that clocks in the system are not synchronized until a point in time called global stabilization time (GST). Afterward, all correct nodes behave synchronously. Outside those synchronous periods, the liveness of the protocol is not ensured. We think that by using proof-of-work for leader election, we could ensure similar properties in a fully asynchronous timing model.

In this chapter we follow the line of research opened by [13]. But differently from their approach, we consider an asynchronous communication model.

Our contribution In this paper, we formalize the distributed price oracle problem in the context of decentralized finance applications. Furthermore, we propose a protocol and prove that it verifies the specification of the problem in asynchronous communication environments prone to Byzantine failures. The protocol combines a gossip module with a light proof-of-work module and incentives oracles via a simple reputation mechanism to have a correct behavior.

2.4 Model

Considering the oracle network, we consider a similar model as the one used by the chainlink's off-chain reporting protocol [13]. The main difference is the communication model which is partially synchronous while in our system it is asynchronous.

2.4.1 System model

Distributed ledger We use the terms distributed ledger and blockchain as synonyms and adapt the distributed ledger model based off [22]. The blockchain BC is a concurrent object that stores a totally ordered sequence of records. The blockchain BC supports two operations, $BC.append(b)$ which appends a new block b to the chain and, $BC.get()$ that returns the whole chain. A block is a triple $b = (\tau, p, v)$, where p is the identifier of the process that created block b, v is the data of the record drawn from an alphabet Σ, and τ is a unique record identifier from a set T (e.g., the cryptographic hash of (p, v)). The blockchain is implemented by a set

of servers that collaborate running a distributed algorithm. The blockchain is used by a set of clients that access it by invoking append and get operations, which are translated into request and response messages exchanged with the servers. An execution is a sequence of invocation and return events, starting with an invocation event. An operation π is complete in an execution ξ, if both the invocation and matching return of π appear in ξ. We say that an operation π_1 precedes an operation π_2 in an execution ξ if the return event of π_1 appears before the invocation event of π_2 in ξ; otherwise the two operations are concurrent. By default any client can append blocks or access the state of the blockchain with get. However, if convenient, we may assume that the set of clients that can issue (append and get) operations is restricted. For instance, we assume that only the creator $b.p$ of a block b can append the record in a blockchain BC, or restrict append operations to a predefined set of clients.

Communication network The network is composed of a set of participants $P = \{p_1, ..., p_n\}$ that are referred to as nodes. In order to communicate, participants are linked to each other over a network. A network is a couple (P, E) where P is the set of participants in the system, and $E \subseteq P \times P$ is the set of bidirectional communication channels (edges) between participants. Let p_i, p_j be two participants, if $(p_i, p_j) \in E$, there exists a communication channel between them; we say that p_i is a neighbour of p_j and vice-versa. If $(p_i, p_j) \notin E$ there is no direct communication channel between p_i and p_j, they are not neighbours of each other. We say that there is a path between two participants if they are neighbours, or if there exists a finite sequence of participants $p_1, p_2, ..., p_n$ where n is the length of the path, such that the following conditions hold:

- p_i, called the source;
- $p_n = p_j$, called the recipient; and
- $\forall k \in \{1, ..., n-1\}, (p_k, p_{k+1}) \in E$

When between any two participants there exists a path, we say that the network is connected.Participants can send messages to their neighbours through this network.

Failures We consider that any $f < n/3$ nodes may exhibit Byzantine faults, which means that they may behave arbitrarily and as if controlled by an imaginary adversary. All non-faulty nodes are called honest or correct. We consider that these faults can occur adaptively meaning an adversary can choose and control the faulty nodes on the fly. It is expected that the protocol operates usually with $n \geq 3f + 1$ since this gives optimal resilience. The failure assumption

also covers network failures and crashes. This means that no more than f nodes can become isolated from the network for the protocol to function correctly.

Cryptographic assumptions For the cryptographic assumptions, we use a model similar to [3]. In blockchain systems, the paradigm of asymmetric cryptography is often used; intuitively, each participant has a pair of keys:

- A private key: which is a string of alphanumerical characters that is known only by its owner.
- A public key: which is another string of alphanumerical characters, derived from the private key, that can be shared to other participants.

The pair private/public key has to satisfy some mathematical properties depending on the cryptosystem used. All participants use the same cryptosystem. To compute the pair of keys, one may use a one-way function. A one-way function $f : X \to Y$ is a function such that: (i) f (x) is easy to compute (i.e., possible in polynomial time), and (ii) inverting f , i.e., knowing $y = f(x)$, and x is difficult to compute (i.e., is not possible in polynomial time). It is an open question to prove whether such functions exist or not, in fact, their existence will be a proof of $P = NP$. However, there exist some candidates for such functions, e.g., prime factoring, discrete logarithm, etc. There is, in the state of the art, no proof that these candidate functions cannot be inverted in polynomial time on a classical computer. Using these keys (private/public), participants can have unique digital identities and can sign messages. We assume an idealized public-key infrastructure: each process is associated with its unique public/private key pair that is used to sign messages and verify the signatures of other nodes. A message m sent by a process p_i that is properly signed with the private key of p_i is said to be properly authenticated. We denote by m_{σ_i} a message m signed with the private key of a process p_i. We denote by $sign_i(m)$ the function that signs the message m with the private key of process p_i producing the signed message m_{σ_i}. Similarly $verify_i(m_{\sigma_i})$ verifies the signature of signed message m_{σ_i} with the public key of process p_i.

Smart contracts In this document we are going to use the concept of smart contract. Intuitively, they can be taught of as programs that are executed on the blockchain. But here we model them more formally as state transition systems like in [42]. The representation of a contract C in the state transition system is a quintuple of elements $M = (Q, \Sigma, \delta, s_0, F)$ where;

- Q is the finite set of all possible states of smart contract C;

- Σ is the set of all input event on C
- δ is the set of transition-function of C, $\delta : Q \times \Sigma \to Q$
- F is the final state of C, $F \in Q$

If the blockchain state is noted by γ, for any successful transaction executed by C, the blockchain state will be updated into γ'. The new state γ', might impact many user accounts, or other smart contracts, that might have their impact on the empirical data on the blockchain.

Oracle network. The system consists in a set of n Oracles $P = \{p_1, ..., p_n\}$ that are referred to as nodes. Each oracle node p_i makes time-varying observations over the price of an asset pair. The set of oracles is determined by an oracle smart contract C that records the public keys of the nodes. The owner of the smart contract has administrative powers which allow him to update the list of oracles. As we are working with time-varying quantities, there is no proper way to evaluate if an observation is correct or not. Thus the protocol only guarantees that the report contains a sufficient number of observations signed by honest oracle nodes.

Oracle smart contract and report. The goal of the system is to produce reports containing a sufficient number of signed observations from the oracle nodes when a client requests them. Differently from Breidenbach et. al in [13], the reports are submitted to the oracle smart contract C by some node during a proof-of-work inspired cryptographic race. The smart contract C corresponds to a single asset pair (e.g. BTC/USD). When submitted a report, the oracle smart contract C verifies the signatures of each observation as well as the proof-of-work. If they are valid, the oracle smart contract updates its variable $lastPrice$ to the median of observation values contained in the report. Using the median value among more than $2f$ observations guarantees that the reported value is plausible in the sense that malicious nodes cannot move the value outside the range of observations produced by honest nodes. The value $lastPrice$ can then be consumed by the requesting client or by any other user. The requesting client pays a fee for each new report he requests, which will be distributed equally among the observant nodes.

2.5 Decentralized price Oracle problem

In this section, we will define the decentralized oracle problem and review major threats and constraints that must be taken into account when designing a decentralized oracle system. First, we will define the oracle problem which sums up why reliable oracles are so crucial, and

then we are going to review the individual threats that make the oracle problem so difficult to resolve.

The blockchain Oracle problem is well known in the ecosystem and the grey literature. It is also described in [20] by Cardelli et al. The oracle problem describes the ambivalence between blockchain systems that are supposed to be immutable through decentralization and oracles that, by definition, input outside world data that cannot be verified by the blockchain itself. A blockchain is a self-contained environment with its own validation rules and consensus mechanism. This isolation is what makes blockchain transactions safe and immutable. However, when data is inputted from off-chain data sources, the said data cannot be verified by the blockchain itself. A piece of software, in this case, an oracle, is required to guarantee the veracity of the inputted data. As the safety of a system is limited by its weakest element, in a system where the execution of a smart contract depends on the data provided by an oracle, the oracle may be a single point of failure. This oracle problem has already led to several exploits. In 2019, an oracle reported a price a thousand times higher than the actual price [24], which led to a one billion U.S. dollars loss. Funds have then been recovered but it shows how crucial is it to have reliable oracles. To be best of our knowledge there is no formal definition of the blockchain price oracle problem.

The price oracle smart contract can be seen as a particular single-writer multi-reader shared register. The variable of this particular shared register, $lastPrice$, can be read by any client of the system. This variable can be modified only by the smart contract C, and the modification is triggered each time clients invoke $requestNewPrice()$. We propose below a definition of the blockchain price oracle problem in terms of liveness, integrity, and uniformity.

Definition 1 (Decentralized blockchain price oracle) A decentralized blockchain price oracle should satisfy the following properties :

- **Δ-Liveness**: There exist a $\Delta > 0$ such that if a client invokes a price request to the smart contract C at time $t > 0$ then a corresponding report r will be retrieved from C within $t + \Delta$ time.
- **Observation integrity**: If a report with v is declared final by C, then v is the observation of a correct oracle or in the range of the observations of the two correct oracles in the system.
- **Uniformity**: If two clients, c_1 and c_2 read the oracle $lastPrice$ at time $t > 0$ then the same price report will be retrieved by both of them.

The price oracle smart contract can be seen as a shared register. The variable of this particular shared register, $lastPrice$, can be read by any client of the system. This variable can be modified only by the smart contract and the modification is triggered each time clients invoke price requests.

Designing distributed price oracles is not an easy task. In the following, we discuss several difficulties.

Freeloading attacks The freeloading attack, also known as mirroring is formally described in [66]. It refers to the technique employed by malicious oracles, that instead of obtaining data from their data source, replicate the data fetched by another oracle. Since oracle systems are often comprised of a reputation system, a "lazy" oracle could simply copy the values provided by a trusted oracle instead of making the effort of fetching the data itself. By doing so, the lazy oracle maximizes its chances of receiving its payout and avoids the cost of requesting data from sources that might charge per-query fees. As explained in [76], this freeloading attack weakens the security of the system by reducing the diversity of data sources and also disincentivizes oracles from responding quickly: Responding slowly and freeloading is a cheaper strategy.

Majority attacks Much like in blockchain systems, oracle systems may be threatened by majority attacks. If an entity controls a majority of oracles in a network, it may be able to manipulate the data to provide prices that diverge from the real observable data.

Price slippage Price slippage refers to the fact that the price returned by an oracle may differ from the actual price. This may be intentional or unintentional but the result is the same. Price slippage may be the consequence of delays generated by the transaction validation time. Indeed real-world prices evolve continuously while the events on a blockchain are discrete. The state of the chain only changes when a new block is added.

Data source manipulation The source which the data is gathered from might also be a vector of attack. Indeed, if the data source or the communication channel between the oracle and the data source can be altered, the resulting observation will ultimately be altered too.

2.6 PoWacle Protocol Overview

In this section, we propose first a high-level overview of the PoWacle protocol and then propose the protocol pseudo-code.

The goal of the protocol is to publish reports containing price observations from the oracles when a client makes a request. The oracle nodes are connected to external data sources such as a market's APIs where they find their price data. When a client requests a price, the oracles will exchange messages containing hash-signed observations of the asset's price. It is very important to note that the content of the observations exchanged is not readable by the other oracles, as they are hashed observations. This is to avoid the freeloading problem. Each oracle node listens for incoming hash-signed observation messages and much like in a proof-of-work-based blockchain builds a pool of messages in their local memory. The difference with a proof-of-work-based blockchain is that instead of transactions, the pool contains hash-signed price observations, and instead of producing a block of transactions, the goal is to produce a report containing the said observations. To select the oracle that will be responsible for proposing and publishing the report, a proof-of-work protocol is applied. Once they have received hash-signed observations from a sufficient number of nodes, i.e. more than $2f + 1$, the oracles start the report mining process. They try to build a report proposal whose hash value is inferior to some target difficulty number. When a node finds a report proposal whose hash value is inferior to the target difficulty number, he broadcasts the report proposal to the network. On receiving the report proposal, the other nodes return the readable pre-image of their hash-signed observations. For each received observation, the proposer verifies that the observation matches its hash. Once he has collected at least $2f + 1$ clear observations, he submits the report proposal to the oracle smart contract along with the proof-of-work. The oracle smart contract verifies the proof-of-work, the match between observations and their hashes, and the signatures. Once all verifications are done, the smart contract calculates the median of the received observations and updates the price of the asset. The price can then be consumed by the client. The smart contract calculates the payout and updates the reputation of the oracle nodes.

The incentive mechanism is designed such that each oracle that produced an observation will be paid equally. Thereby, the report proposer, the one that won the proof-of-work and published the report to the chain, has no more incentives than the other oracles. This should help to limit the computing power that the oracles will put in the network, and thus the operating

cost of the system. In the meantime, the oracles are still encouraged to find valid reports regularly, as this is the way they get paid. The goal is to have an incentive equilibrium between producing reports regularly and having a low operating cost in terms of computing power.

Let us unfold the main steps of the protocol:

1. The client requests a new price to the smart contract C and pays a fee.
2. The oracles pick up the request and gather price data from their data sources. They create their observations, hash them, and sign them. They broadcast it to the oracle network through gossiping.
3. On receiving the hashed signed observations, the nodes verify the signatures. If they are correct, they add the observation to their local memory. Once they have received a sufficient number of hashed signed observations, i.e., above the $2f + 1$, the oracles sequentially rearrange their report and a nonce until they find a report proposal whose hash value is below a target difficulty number.
4. The first oracle to find a valid report proposal broadcast it to the oracle network via gossiping.
5. On receiving a valid report proposal, the oracles whose observations were contained in the report return the readable pre-images of their hashed signed observations to the candidate leader.
6. Once he has collected enough pre-images, the proposer verifies that they match their hashes. If they do match, he submits the report to the smart contract C.
7. On receiving a report proposal, the smart contract C verifies the signatures of each individual observation as well as the matching between clear observation and their hash. If everything matches, the smart contract verifies the proof of work. It also verifies that there are at least $2f + 1$ clear observations in the report. Once all checks are passed, the smart contract updates the price, making it available to the client. The report becomes final.
8. The smart contract C calculates the payouts and updates the reputations.

2.7 Protocol detailed description

We provide a detailed description of the protocols using an event-based notation (e.g. [19] Chap. 1).

The protocol consists of an oracle smart contract hosted on a blockchain presented in Algorithm 1 and an oracle network. The nodes in the oracle network react to events triggered by the oracle smart contract C. The oracle nodes can at any time read the state and the variables of C. Protocol instances that run on the same oracle node (instances of Algorithm 2,3 and 4) also communicates trough events. In the following, Algorithm 1 is the oracle smart contract C, Algorithm 2, executed by every node is a daemon that reacts to events triggered by C. Algorithm 3 is the observation gossip protocol and Algorithm 4 is the report mining protocol.

2.7.1 The Smart Contract (C).

The protocol Algorithm 1 is orchestrated by an oracle smart contract C hosted on a blockchain. Each oracle smart contract represents a price feed for a single pair of assets, for example, USD/BTC for bitcoin versus U.S. dollars. The oracle smart contract C consists of four primitives; identity, proof-of-work, incentive, and reputation. We will review individually each primitive in this section.

Identity. The oracle smart contract is responsible for storing the identity of the oracles nodes. It maintains a list of oracle nodes and their public keys. The set of oracles is managed by an owner with administrative power. How the administrator curates the list of authorized oracles is behind the scope of this document. The oracle list is used to verify signatures.

Proof-of-work. The proof-of-work primitive is responsible for verifying the proof-of-work that is submitted along with each report. It verifies the match between hash-signed and clear observation, verifies that there are more than $2f$ clear observations, requests the identity primitive to verify the signatures, and finally verifies the proof-of-work. To do so, it checks that the submitted report header hash value is below the target difficulty number. If so, the report is final and the contract updates the last price value, making it available to the client.

The proof-of-work primitive is also responsible for adjusting the difficulty target number. To do so it uses a system similar to Bitcoin's difficulty adjustment [67]. For each new request, the smart contract C records the time difference between the moment the request was made by the client, and the moment the corresponding report was submitted. Then for every new report, the contract calculates the average report generation time over the last hundred reports. It then adjusts the target difficulty number to have a stable average report generation time. To do so, it

multiplies the current difficulty target number by a ratio between observed and desired report generation time. Target report generation time is specified by the owner with administrative power. What value should be chosen is out of the scope of this document and would require further analysis.

Incentive. The incentive primitive calculates and broadcasts the payouts to the oracles. For each final report, the contract C pays an equal share of the total fee to each oracle that produced an observation that was included in the report.

As introduced earlier, the incentive system must be designed to create an incentive equilibrium between producing reports regularly and having a low operating cost in terms of computing power. Thus, the payout received by the report publisher will be equal to the payouts received by the other oracles that produced an observation contained in the report. The only difference is that the report producer gets his transaction fees refunded.

In order to reduce transaction fees, the payout is not automatically transferred at each report. Instead, the smart contract C records the total payout each oracle is eligible for, and they can cash out on request.

Reputation. The reputation system is primarily used by the oracles to prioritize the observation they will include in their report proposal. Indeed, an adversarial oracle could choose not to send back his clear observation at stage 7, thus slowing down protocol execution. To minimize this risk, each oracle is associated with a reputation number corresponding to the number of his observations that have been included in past valid reports. When an oracle tries to find a valid report proposal, it is in his interest to prioritize observations from nodes that have a high reputation number. When a submitted report becomes final, the smart contract increases by one the reputation number of each oracle that produced an observation contained in the said report.

2.7.2 The oracle network

In this section, we detail the algorithms executed by nodes in the oracle network. It consists of three algorithms. Algorithm 2 is a daemon that listens for event triggered by C. Algorithm 3 is instantiated by every p_i for each new request by the daemon. Its role is to propagate and collect observations among the network. Algorithm 4 is the report mining protocol. Its role is to build a report proposal out of the delivered observation and eventually submit this report to

Algorithm 1 Oracle smart contract

state
$lastPrice \leftarrow \perp$: last valid reported price
$reports \leftarrow [\perp]$: table of valid reports
$oracles \leftarrow [\perp]^n$: list of oracles' public keys and their reputation
$targetDifficulty \leftarrow 0$: current difficulty target number
$tergetReportTime \leftarrow 0$: Target report generation time

function $requestNewPrice()$
$reports[requestID].requestSubmitted \leftarrow time.now$
Emit event $newRequest()$

function $verifyProofOfWork(reportHeader)$
return $hash(reportHeader) \leq targetDifficulty$

function $verifySignatures(hashSignObs)$
return $\forall\, h_{\sigma_i} \in hashSignObs | verify_i(h_{\sigma_i})|$

function $verifyHashes(hashSignObs, observe)$
return $\forall o_i \in observe | hash(hashSignObs[i] = hash(o_i))|$

function $submitReport(requestID, [reportHeader, hashSignObs, observe])$
if $verifyProofOfWork(reportHeader) \wedge verifySignatures(hashSignObs) \wedge verifyHashes(hashSignObs, observe) \wedge observe.length \geq 2f + 1$ **then**
$reports[requestID] \leftarrow [reportHeader, observe, hashSignObs]$
$reports[requestID].requestFulfilled \leftarrow time.now$
$lastPrice \leftarrow median(observe)$
$\forall o_i \in observe | oracles[i].reputation \leftarrow oracles[i].reputation + 1|$
$adjustDifficulty()$
Emit event $finalReport(requestID)$
end if

function $adjustDifficulty()$
$l = reports.length$
$sum \leftarrow \sum_{i=l-100}^{l} reports[i].requestFulfilled - reports[i].requestSubmitted$
$averageReportTime \leftarrow sum/100$ ▷ Calculate the average report generation time over the last 100 reports
$targetDifficulty \leftarrow targetDifficulty * \frac{averageReportTime}{targetReportTime}$ ▷ Adjust the difficulty by a factor of the ratio between observed and desired report generation time

function $registerOracle(publicKey)$ ▷ Function restricted to administrator
$oracles.append([publicKey, 0])$ ▷ Register new oracle and set reputation to 0

function $setTargetReportTime(time)$ ▷ Function restricted to administrator
$targetReportTime \leftarrow time$

the oracle smart contract C. The oracle smart contract C maintains a list of oracles, their public keys, and their reputation. The client makes his request to the smart contract C. The instances of Algorithm 3 and Algorithm 4 can read the state of C at any time. They can access the list of oracles, their public keys, reputation, current request-id, and current target difficulty number. Those public variables are used implicitly in the pseudo-code presented here. We denote by $sign_i(m)$ the function that signs the message m with the private key of process p_i producing the signed message m_{σ_i}. Similarly $verify_i(m_{\sigma_i})$ verifies the signature of signed message m_{σ_i} with the public key of process p_i.

The daemon. Algorithm 2 is a deamon that is executed continuously by every process. This deamon awaits for $newRequest$ events from C to instantiate the observation gossip protocol presented in Algorithm 3. It is also responsible for stopping instances of Algorithm 3 and Algorithm 4 when a $finalReport$ event is emitted by C.

Algorithm 2 Oracle daemon executed continuously by every $p_j \in P$

Upon event $newRequest(requestID)$ from C **do**
  initialize instance $(requestID)$ of observation gossip protocol

Upon event $finalReport(requestID)$ from C **do**
  abort instance gossip protocol $(requestID)$
  abort instance report mining $(requestID)$

The observation gossip protocol. Algorithm 3 is the observation gossip protocol. Its goal is for the nodes to propagate observations messages among the network. It is instantiated by Algorithm 2 upon new request event emitted by C. Oracle nodes gather data from sources, hash and sign their observation, and broadcast it to every $p_i \in P$. Every oracle p_i maintains a list $hashSignObs$ of hashed-signed observations delivered by any $p_j \in P$. When a node p_i has received at least $2f + 1$ hashed-signed observations, he starts the report mining protocol presented in Algorithm 4. For the sake of simplicity and readability, we separated Algorithm 3 and Algorithm 4. In practice, those algorithms will be executed in parallel on a single machine and thus share the same local memory. This means that when Algorithm 3 starts a report mining instance for request $requestID$, the list of hashed observations $hashSignObs$ can still be updated by the observation gossip protocol. The report gossip protocol continues execution even if the report mining process has started. It is in the interest of the nodes to include as many observations as possible in their report proposal to minimize the chances of an adversarial

oracle blocking execution. This will be further developed in the paragraph about the report mining process. Yet it must be noted that the abort condition of the observation gossip process is a $finalReport$ event from C.

Algorithm 3 Observation gossip protocol instance $requestID$ (executed by every oracle p_i)

state
$observe \leftarrow [\perp]^n$: table of observations received in OBSERVATION messages
$hashSignObs \leftarrow [\perp]^n$: table of hashed signed observations received in HASHED-OBSERVATION messages

Upon initialization **do**
$v \leftarrow$ gather price value from data source
$observe[i] \leftarrow [time.now, assetPair, v]$
$hashSignObs[i] \leftarrow sign_i(hash(observe[i]))$
send message[HASHED-OBSERVATION, $hashSignObs[i]$] to all $p_j \in P$

Upon receiving message [HASHED-OBSERVATION, h_{σ_j}] from p_j **do**
if $verify_j(h_{\sigma_j})$ **then**
$hashSignObs[j] \leftarrow h_{\sigma_j}$
end if

Upon $|\{p_j \in P | hashSignObs[j] \neq \perp\}| = 2f + 1$ **do**
initialize instance report mining $(requestID)$

Upon receiving message [REPORT-PROPOSAL,$hashSignObs'$,$reportHeader$] from p_l **do**
if $hashSignObs[i] \in hashSignObs'$ **then**
if $\{\forall\ h_{\sigma_j} \in hashSignObs' | verify_j(h_{\sigma_j})|\} \wedge hash(reportHeader) \leq targetDifficulty \wedge hashSignObs'.length \geq 2f + 1$ **then**
send message[OBSERVATION, $observe[i]$] to p_l
end if
end if

The report mining protocol Algorithm 4 presents the report mining process. Much like in a proof-of-work-based blockchain the principle is for the oracles to rearrange the content of the report until the hash value of the report header is below a target difficulty number. Each report maintains a pool $hashSignObs$ of n pending observations that have not been yet included in a report. An observation has three attributes, the target asset pair, the observed price, and a timestamp. To this observation correspond an observation header that contains the public key of the oracle, the hash of the observation, and the oracle's signature. When oracle nodes try to find a valid report, they don't hash the full report. Instead, much like in a proof-of-work-based blockchain, they only hash the header of the report. This header consists of a timestamp a

nonce and most importantly the observations hash. The observations hash would correspond in a proof-of-work blockchain to the Merkle root. But because we don't need simple payment verification, building a Merkle tree out of the list of observations is unnecessary. Hashing the concatenated list of observations is sufficient in our case.

The way nodes rearrange their hash-signed observation list to find a valid report proposal is not explicitly developed in Algorithm 4. Indeed, similarly to proof-of-work-based mining, it is the responsibility of the nodes to find a strategy that maximizes their rewards and reduces their costs. However, we would advise that they rearrange their list according to the reputation of the other nodes. Indeed, the only way for an adversarial oracle to slow down or block protocol execution is not to respond with an OBSERVATION message on receiving a REPORT-PROPOSAL message. In order not to include dishonest parties in their report proposal, oracles should prioritize observations from oracles that have a good reputation. It is to be noted that to be considered valid by the smart contract C, a report does not need to have a clear observation for every hash-signed observation contained in the $hashSignObs$ list. Indeed, the only requirement is that the report contains at least $2f + 1$ clear observations.

2.8 Analysis

In this section, we prove that the PoWacle protocol satisfies the properties of the oracle problem definition as presented in Section 2.5.

Lemma 2.1

The PoWacle protocol satisfies the liveness property of a decentralized blockchain price oracle. ♡

Proof Consider a client request made to C at time t. Every correct node is triggered by this event and broadcasts its observation to every $p_j \in P$. Recall that correct oracles resend every message until the destination acknowledges it. Even if an adversary coalition can choose faulty nodes on the fly, they can not control more than $f < n/3$ process. Thus every correct oracle should eventually deliver a hash-signed observation from every correct node. This also holds for the process where nodes send clear observations to the proposer. Consequently, every correct node should be able to start the mining process. Let $d > 0$ be the difficulty target number. Due to the uniformity and non-locality properties of hash functions (outputs should be uniformly distributed), for each trial, there is a non-zero chance of finding a report whose hash value h is

Algorithm 4 Report mining algorithm instance $requestID$ executed by every p_i that has delivered more than $2f + 1$ hash-signed observation

state
$reportHeader \leftarrow \perp$

Upon initialization **do**
loop
$nonce \leftarrow 0$
$hashSignObs' \leftarrow prioritize(hashSignObs)$ ▷ Create a prioritized copy of the hash-signed observation list
while $reportHeader = \perp \vee nonce \neq 2^{32}$ **do**
if $hash(time.now, nonce, hash(hashSignObs')) \leq targetDifficulty$ **then**
$reportHeader \leftarrow [time.now, nonce, hash(hashSignObs')]$
send message[REPORT-PROPOSAL, $hashSignObs'$, $reportHeader$] to all $p_j \in P$
break
end if
$nonce \leftarrow nonce + 1$
end while
end loop

Upon receiving message [OBSERVATION, o_j] from p_j **do**
if $hash(o_j) = hashSignObs[j]$ **then**
$observe[j] \leftarrow o_j$
end if

Upon $observe.length \geq 2f + 1$
$C.submitReport(requestID, [reportHeader, hashSignObs', observe]$

less or equal to the difficulty d. Consequently, there is a finite Δ within which an oracle should find and submit a valid report to C. Because the observations contained in the report are from correct nodes, C should declare the report final. Thus the protocol satisfies the Δ-Liveness property of a decentralized blockchain price oracle.

Lemma 2.2

The PoWacle protocol satisfies the observation integrity property of a decentralized blockchain price oracle. ♡

Proof Consider a protocol execution where a report with value v has been declared final by C. To be considered final by C, a report must contain at least $2f + 1$ observations that have been individually signed by each oracle. The value v is the median of those observations. Since there are at most f faulty nodes, out of the $2f + 1$ observations, more than half have been produced by a correct node. Trivially, the median of those observations is either the one of a correct node

or the one of a faulty node but in the interval between a larger and a smaller value provided by honest nodes. Thus, the protocol satisfies the observation integrity property of a decentralized blockchain price oracle.

Lemma 2.3

The PoWacle protocol satisfies the uniformity property of a decentralized blockchain price oracle. ♡

Proof Recall that the value $lastPrice$ of C can only be updated by the smart contract itself when $requestNewPrice$ is invoked by a client. Thus trivially, if two clients c_1 and c_2 read this value at some time $t > 0$, the value they will read corresponds to the same report.

2.9 Conclusions

Price oracles are at the core of various applications in Decentralized Finance. The oracle problem, very well known in the industry and grey literature describes the ambivalence between blockchain that are supposed to be immutable through decentralization and oracles that input data from the outside world that cannot be verified by the blockchain itself. A perfect blockchain oracle system would have a very large number of observer gathering data from multiple sources and each of them reporting the data directly to the chain without communicating between them. However because of design constraints intrinsic to blockchain system such as transaction fees, it is yet more cost effective that a single oracle gather data from a network of observer and publish a report containing signed observations. In this chapter we review some existing price oracle solutions and then propose a definition for the decentralized blockchain price oracle problem. We propose a new protocol different from the ones in the literature as it is designed for asynchronous Byzantine prone environment. The protocol uses a decentralized network of observers as well as a light proof-of-work scheme to select a report producer. We semi-formally define the protocol and prove its correctness.

Chapter 3 Atomic cross-chain swap

3.1 Introduction

3.1.1 Context

Bitcoin is considered to be the first-ever digital payment system based on blockchain technologies. During its first years, in the early 2010s, Bitcoin had nearly no competition, representing around 90% of the total crypto-asset market capitalization. At this time it was common to think that Bitcoin would become the one and only blockchain-based digital cash system that would fit all applications.

A few years later, in 2015, the Ethereum network was launched. As explained earlier, this blockchain popularized smart contracts; programs that are deployed and executed on-chain. This new iteration of blockchain technologies is referred to as the second-generation blockchains.

The third-generation blockchains are the ones that offer the same features as generation two blockchains while being less electricity-intensive and offering better scalability.

The rise of those new generations of blockchain brought us a new range of applications. Smart contracts opened the way for payments in decentralized applications while scalability improvements allowed us to envision applications where smaller devices such as IoT devices could exchange assets. For example, we could think of autonomous electric vehicles that would pay the charging stations for electricity.

The rise of those new blockchain systems had a lot of impact on the market. The dominance of Bitcoin in terms of market capitalization has greatly dropped and is at the time of writing around 40%. While Bitcoin has proven its strengths as a store of value, it is now clear that instead of a "one size fits all" solution, meaning that all transactions, smart contracts, or anything else are performed on a single chain, we will have a variety of blockchains and crypto-assets, each one fulfilling a particular use case or market.

With the certainty that a variety of assets hosted on different blockchain systems will coexist came the need for blockchain interoperability.

3.1.2 The centralized model for cross-chain asset exchange

Today the use case that generates the most volume in the blockchain industry is crypto-asset trading. In 2021 the volume of crypto-asset trading reached a mean of around $2 trillion per day.

Today, most cross-chain exchanges are achieved with the help of a trusted third party, most often an exchange platform. Centralized exchange platforms (CEXs) don't support peer-to-peer cross-chain transactions between users. Instead, users deposit their funds inside the platform's digital wallets and the transactions are handled by the platform's system. Most of the transactions are not even published to the chain. Platforms keep a separate record of their clients assets; the only transactions that are written to the blockchain are deposits and withdraws.

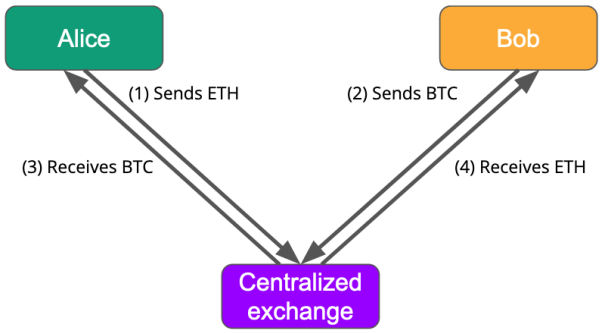


Figure 3.1: Illustration of a cross chain swap using a centralized exchange

This is not a satisfactory solution, because as stated in Bitcoin's white paper [67] all benefits of blockchain technologies are void if a centralized financial institution is required. Indeed, trusting a centralized third party comes with all the flaws of centralization. First, centralized platforms are attractive target for attackers [87]. The largest crypto-asset thefts to this day involve centralized platforms that have been hacked. Centralization can also lead to governance issues [96]; since platforms have total control over their client's assets, they can artificially influence the market by buying or selling a large amount of crypto-assets. We talk about the "whale effect" or "pump and dump" schemes. We can see in Figure 3.2 that out of the ten richest Bitcoin addresses, five of them belong to centralized crypto-asset trading platforms, controlling alone around 3.5% of all bitcoins in existence.

The platforms also have total control over the order book and the display of executed transactions. Thus they can artificially create and display transactions that have never happened, or that have not been requested by any client. It has been shown in [70] that major centralized crypto-asset platforms have been artificially faking trading volume to attract investors and

	Address	Balance △1w/△1m	% of coins
1	34xp4vRoCGJym3xR7yCVPFHoCNxv4Twseo wallet: Binance-coldwallet	265,480 BTC ($11,621,145,143) / +0.00621 BTC	1.40%
2	bc1qgdjqv0av3q56jvd82tkdjpy7gdp9ut8tlqmgrpmv24sq90ecnvqqjwvw97 wallet: Bitfinex-coldwallet	168,010 BTC ($7,354,492,702)	0.8876%
3	1P5ZEDWTKTFGxQjZphgWPQUpe554WKDfHQ	123,262 BTC ($5,395,669,403) +1865 BTC / +4059 BTC	0.6512%
4	3LYJfcfHPXYJreMsASk2jkn69LWEYKzexb wallet: Binance-BTCB-Reserve	116,601 BTC ($5,104,114,532) / +43000 BTC	0.6160%
5	37XuVSEpWW4trkfmvWzegTHQt7BdktSKUs wallet: 77604498	94,505 BTC ($4,136,880,162)	0.4993%
6	38UmuUqPCrFmQo4khkomQwZ4VbY2nZMJ67 2-of-6 wallet: OKEX-coldwallet	84,067 BTC ($3,679,961,444)	0.4441%
7	1FeexV6bAHb8ybZjqQMjJrcCrHGW9sb6uF	79,957 BTC ($3,500,058,449)	0.4224%
8	bc1qa5wkgaew2dkv56kfvj49j0av5nml45x9ek9hz6	69,370 BTC ($3,036,619,823)	0.3665%
9	3Kzh9qAqVWQhEsfQz7zEQL1EuSx5tyNLNS 2-of-3 wallet: OKEX-coldwallet	68,233 BTC ($2,986,832,979) -437.4 BTC / +3851 BTC	0.3605%
10	1LdRcdxfbSnmCYYNdeYpUnztiYzVfBEQeC	53,880 BTC ($2,358,553,347)	0.2847%

Figure 3.2: Top 10 richest Bitcoin addresses and their respective share of the total existing bitcoins

traders. Finally, those platforms apply commissions to their clients as their business model.

For all the reasons listed above, we can conclude that the current centralized model for exchanging assets across chains is not acceptable. Thus we would like to propose new models where processes can exchange assets across chains without the need for a trusted third party. In these new models, the safety of the cross-chain transactions will not be guaranteed by a third party but by the protocol itself. To achieve this, the new model needs to integrate some interoperability between blockchains.

3.1.3 Interoperability strategies classification

As explained earlier, there currently is no consensus in the academia over the definition of blockchain interoperability. Neither is there consensus over the categorization of blockchain interoperability solutions.

The last survey to our knowledge, [90] divides the field into three categories "Chain-based" interoperability mainly targets public blockchains, especially for the applications of cryptocurrencies. "Bridge-based" interoperability targets the implementation of a bridge as a connection

component between homogeneous and heterogeneous blockchains. "Dapp-based" interoperability aims at inducing communication among blockchain-based decentralized applications.

Belchior et al. previously proposed a similar classification. In [9], they define three categories. "Public connectors" are chain interoperability strategies across public blockchains supporting cryptocurrencies. "Blockchain of Blockchains" are frameworks that provide reusable data, network, consensus, incentive, and contract layers for the creation of application-specific blockchains. Finally "Hybrid connectors" are interoperability solutions that do not belong to previous categories.

While both of those proposals are pertinent in the context of a systematic literature review, we find that those classifications are overthought and because of nuances and implementation details, it is often difficult to state which category a project belongs to. Moreover, those classifications focus on the components that are being connected rather than on the strategy employed. Thus we prefer the classification from Butterin [89], which happens to be the oldest. He defines three interoperability strategies:

- Centralized or multi signatures notary schemes: a solution where an entity or a group of entities agree to perform some action if some conditions are met.
- Hash time locking: a solution that involves locking assets on a smart contract and setting the release condition of the said asset to both a timelock and a hashlock.
- Side-chains and relays: a solution that involves verifying events that happened on another chain thanks to a piece of software called a relay.

It is to be noted that both Belchior and Wang acknowledge Butterin classification. They proposed a different classification because their goal was to review many heterogeneous projects. But in our case, Butterin's classification is sufficient and in our opinion scientifically more accurate. In the next section we are going to introduce the hash time locking strategy with hash time locked contracts.

3.1.4 Interoperability results

While the definition and classification of blockchain interoperability remains cloudy in the literature, some results deserve to be presented here.

In [54] Lafourcade et al. defines a blockchain B as a tuple $B = (L, W, Emit, Mine)$ where:

- L is the ledger is a list of transaction T and their proofs P such that:

$L = [([t_{1,1}, t_{1,2}, ...], p_1), ..., [([t_{n,1}, t_{n,2}, ...], p_n)]$.

- W is the pool of awaiting transactions.
- Emit is a deterministic algorithm taking one transaction $t \subset T$ and W as input, and returning an updated pool $Emit(t, W) = W \cup t$.
- Mine is an algorithm taking L, W and returning a new ledger L', a new pool W'.

Lafourcade states that interoperability occurs when a transaction on a ledger A depends on the state of ledger B. More formally he says that two blockchains A and B are interoperable if there exists a transaction t such that t is accepted by the consensus of A if $L_A \times L_B \in \omega_A \times \omega_B$ and rejected otherwise. With ω_A and ω_B being non-empty subsets of the possibles states of A and B's ledgers.

This definition automatically excludes centralized and multisig solutions and hash locking. While our conception of interoperability goes behind Lafourcade's definition, considering his definition he obtains the following result. He states that we cannot build interoperable blockchains without one of them containing the other. Interoperable blockchains are equivalent to creating a 2-in-1 blockchain containing both ledgers. This academic result proves itself right in practice as most emerging interoperable blockchain solutions are built on top of the relaying technique, which consists in storing some part of ledger B in ledger A, to verify events of B from A.

In [99] Zamyatin et al. formalize the problem of cross-chain communication or CCC. They consider two processes P and Q as well as two ledgers L_x and L_y. They define CCC as "the synchronization of processes P and Q such that Q writes a transaction tx_Q to L_y if and only if P has written transaction tx_P to L_x. The intuition is that tx_P and tx_Q are two transactions that must either both, or neither, be included in L_x and L_y, respectively. For example, they can constitute an exchange of assets which must be completed atomically." As we can see this problem definition is similar to Lafourcade's interoperability definition. Indeed, both state that the execution of a transaction in a ledger X must depend on a transaction of ledger Y. In this paper, Zamyatin et al. prove that cross-chain communication is impossible without a trusted third party. However, the trusted third party can be distributed entity such as a blockchain system.

3.1.5 Hash time locked contract (HTLC)

A proposed strategy for atomic cross-chain swaps relies on hash time locking [89]. This technique consists of locking some digital assets inside a smart contract or script and setting the release condition to both a time condition and the revelation of a cryptographic hash pre-image. Herlihy's protocol [40] for instance, makes use of this technique. The protocol is as follows: Alice has X assets of chain BC_a, Bob Y assets of chain BC_b and they are willing to exchange one for the other. They agreed on the exchange rate through off-chain communication channels (1). Alice creates a random secret s and computes its hash $h = H(s)$ (2). She publishes a contract SC_1 on blockchain BC_a where she locks her assets and sets the release condition to the revelation of the pre-image of h or the expiration of a time lock Δ_1 (3). Bob learns h from SC_1 and sets up a similar contract on BC_b with a time lock $\Delta_2 < \Delta_1$ (4). This phase of setting up the contract before the exchange is shown in Figure 3.3

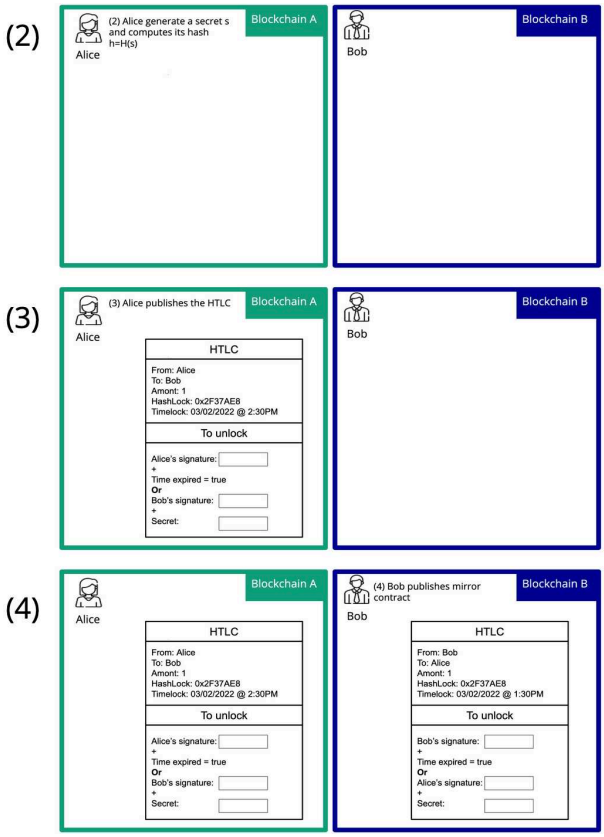


Figure 3.3: Illustration of the contracts' setup phase of an atomic cross-chain swap using hash time locked contracts

Alice redeems SC_2 by revealing s and by doing so triggers the transfer of Y from SC_2 to her address (5). Bob learns about s from this transaction and can now proceed to redeem X from SC_1 (6). This phase is shown in Figure 3.4

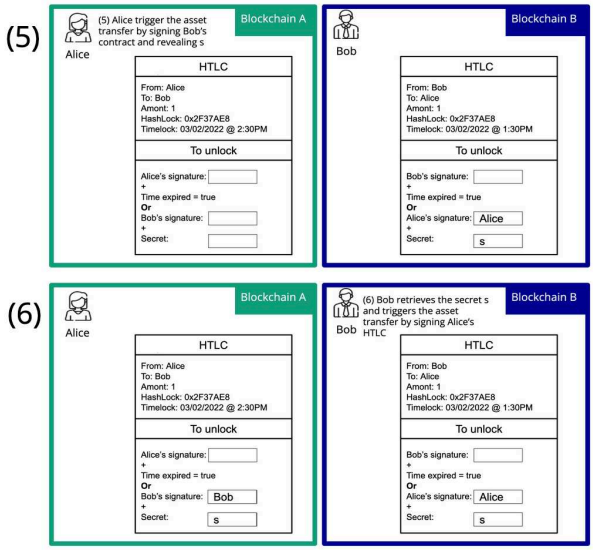


Figure 3.4: Illustration of the redeem phase of an atomic cross-chain swap using hash time locked contracts. In this case both participants complied to the protocol and both asset transfer were triggered.

If for some reason, a participant stops in the process, for example, if Bob does not set up an HTLC contract and does not lock assets, Alice can safely wait for the time lock to expire and proceed to refund. This is shown in Figure 3.5

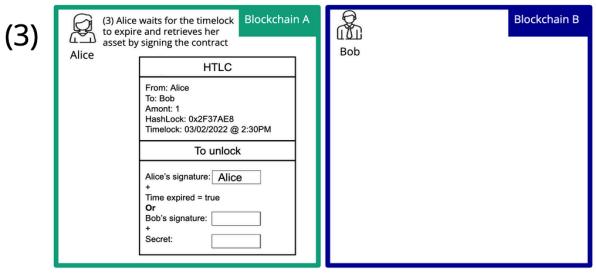


Figure 3.5: Illustration of the refund phase of an atomic cross-chain swap using hash time locked contracts. Here Bob has not committed to the swap so Alice waits for the timelock to expire to retrieve her locked funds

3.1.6 Herlihy's algorithm

The hash time locking technique for atomic cross-chain swap was first discussed here [86] on a Bitcoin forum in 2013. However, blockchain systems that enabled smart contracts were in their infancy and Ethereum was not even launched yet. Thus the first formal description of the protocol from Maurice Herlihy in [40] only came years later in 2018.

As explained in Section 3.1.5, the protocol is based on smart contracts where funds are locked and where the release condition of the said funds is either the revelation of a hash

pre-image or the expiration of a time-lock.

Those contracts, called hash time-locked contracts (or HTLCs) can be defined by four parameters h, A, B, v, t. Parameter h is the computed image of a secret $h = H(s)$, H being a cryptographic hash function. Parameter A represents the blockchain address of the swap initiator and B the one of the recipients (or participant). Parameter v is the amount of coins transferred and t is the period of time during which the assets cannot be unlocked by A.

We presented the protocol as a swap of assets between two users willing to exchange assets between two blockchains. However, this protocol can be extended to a larger amount of participants and any type of digital assets. In his paper, Maurice Herlihy takes the example of a three-way swap. Alice, Bob, and Carol each possess some assets of some blockchain and are willing to exchange at some exchange rate they agreed upon before the swap. Alice has assets of blockchain BC_1, Bob of chain BC_2 and Carol BC_3. Alice generates a secret s and computes its cryptographic hash h. She then publishes an HTLC to BC_1, sets Bob as the recipient, 6Δ as a timelock, and h as the hashlock. Δ being a sufficient amount of time for any participant to execute a transaction on any chain. Once he is able to see Alice's HTLC, Bob publishes a similar contract on BC_2 with Carol as recipient and 5Δ as a timelock. Carol does the same on BC_3 with 4Δ as a timelock and Alice as the recipient.

Once all contracts have been set up, Alice proceeds to redeem the assets locked by Carol by revealing the secret s. Carol learns the secret s by parsing the transaction and proceeds to redeem Bob's assets. Finally, Bob learns s from this transaction and can proceed to redeem Alice's contract.

As we can see, as long as the last participant sets as a receiver the first participant, and as long as the original timelock is sufficiently high for all operations, the HTLC technique for swapping assets can, in theory, be extended to any amount of participants and assets.

Such a protocol can be represented as a digraph $D = (V, A)$ where each vertex in V represents a party, and each arc in A represents a proposed asset transfer from the arcs head to its tail via a shared blockchain. Figure 3.6 presents the contract's deployment phase of this scenario, as a digraph. Figure 3.7 shows the redeem phase.

3.1.7 Our implementation

We implemented a version of Herlihy's atomic cross-chain swap protocol between the two most important blockchains in terms of total market capitalization at the time, Bitcoin and

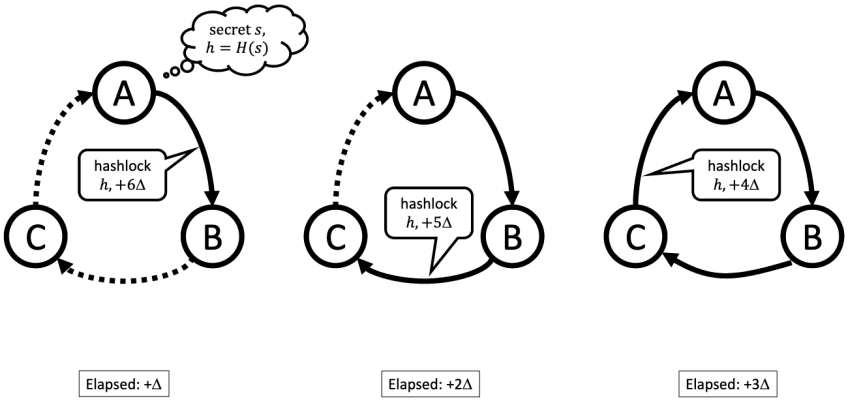


Figure 3.6: Atomic cross-chain swap: deploying contracts [40].

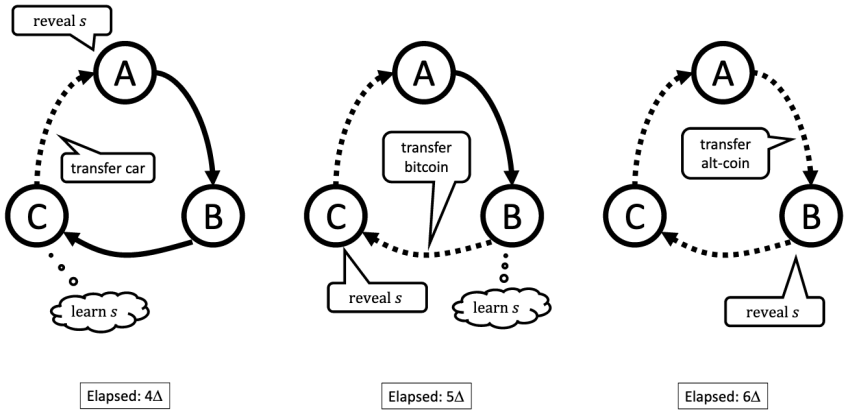


Figure 3.7: Atomic cross-chain swap: triggering arcs [40].

Ethereum. The hash-time locking technique doesn't require both blockchains to be interoperable in Lafourcade's definition (see Section 3.1.4). Thus to implement Herlihy's swap, it is only required to implement a hash time-locked contract on both chains.

Hash time-locked contract implementation for Ethereum As presented in Section 1.4.1, Ethereum has a smart contract development language called solidity. This Turing complete language allows for high-level, object-oriented programming paradigms. Thus it is possible with solidity to create complex data structures and functions. We took advantage of the features offered by Solidity to define a data structure to store a swap's details. We then created a mapping that associates each swap with a unique identification number.

As we can see in Figure 3.8 each swap is associated with a state variable that can be in one of four possible states. Invalid means the swap does not exists yet, opened meaning that the asset has been locked, redeemed meaning that the swap took place, and refunded meaning that the swap did not take place and that the swap's initiator has gotten its funds refunded. By default a swap is invalid. Redeemed and refunded are what we call final states. Because the protocol is

```
3   contract AtomicSwapEther {
4
5     struct Swap {
6       uint256 timelock;
7       uint256 value;
8       address initiator;
9       address recipient;
10      bytes32 secretLock;
11      string secretKey;
12    }
13
14    enum States {
15      INVALID,
16      OPENED,
17      REDEEMED,
18      REFUNDED
19    }
20    //Mapping to store swaps
21    mapping (bytes32 => Swap) private swaps;
22    //Mapping to store states
23    mapping (bytes32 => States) private swapStates;
```

Figure 3.8: Solidity HTLC swap data structures

acyclic, i.e. a swap's state can only transition to the next one until he reaches a final state, there are only three state transition functions to implement; Initiate makes the transition between Invalid and opened, redeem from open to redeemed and refund from opened to refunded.

```
55    function initiate(bytes32 _swapID,address _recipient,bytes32 _secretLock,uint256 _timelock)
56    public
57    onlyInvalidSwaps(_swapID)
58    payable {
59      // Store the details of the swap.
60      Swap memory swap = Swap({
61        timelock: _timelock + now,
62        value: msg.value,
63        initiator: msg.sender,
64        recipient: _recipient,
65        secretLock: _secretLock,
66        secretKey: ""
67      });
68      swaps[_swapID] = swap;
69      swapStates[_swapID] = States.OPEN;
70      // Trigger open event.
71      emit Initiated(_swapID, _recipient, _secretLock);
72    }
```

Figure 3.9: Solidity HTLC swap initiate transition function

Figure 3.9 presents the initiate transition function. As we can see it instantiates a swap structure with the provided parameters and stores it in the smart contract's memory. We can see that the initiator address and the amount transacted are not explicitly given in the function's argument. This is because Solidity is provided with a special object called msg that contains, among others, the address of the function caller and the value transacted. We can also see that an event is emitted at the end of the function at line 71. This has no logical or algorithmic purpose, it is used for logging and easier parsing of the chain. When the other user will have to parse the chain for auditing purposes, he can simply look for those events.

As we can see in Figure 3.9 and 3.4, between the declaration of the function and the implementation lies some declarations starting with "only". Those are specific modifiers that are often used in Solidity to restrict access to a function. It can be used to restrict a function to

```
74  function redeem(bytes32 _swapID, string _secretKey)
75  public
76  onlyOpenSwaps(_swapID)
77  onlyWithSecretKey(_swapID, _secretKey){
78    // Close the swap.
79    Swap memory swap = swaps[_swapID];
80    swaps[_swapID].secretKey = _secretKey;
81    swapStates[_swapID] = States.CLOSED;
82    // Transfer the ETH funds from this contract to the withdrawing trader.
83    swap.recipient.transfer(swap.value);
84    // Trigger close event.
85    emit Redeemed(_swapID, _secretKey);
86  }
87
88  function refund(bytes32 _swapID)
89  public
90  onlyOpenSwaps(_swapID)
91  onlyExpirableSwaps(_swapID){
92    // Expire the swap.
93    Swap memory swap = swaps[_swapID];
94    swapStates[_swapID] = States.EXPIRED;
95    // Transfer the ETH value from this contract back to the ETH trader.
96    swap.initiator.transfer(swap.value);
97     // Trigger expire event.
98    emit Refunded(_swapID);
99  }
```

Figure 3.10: Solidity HTLC swap final states transition functions

a certain address or to processes that are able to pass a certain challenge. We used modifiers to verify that the process asking for a redeem is able to provide the pre-image of the hashed secret.

```
49  modifier onlyWithSecretKey(bytes32 _swapID, string _secretKey) {
50    require(swaps[_swapID].secretLock == sha256(abi.encodePacked(_secretKey))
51    , "modifier throws : secret key does not match");
52    _;
53  }
```

Figure 3.11: Check secret key Solidity modifier.

The modifier presented in Figure 3.11 verifies that the provided key matches the hash-lock of the corresponding swap. Similar modifiers are used to check if the address calling the function is allowed to do so or to verify that the swap is in the correct state to proceed.

Hash time-locked contract implementation for Bitcoin The implementation of a hash time-locked contract in Bitcoin is radically different than the one for Ethereum. Indeed, Bitcoin was not designed for smart contracts and is UTXO based rather than account-based. This is detailed in Section 1.4.1, but to sum up, Bitcoin keeps track of what is called UTXO for unspent transaction output. Each UTXO represents an amount of bitcoins that have not been spent yet. Each UTXO has a lock script that describes how this transaction output can be spent. The simplest lock script is the P2PK script for Pay to Public Key. To unlock the asset, i.e. to spend the UTXO, one must provide a digital signature corresponding to the public key contained in the lock script. If the provided signature matches the public key, then the UTXO can be spent generating one or several new UTXOs.

Thus in Bitcoin, rather than programming a contract that implements the functionalities of an HTLC contract, we will build lock scripts that fulfill the same functionalities. Those scripts

will be implemented using the Bitcoin SCRIPT, Bitcoin's scripting language.

There are three scripts to be implemented; an initiate script, that will hash time lock the assets, a redeem script that unlocks the initiate script by providing the secret key, and a refund script that unlocks the assets when the time lock is expired.

To build the transactions we developed an HTML interface that allows the user to input the desired time-lock and hash-lock. Then, thanks to a javascript library we were able to build the transaction with the values inputted by the user. It was also used to manipulate the keys and retrieve the user's UTXOs. Here we will focus on the script itself.

```
53    // build the script
54    var script = bitcore
55        .Script()
56        .add('OP_IF')
57        .add('OP_SHA256')
58        .add(new Buffer(hashKey.toString(), 'hex'))
59        .add('OP_EQUALVERIFY')
60        .add(bitcore.Script.buildPublicKeyHashOut(bitcore.Address.fromString(toAddress)))
61        .add('OP_ELSE')
62        .add(bitcore.crypto.BN.fromNumber(timeLock).toScriptNumBuffer())
63        .add('OP_CHECKLOCKTIMEVERIFY')
64        .add('OP_DROP')
65        .add(bitcore.Script.buildPublicKeyHashOut(bitcore.Address.fromString(fromAddress)))
66        .add('OP_ENDIF');
```

Figure 3.12: Bitcoin HTLC initiate script.

Figure 3.12 presents the building of the initiate lock script. It is inspired by the work of Poon et al. [78] on the Bitcoin lightning network. This script describes the two ways the output can be spent. The first one is from lines 56 to 60 inside the "if" statement describes how to spend the UTXO by providing the secret key. It corresponds to the redeem state transition. The second one from lines 61 to 66 in the "else" statement describes how to unlock the UTXO once the time-lock is expired, this is the refund state transition.

Line 59, the OP_EQUALVERIFY opcode checks if the provided key once hashed matches the hash lock defined line 58. If it matches, the instruction line 60 is executed. As we can see this instruction transfers the UTXO to the $toAddress$ which is the recipient address. This corresponds to a redeem.

Line 63, the OP_CHECKLOCKTIMEVERIFY checks if the time-lock provided in line 62 has indeed expired. If so, it executes the instruction line 65 that transfers the asset back to the initiator, the $fromAddress$. This corresponds to a refund.

```
96    // setup the scriptSig of the spending transaction to spend the p2sh-cltv-p2pkh redeem script
97    refundTransaction.inputs[0].setScript(
98        bitcore.Script.empty()
99            .add(signature.toTxFormat())
100           .add(new Buffer(myPublicKey.toString(), 'hex'))
101           .add(new Buffer(toHex(key).toString(), 'hex'))
102           .add('OP_TRUE')
103   )
```

Figure 3.13: Bitcoin HTLC redeem script.

Figure 3.13 presents how the redeem script is built. We can see in line 101 that the secret key is provided to the lock script.

```
84      // setup the scriptSig of the spending transaction to spend the p2sh-cltv-p2pkh refund script
85      refundTransaction.inputs[0].setScript(
86          bitcore.Script.empty()
87              .add(signature.toTxFormat())
88              .add(new Buffer(myPublicKey.toString(), 'hex'))
89              .add('OP_FALSE')
90      )
```

Figure 3.14: Bitcoin HTLC refund script.

Figure 3.14 presents how the refund script is built. Nothing else than the initiator's signature is provided. Indeed, since the asset will be unlocked with a time condition, the script will refer to the network's time.

3.1.8 Discussion on latency

The most important factor when evaluating the performance of this implementation is the value of the time locks. The latency of the protocol can be defined as the time difference between the moment a party locks its asset and the moment where it gets its payoff or gets refunded. If the swap goes as planned and both parties exchange their assets, then the maximum latency is equal to the time lock value plus the time for the redeem transaction to be finalized. If the protocol does not go as planned and for some reason, a party halts at some point of the protocol, then the minimum latency is equal to the time lock value plus the time for the refund transaction to be finalized. Obviously, a party could choose not to send the refund transaction but it would be against their own interest. Thus we will only consider the minimum latency in the case where the swap does not happen. We can see here that, no matter the scenario, the latency of the protocol is equal to the time-lock value plus the time for a transaction to be finalized.

Let Δ_1 be the time-lock value of the initiator's HTLC (the one that created the secret) and Δ_2 the time-lock of the participant (the one that learns the secret). We have already seen that Δ_2 must be smaller than Δ_1 because if it was not, the initiator could wait for the very last moment and send both a redeem on the participant's contract and a refund on his own contract. However, to determine the protocol's latency we have to define those values.

Defining those values turns out to be a very challenging problem and this all comes down to the finality of proof-of-work blockchains. In proof-of-work blockchains, there is a time difference between the moment a transaction message is sent and the moment this transaction is included in a block. Moreover, even when a transaction has been included in a block, there

is a probability that this block gets reverted in the event of a fork. Therefore, to determine a time-lock value that would guarantee the safety of the swap, we have to determine a time period that is long enough for a transaction to be finalized with high probability. We will discuss this problem in the next Section, however, in this implementation, our conclusion was to set the time-lock to a very high value, for example, 24 hours (knowing that the average transaction validation time for Bitcoin is around one hour). We consider this is an optimistic decision; if the swap takes place, which should happen in the majority of the cases, the latency can be lower than the time-lock. If the swap does not take place, the participants will get refunded the next day. This does not compare with the centralized solutions performances but it guarantees safety with high probability.

3.2 R-SWAP: Relay based atomic cross-chain swap protocol

3.2.1 Introduction

In this section, we will propose R-SWAP, a protocol that we designed to overcome the existing strategies to perform atomic cross-chain swaps. We will start by reviewing the existing solutions to achieve atomic cross-chain swaps and discuss their limitations. Then we will present our proposal.

Hash time locking As stated by Buterin et al. in [89] one of the possible strategies for atomic cross-chain swaps relies on hash time locking. This technique was formally described by Herlihy in [40] and we implemented the protocol between Bitcoin and Ethereum. We already described the protocol in detail in the previous Section 3.1.5. Here we are going to focus on the limitations of this technique. This for the numbering of the steps of the protocol please refer to Section 3.1.5.

Currently proposed protocols that use the hash time locking technique such as [40] are subject to several limitations. They suffer from a potential safety violation, they may be economically unfair and have bad ergonomics [97, 98]. Indeed, after stage (5), if for some reason Bob is unable to broadcast his redeem transaction message (client's crash, DDOS attack on the client or network, packet loss, etc.), Alice can wait for the time lock to expire, send a refund transaction message and retrieve both assets, causing a safety violation. Another weakness is that after stage (3), Bob can choose not to publish a contract SC_2, causing Alice's

assets to be locked until the expiration of Δ_1 [39]. This is not a proper liveness violation, but in a highly competitive market such as crypto-assets trading, having an adversary's funds locked up for a certain amount of time is a great advantage. Furthermore, the cost of such an attack is null for Bob given that at no point he had to prove ownership of assets. Finally, this protocol has poor ergonomics; it requires both participants to stay online during the swap, to run two blockchain clients each, and to perform auditing tasks and transactions on both chains. This is very error-prone for the participants.

Relays Another strategy to achieve atomic cross-chain swaps relies on *relays* [26, 49, 89]. Relays are abstractions (in general a smart contract or a script) hosted on some chain BC_a that has light client-like verification capabilities over chain BC_b. For each new block appended to chain BC_a, the block header is passed on to the relay on chain BC_b. The relay itself implements the standard verification procedure of chain BC_a's consensus algorithm and can therefore verify the validity of the block. Once the proof of work has been verified, in the case of a Proof of Work (or PoW) blockchain, or the two-thirds of validators' signatures, in the case of a Byzantine Fault Tolerant (or BFT) blockchain, it is possible to verify any transaction of chain BC_a from chain BC_b. To prove the validity of a transaction, a user should provide the transaction details as well as the Merkle proof of the said transaction. With that information and the block header in its storge, the relay smart contract should be able to determine if it is a valid transaction of BC_a or not.

Figure 3.15 presents the overview of the architecture of a Bitcoin relay hosted on a polkadot parachain. This was implemented by interlay in [44]. The utils and the parser contains the logic that implements the Bitcoin SPV. When submitted a Bitcoin block header, the relay is able to check the block's proof-of-work. The relay is also capable of verifying a single transaction given that it is provided with the Merkle proof of the said transaction.

With the light client-like verification capabilities of chain BC_a from chain BC_b, we can imagine the following scenario. Bob has X assets of chain BC_b. He is willing to exchange them for Y assets of chain BC_a. Bob sets up a smart contract SC_1 and locks his assets in it (1). This smart contract SC_1 is set to release the assets to anyone providing proof that they made a payment of Y assets of chain BC_a to Bob's address. Alice, who is interested in this trade, transfers Y assets to Bob's address (2). She retrieves the transaction hash tx and provides it to SC_1 (3). SC_1 calls the relay and asks for verification of transaction tx (4). The relay verifies

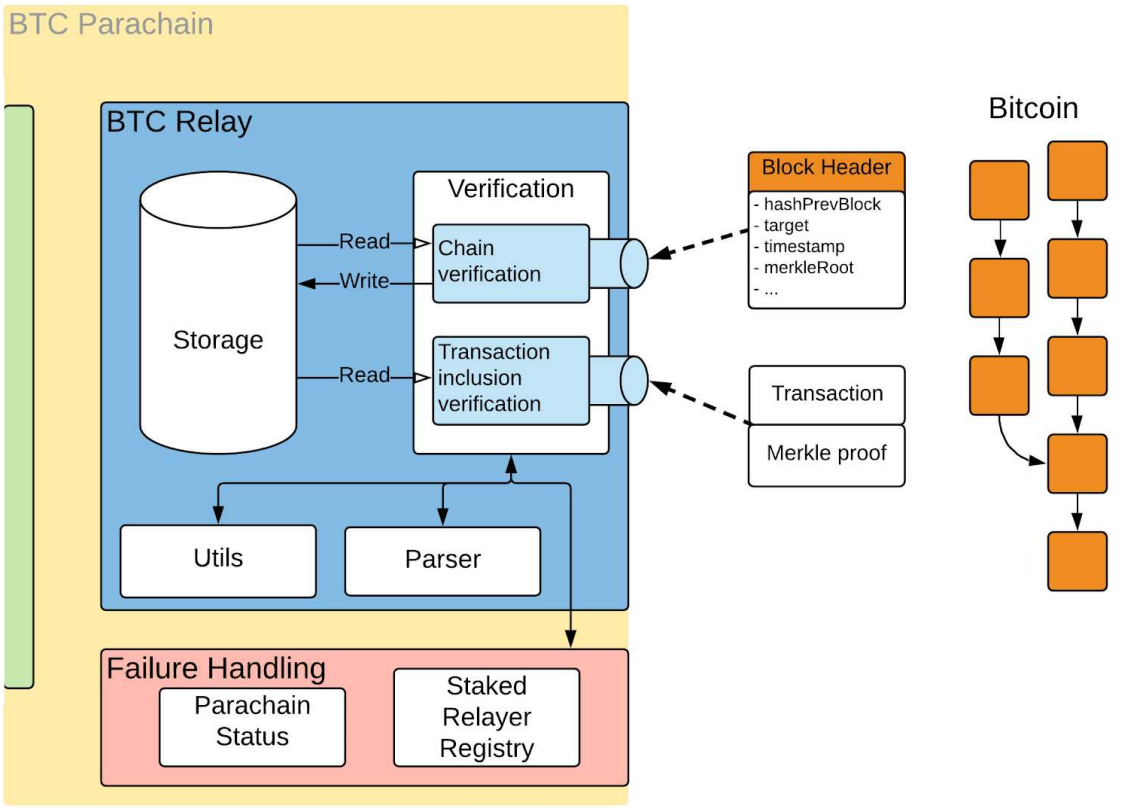


Figure 3.15: Overview of the BTC-Relay architecture. Bitcoin block headers are submitted to the Verification Component, which interacts with the Utils, Parser and Failure Handling components, as well as the Parachain Storage. From [44]

that the transfer has taken place and if so, returns ok to SC_1 (5). On receiving the answer from the relay, SC_1 transfers the X assets of BC_b to Alice's address.

Though this approach seems like a reasonable solution to achieve atomic cross-chain swaps, it has several flaws that can lead to a safety violation, liveness violation, and commercial unfairness. The potential safety violation comes from a race condition type of attack. Indeed at stage (2), it is possible that another person, for instance, Carol, is interested in the swap and sends some assets to Bob's address. Alice and Carol will only see the other one's transfer when the next block is mined. Therefore there is no way for them to coordinate their transfers. Bob will end up with twice the assets he wanted and he has no incentive to refund the losing party. The potential liveness violation comes from the fact that after stage (1) Bob has no guarantee that someone will take the swap. Therefore his assets could be locked up forever leading to a liveness violation. Finally, the protocol is commercially unfair. Indeed, after stage (1) the rate of the swap is fixed and the assets locked up. With the high volatility of cryptocurrency markets, Alice can safely wait for the price to evolve in a direction that is profitable to her and

then decide to perform the swap or not. This is commercially unfair to Bob.

Crypto-currency-backed assets (CBAs) The most recent proposed strategy for cross-chain swap is cryptocurrency-backed assets (or CBAs) such as [98]. Cryptocurrency-backed assets are tokens issued by a smart contract on chain BC_i that are backed at a one-to-one ratio by assets locked on chain BC_b. Thus each token $i(b)$ hosted on chain BC_i represents a unit of asset of chain BC_b. Each token $i(b)$ is backed by an asset on chain BC_b held in custody. To issue a token $i(b)$, one must provide proof that a unit of asset of chain BC_b has been locked in a *vault* (the custody). The proof, the locking transaction, is then verified thanks to a blockchain relay. Once the proof is verified, the token $i(b)$ is issued. As they now exist in the same self-contained blockchain system, $i(b)$ tokens can be atomically swapped with assets of chain BC_i (like ERC-20 tokens). To recover the assets locked on chain BC_b, one must *burn* an equivalent amount of token $i(b)$. This proof-of-burn is then submitted to the *vault*, which will redeem the assets. To ensure that the vault behaves honestly and redeems the assets, it must lock assets of chain BC_i in the smart contract as collateral. A single *vault* will only be able to issue as many $i(b)$ tokens as it has locked assets of BC_i in collateral, times the exchange rate. If the exchange rate ϵ between assets of BC_i and BC_b becomes critical (i.e., if the collateral value is too low to reimburse the token owners), the collateral is liquidated and sent to the token owners as payback.

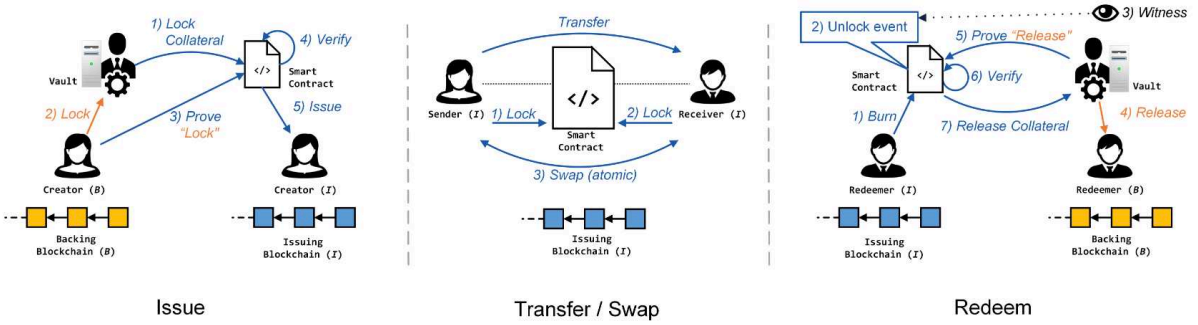


Figure 3.16: High-level overview of the Issue, Swap and Redeem protocols in XCLAIM

Although protocols such as [98] show good performance, reasonable costs, and allow asset portability across chains, they do not solve the problem of atomic cross-chain swaps. Such protocols can be assimilated with side-chain pegging like in [7]. The ownership of the assets of chain BC_i is not transferred. Instead, a token of equal fiat value is issued on chain BC_i. Moreover strong assumptions are made on the price oracle. It is assumed in the model that the exchange rate ϵ between assets of BC_b and BC_i given by the oracle is correct. Under weaker

assumptions (i.e, if the exchange rate returned by the oracle is not correct), a safety violation could occur. The liquidation of the collateral is triggered by the exchange rate ϵ given by the price oracle. If an attacker can achieve an "Oracle poisoning attack", he could trigger liquidation of the collateral and lead to a safety violation. Such attacks have already happened [46, 80]. Our proposal does not suffer from this kind of risk because price oracles are not involved.

3.2.2 Distributed Ledger Model

As presented in the model of Chapter 2, a distributed ledger (a.k.a. blockchain) BC is an append-only list of blocks chained together. Each block b_i somehow contains a list of transactions as well as a variable pointing to the previous block b_{i-1}, hence the name blockchain. The blockchain supports two types of operations READ(BC) and APPEND(BC, b_i).

Ledger conflict This definition only applies to probabilistic delivery blockchains such as Bitcoin. A ledger conflict (or fork) is an event during which two concurrent blocks have been found at the same height h. Due to the system being asynchronous, if two miners find respectively blocks b and b' nearly at the same time, then both b and b' will propagate, resulting in a network partition. A partition P that appended b to their copy of the blockchain and another P' that appended b'.

k-safety The parameter k is the depth at which the probability of a fork or ledger conflict is sufficiently low to consider that the risk of this block being abandoned is null. The parameter k depends on the blockchain you are using and the level of safety your system requires. A block that has been appended to the chain and that is now at a depth equal to or greater than k. Each new block appended to the chain after the block of interest is called a confirmation. A transaction in a k-valid block is said to be k-valid. It is said to have received k confirmations.

Simple Payment Verification (SPV) Simple payment verification is a process that allows a client to verify the validity of a transaction without having to maintain a full copy of the blockchain. Instead, the client only needs the header of the block containing the transaction as well as the Merkle proof of the said transaction.

Block time The block time is the average block creation time. It is calculated by dividing a large period of time by the number of blocks produced during this period. This parameter is

specific to each blockchain.

Transaction validation time Let process p broadcast a transaction tx via the primitive BROADCAST$(< transaction >, tx)$ at time $t_{broadcast}$. Let t_{valid} be the time when the transaction tx appears to be k-valid to p. Then we define the transaction validation time as the difference between t_{valid} and $t_{broadcast}$.

3.2.3 Asset model

Blockchain asset As of today, blockchain asset refers to anything on the blockchain that serves as a store of value or medium of exchange. It can be coins such as bitcoins and ethers, or it can be tokens such as ERC-20 tokens. Nevertheless, blockchain assets share the following properties; A blockchain asset belongs to a public/private key pair. Asset ownership can be transferred through transactions. Asset transfers are recorded to the chain. To transfer the ownership of an asset, one must sign the transaction with the private key.

Asset locking An asset on a blockchain is said to be locked when it is not possible to transact it without meeting some conditions. Accessing the said conditions would unlock the asset and allow the user to transact it. Assets are locked thanks to scripts or smart contracts. If the condition to unlock the asset is the expiration of a time-lock, we talk about time locking. If it is the revelation of a hash pre-image, it is hash-locking.

In this section, we define the time-locked atomic cross-chain swap problem.

3.2.4 Problem specification

A time-locked atomic cross-chain swap protocol uses time locking to ensure the termination of the protocol. At the expiration of the time lock, both participants have received an acceptable payoff.

Definition 1 (time-locked atomic cross-chain swap) A time-locked atomic cross-chain swap protocol should satisfy the following properties:

- **Safety.** No participant abiding by the protocol can lose more money than the transaction fees.

- **Time-bounded termination.** No asset can be locked for more than a period of time γ and if an asset is locked, γ is known before locking.
- **Liveness.** Upon lock time expiration, if all participants abided by the protocol and no failures occurred then they received their payoffs.

3.2.5 Relays

A blockchain relay $R_{a\leftarrow b}$ is an abstraction (smart contract or script) on chain BC_a that can receive verification requests of transactions on-chain BC_b. It receives block headers of chain BC_b and performs the standard verification for blocks of BC_b (1). It stores block headers of chain BC_b (2). It can perform Simplified Payment Verification over transactions on-chain BC_b and either returns true if the transaction is valid or false if it is not (3).

Definition 2 (Blockchain Relay) A blockchain relay must satisfy the following properties:

- **k-Validity.** A blockchain relay returns true if the submitted transaction is a k-valid transaction of blockchain BC_b, otherwise, it returns false.
- **Eventual Persistent Storage.** Every valid block header of chain BC_b eventually ends up being included in a block appended to BC_a.

A blockchain relay hosted on BC_a that performs verification over blockchain BC_b is noted $R_{a\leftarrow b}$.

Algorithm 5 is a possible interface for a blockchain relay. It presents the main functions of such software. The function VALIDATEPOW (line 3) implements the Proof of Work verification algorithm of the relayed blockchain. It returns true if the provided block header is valid and false otherwise. The function VALIDATESPV (line 7) implements the simple payment verification of provided transaction. It returns true if it is a valid transaction and false otherwise. The parameter k in function VERIFYTX(tx, k) is the safety factor. It specifies at what height the block containing the transaction tx must be in the relay's data store to consider the transaction as valid. Complete implementation of VALIDATEPOW and VALIDATESPV can be found in [43].

Algorithm 5 Blockchain Relay smart contract class

```
1: bytes [] blocks                                   ▷ Table to store block headers
2: procedure STOREBLOCKHEADER(blockHeader)
3:     requires(VALIDATEPOW(blockHeader))
4:     THIS.BLOCKS.APPEND(blockHeader)               ▷ If the block is valid, store block
5: end procedure
6: procedure VERIFYTX(tx, k)
7:     return VALIDATESPV(tx.txid, tx.root, tx.proof, tx.index, k)
8: end procedure
```

Blockchain Relay latency We define blockchain relay latency Δ_{relay} as the time difference between the moment a new block b_i is mined on chain BC_b and the moment a verification request for $tx_i \subset b_i$ to the relay $R_{a \leftarrow b}$ with safety parameter k will return true. Let t_{mined} be the time at which a relayer process p of blockchain BC_b delivers a block b_i via the primitive DELIVER. Let t_{valid} be the time when VERIFYTX(tx_i, k) will return true. Then the blockchain relay latency is given by:

$$\Delta_{relay} = t_{valid} - t_{mined} \tag{3.1}$$

3.2.6 Adapters

A Blockchain adapter is a piece of software that allows sending transactions and querying several blockchain systems. In addition to its multiple-blockchain client capabilities, it can perform off-chain computation and storage. It can store private information such as private keys and credentials [77]. A blockchain adapter can be hosted by several instances. The system's goals of a blockchain adapter are:

- Client capabilities: A blockchain adapter should be able to read the state of at least two blockchains, as well as broadcasting transactions.
- Off-chain computation: A blockchain adapter should be able to perform off-chain computations, such as generating a random value and computing its hash.
- Wallet safety: A blockchain adapter should be able to store private credentials and keys, without the person running it being able to extract those private data.
- Decentralization: A blockchain adapter should be replicated over several nodes

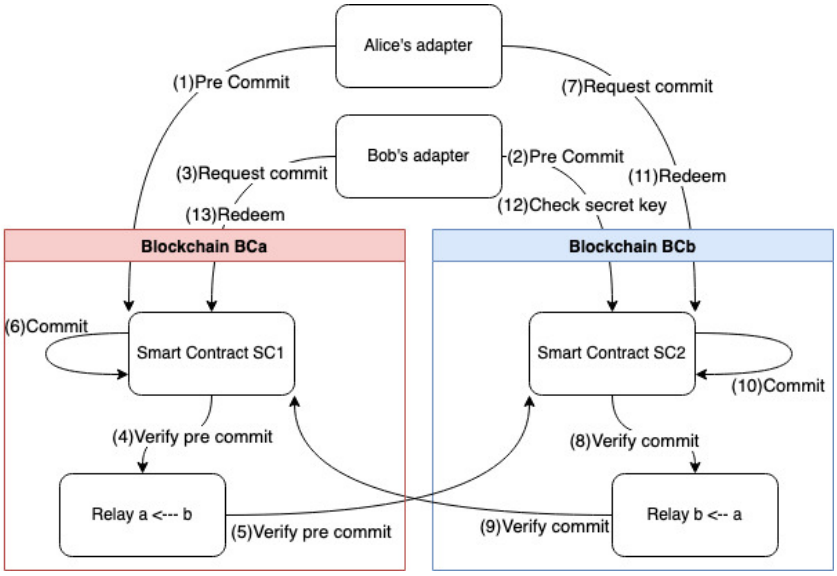


Figure 3.17: High level overview of the R-SWAP components and interactions

3.2.7 R-SWAP hash time locked contract

msg is a variable generated when executing a transaction. It contains information relative to the transaction such as the address of the function caller $msg.sender$ or the value of the transaction $msg.value$. A smart contract is presented in Algorithm 6.

3.2.8 The R-Swap protocol

In this section, we detail our R-SWAP protocol and prove its correctness with respect to the time-locked atomic cross-chain swap specification.

3.2.8.1 Protocol Overview

The R-SWAP protocol relies on three building blocks; hash time-locked contracts (see Algorithm 6), blockchain relays (see Algorithm 5), and blockchain adapters. Each adapter can send transactions on both blockchain BC_a and BC_b. The unfolding of R-SWAP resembles a hash time-locked contract (HTLC) but it makes use of relays for cross-chain transaction verification and adapters for automation.

In the first phase, participants will commit to the swap by locking their assets with smart contracts. In a second phase, participants will receive their payoffs or refunds.

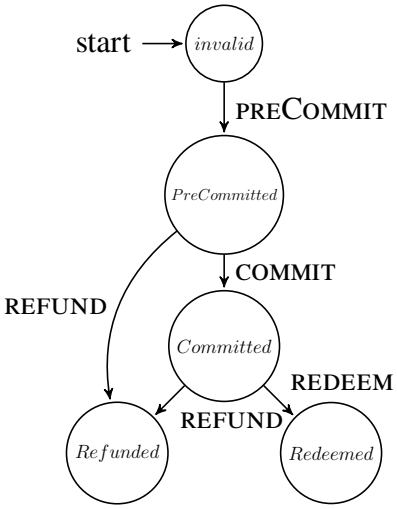
Algorithm 6 R-SWAP hash time lock contract class

address $sender$; ▷ Address of swap initiator
address $receiver$; ▷ Address of participant
float $amount$; ▷ Amount exchanged
float $rate$; ▷ Exchange rate
integer $timelock$;
bytes $secret$;
bytes $hashLock$; ▷ Hash of the secret
address $Relay$; ▷ Address of the relay smart contract
enum State{Invalid, PreCommitted, Committed, Redeemed, Refunded}
State $state \leftarrow Invalid$ ▷ Variable to store state
procedure PRECOMMIT($hashLock$, $amount$, $rate$)
    requires($this.state = Invalid$)
    $this.hashLock \leftarrow hashLock$
    $this.amount \leftarrow amount$
    $this.rate \leftarrow rate$
    $this.sender \leftarrow msg.sender$
    $this.state \leftarrow PreCommitted$
end procedure
procedure COMMIT(tx)
    requires($this.state = PreCommitted$)
    requires(RELAY.VERIFYTX(tx)) ▷ Calling the relay to verify that the provided transaction is valid
    requires($tx.hashLock = this.hashLock$) ▷ Verifying the hash lock
    requires($tx.amount = this.amount * this.rate$) ▷ Verifying the amount
    $this.receiver \leftarrow msg.sender$ ▷ Setting the function caller as receiver
    $this.timelock \leftarrow now + \Delta_1$ ▷ Setting up the time lock
    $this.state \leftarrow Committed$
end procedure
procedure REDEEM($secret$)
    requires($this.state = Committed$)
    requires($msg.sender = this.receiver$) ▷ Check that the caller is the swap receiver
    requires(SHA256($secret$)$= this.hashLock$) ▷ Verify the secret
    $this.secret \leftarrow secret$ ▷ Make the secret public
    TRANSFER($this.amount$, $this.receiver$) ▷ Transfer the funds
    $this.state \leftarrow Redeemed$
end procedure
procedure REFUND
    requires($this.state \neq Redeemed$)
    requires($msg.sender = this.sender$) ▷ Verify that the caller is the swap sender
    requires($this.timelock \geq now$) ▷ Check that timelock is elapsed
    TRANSFER($this.amount$, $this.sender$) ▷ Transfer the funds
    $this.state \leftarrow Refunded$
end procedure
procedure CHECKSTATE
    return $this.state$
end procedure
procedure CHECKSECRETKEY
    return $this.secret$
end procedure

3.2.8.2 Phase 1: Commitment phase

When Alice wants to execute an atomic cross-chain swap with the R-SWAP protocol, she will request her adapter to lock up funds inside a hash time locked contract SC_1. This contract SC_1 exists in one of five states: *Invalid*, *PreCommitted*, *Committed*, *Redeemed* and *Refunded*. State changes and the corresponding asset transfers are triggered by the following functions: PRECOMMIT, COMMIT, REDEEM, REFUND. State changes are unidirectional.

Figure 3.18: Representation of the R-SWAP smart contract states as a Directed Acyclic Graph



Alice only specifies to the adapter the exchange rate and the amount. The adapter will be responsible for generating a random secret s and computing its hash $h = H(s)$. Then the adapter will lock up the funds inside SC_1 hosted on chain BC_a by calling the PRECOMMIT function.

At this point, the hash lock has been specified, but not the time lock. Hence the assets are not locked, because Alice can ask for a refund at any moment. The PRECOMMIT state serves as proof of ownership.

Bob, a participant interested in the swap, would just have to retrieve the parameters from the chain and request his adaptor to set up a mirror contract SC_2 on blockchain BC_b.

Once the funds are hash locked in SC_2 trough PRECOMMIT function, Bob will request that the contract SC_1 sets him as the receiver and define a time lock. To do so, Bob's adapter will call the COMMIT function of SC_1, with as parameter, the transaction hash of SC_2's PRECOMMIT. Indeed before setting Bob as receiving party, SC_1 needs the proof that some funds have been locked up in parallel on BC_b. SC_1 will call the *verifyTx* function of the relay $R_{a \leftarrow b}$, with as a parameter, the PRECOMMIT transaction of SC_2 provided by Bob's adaptor. If the transaction is

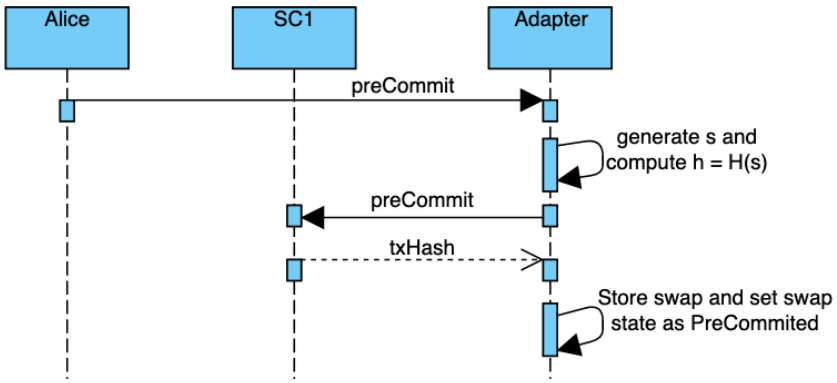


Figure 3.19: Sequence diagram of the precommit of the initiator of the swap

valid, $R_{a \leftarrow b}$ will return the parameters of the transaction.

Here the relay $R_{a \leftarrow b}$ serves as a bridge between BC_a and BC_b. It allows contract auditing to be made directly on-chain by SC_1 and SC_2. The authenticity of the data is ensured and hence the safety of the protocol is improved. Once SC_1 has verified SC_2's parameters, it can safely set Bob as the receiver, Δ_1 as time lock, and change state to *Committed*. SC_1 sends back the transaction details to Bob's adapter, which redirects it to SC_2. SC_2 calls $R_{b \leftarrow a}$ to verify SC_1's commitment, and if so commits with Alice as a receiver and Δ_2 as time lock.

Here is a sum-up of the commitment phase.

1. Alice calls her adapter's PRECOMMIT function with target network, value and exchange rate
2. Alices adapter generates secret s and computes $h = H(s)$
3. Alices adapter calls SC_1 PRECOMMIT function
4. Alices adapter runs a daemon
5. Bob calls the PRECOMMIT function of his adapter with Alices mirror parameters
6. Bobs adapter calls SC_2 PRECOMMIT function
7. Bobs adapter calls SC_1 COMMIT function
8. SC_1 calls the relay $R_{a \leftarrow b}$ to verify that SC_2 is in state *PreCommitted*
9. On receiving the proof that SC_2 is in state *PreCommitted* SC_1 changes its state to *Committed* sets Bob as receiver and Δ_1 as time lock
10. On receiving proof that SC_1 is *Committed*, Bobs adapter calls SC_2 COMMIT function.
11. SC_2 calls the relay $R_{b \leftarrow a}$ to verify that SC_1 is in *Committed* state
12. On receiving the proof that SC_1 is on *Committed* state, SC_2 changes its state to *Committed* sets Alice as receiver and Δ_2 as time lock
13. Bobs adapter runs a daemon

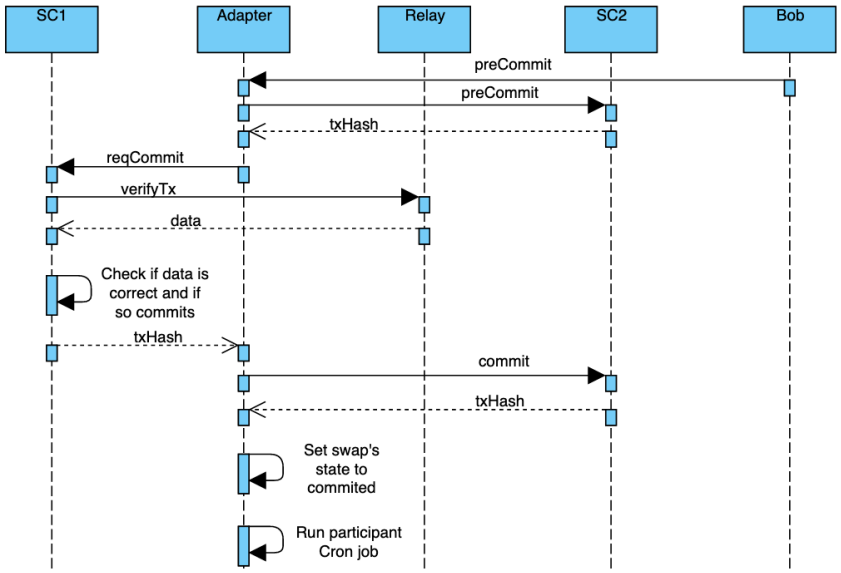


Figure 3.20: Sequence diagram of the precommit of the participant and both commits

3.2.8.3 Phase 2: Contracts redeem/refund

Daemon At stage 4. and 13. each adapters have run a daemon. Those daemons are awaiting a state change of SC_2 before they move on to the redeem/refund phase. For each new block of BC_b Alice's daemon will check if Bob's contract SC_2 has changed from state *PreCommitted* to *Committed*. This would mean that Bob has committed to the swap and thus that she can move on to the redeem phase. For each new block of the blockchain, BC_b Bob's daemon will check if SC_2 has changed from state *Committed* to *Redeemed*. This would mean that Alice has redeemed, and by doing so, revealed the secret key. Bob can now move on to the redeem phase.

Those daemons also handle the cases where some party would stop abiding by the protocol. They have stored the time lock values and a REFUND transaction call is scheduled at the expiration of the time lock.

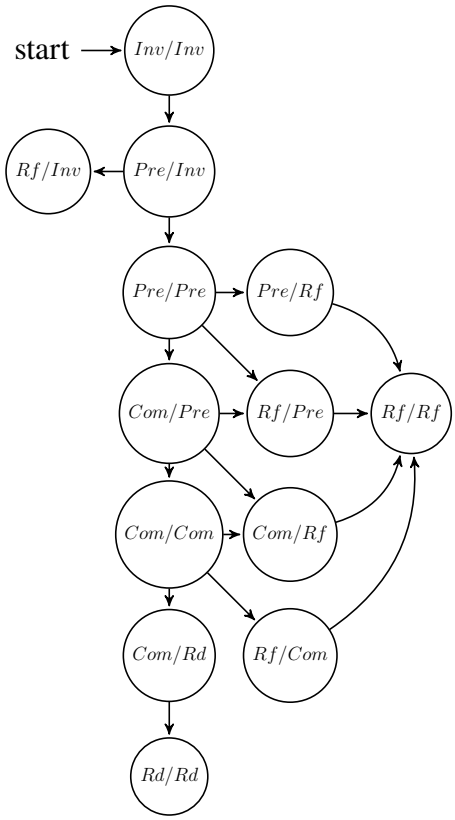
Here is a sum-up of the second phase, if participants abide by the protocol:

1. Alices adapter calls SC_1 CHECKSTATE function
2. If state is *Committed* call SC_2 CHECKSTATE function
3. If state is *Committed* Alice's adapter calls REDEEM function of SC_2 with s, triggering the asset transfer to her address
4. Bobs adapter calls SC_2 CHECKSECRETKEY function
5. If key is revealed Bob's adapter calls SC_1's REDEEM function with the secret s he just learned, triggering the asset transfer to his address

If one of the participants does not abide by the protocol, the daemon will simply call REFUND

at the expiration of their respective smart contract's time lock expiration.

Figure 3.21: Representation of the R-SWAP protocol possible set of states as a Directed Acyclic Graph



3.2.9 R-SWAP correctness

In this section, we prove the correctness of the protocol, i.e., that it satisfies the properties of safety, time-bounded termination, and liveness of a time-locked atomic cross-chain swap, defined in Section 3.2.4. But first, we will try to determine the time lock values that would guarantee the properties of the protocol.

3.2.9.1 Time lock value determination

Choosing the right value for the time locks is essential. A time lock value that is too short could lead to a safety violation. Conversely, a time lock value that is too long could be commercially unfair, as prices are very volatile.

Consider a swap executed via the relays swap protocol. The swap is executed between blockchains BC_a and BC_b. k_a resp. k_b is the k parameter of BC_a resp. BC_b (see k-safety in

Section 3.2.2). ζ_a is the target block time of BC_a and ζ_b of BC_b. There are two time locks values to be chosen; Δ_1 for SC_1 and Δ_2 for SC_2.

The first value of interest is the difference between Δ_1 and Δ_2. This difference has to be long enough for the participant to call the redeem function and for this transaction to be included in a valid block. If not long enough this could lead to a safety violation. Δ_1 and Δ_2 are strongly dependent on parameters specific to each blockchain. A naive estimation of the minimal value for the difference between time locks could be:

$$\Delta_1 - \Delta_2 = (\zeta_b k_b) \tag{3.2}$$

Assuming synchronous communication, this should be long enough for the redeem transaction to be included in a valid block. But as stated in the model, the block time is not necessarily fixed and blocks can be delayed for an arbitrarily long time, leading to potential safety violations.

Therefore for the rest of this section, we will assume that there is a time λ, specific to each blockchain, within which a broadcast transaction will be included in a block and receive k confirmations with a probability ϵ. Thus $\Delta_1 - \Delta_2 \geq \lambda_b$ is the first constraint to satisfy the safety property, where λ_b is the upper bound on transaction validation for blockchain BC_b.

The second constraint does not concern safety, but commercial fairness. Bob will not call the COMMIT function of SC_2 unless he is sure that SC_1 is *Committed*. He needs the guarantee that Alice's commitment is a valid transaction. If it is not yet, in the fear of a safety violation, Bob should choose not to COMMIT and instead call REFUND on SC_2. Alice would end up having her assets locked for at least λ_b, which is commercially unfair for her. Thus, for the swap to be fair, time-locks should be chosen such that $\Delta_1 \geq \Delta_2 + \lambda_a$.

We obtain the following:

$$\Delta_1 \geq \Delta_2 + max\{\lambda_a, \lambda_b\} \tag{3.3}$$

Now that we found the minimal value for Δ_1, we want to find a minimal value for Δ_2. We want to minimize the duration of the swap for commercial fairness. If $\Delta_2 = 0$, Bob can call for a REFUND on SC_2 at any moment. Yet he cannot ask for REDEEM on SC_1 either he doesn't know s. Alice's adapter has been triggered by Bob's COMMIT and proceeds to call REDEEM on SC_2. This REDEEM transaction contains s in plain text as a parameter. At this point this redeem transaction is not confirmed yet, but still waiting in the mempool. As transactions are

public, Bob can run a transaction sniffer that would extract s from Alice's unconfirmed REDEEM transaction. Then he would send two transaction: REFUND on SC_2 and REDEEM on SC_1 with the secret s he just sniffed, leading to a safety violation. He could even increase its chances of success by setting high transaction fees.

Therefore it is necessary that Δ_2 be long enough for Alice to call REDEEM on SC_2 and for this transaction to be included in a valid block. Thus we have:

$$\Delta_2 \geq \lambda_b \tag{3.4}$$

Relay latency Cross chain transaction verification requires that, for each new block b_i produced on chain BC_b, a relayer submits b_i to the relay $R_{a \leftarrow b}$. Thus, between the time a block b_i is produced on chain BC_b and the time VERIFYTX(tx), $tx \subset b_i$ to the relay $R_{a \leftarrow b}$ will return true, there is a maximum delay given by [98].

$$\Delta_{relay} = \lambda_b + \Delta_{submit} + 2\lambda_a \tag{3.5}$$

λ_a and λ_b being the upper bound on transaction validation time of blockchain BC_a resp. BC_b. Δ_{submit} is the delay between the moment a block is produced and the moment a relayer submits the block to the relay.

In the R-SWAP protocol, each relay $R_{a \leftarrow b}$ and $R_{b \leftarrow a}$ is called once. SC_1 calls VERIFYTX(tx, k) at stage 8. to verify that SC_2 is in state *PreCommitted*. SC_2 calls the relay $R_{b \leftarrow a}$ at stage 11. to verify that SC_1 is in state *Committed*.

Let $tx1_{com}$ be the commitment transaction of SC_1. There is a latency Δ_{relay} during which the relay $R_{b \leftarrow a}$ will not consider $tx1_{com}$ as a valid transaction. Thus calling VERIFYTX$(tx1_{com}, k)$ on $R_{b \leftarrow a}$ during this period of time might return false. Thus the time lock value Δ_1 of contract SC_1, must be such that $\Delta_1 \geq \Delta_{relay}$. Considering relay's delay we must now ensure that Δ_1 is large enough for the commit transaction of SC_1 to be valid regarding the relay:

$$\Delta_1 \geq \Delta_2 + \Delta_{relay} \tag{3.6}$$

or

$$\Delta_1 \geq \Delta_2 + \lambda_a + \Delta_{submit} + 2\lambda_b \tag{3.7}$$

Finally we obtain the following system of inequalities for the time lock values:

$$\begin{cases} \Delta_1 \geq \Delta_2 + \lambda_a + \Delta_{submit} + 2\lambda_b \\ \Delta_2 \geq \lambda_b \end{cases} \tag{3.8}$$

3.2.9.2 Proof-of-correctness

Lemma 3.1

R-SWAP protocol satisfies the time-bounded termination property of a time-locked atomic cross-chain swap. ♡

Proof In the following, we prove that there exists a bounded value $\gamma > 0$ such that the R-SWAP protocol verifies the time-bounded termination property with parameter γ. Suppose R-SWAP protocol violates the time-bounded termination property. A violation of the time-bounded termination property implies that an asset is locked forever, thus excluding all set of states before *PreCommitted/PreCommitted*, *PreCommitted/PreCommitted* included. Indeed, before being in the state *Committed*, contracts are only hash-locked, not time-locked and thus can be refunded. Given the possible set of states, subsequent possible states are *Committed/PreCommitted*, *PreCommitted/Refunded* and *Refunded/PreCommitted*. *PreCommitted/Refunded* and *Refunded/PreCommitted* don't imply a violation of the time-bounded termination property since assets are refunded or can be refunded, so the only case to consider is the set *Committed/PreCommitted*.

To transition to this set, SC_1 must commit. It does so by calling the COMMIT function with as a parameter Δ_1. Δ_1's value is hardcoded in the contract (see Appendix 3.2.9.1), thus it is known and finite. The result of the SC_1's commitment is a transaction tx.

Subsequently, three sets of state are possible:

- *Refunded/PreCommitted*: Since Δ_1 is known and finite, SC_1 will eventually be able to move to $Refunded$, not leading to a violation of the time bounded termination property. In this case the asset cannot be locked for more than $\gamma = \Delta_1 + \lambda_a$.
- *Committed/Committed*: To proceed to state $Committed$, SC_2 calls $R_{b \leftarrow a}$ to verify tx. If the relay returns true then SC_2 extracts Δ_1 from tx. It then proceeds to call it's own COMMIT function with as a parameter $\Delta_2 = \lambda_b$ (see Appendix 3.2.9.1). Yet a violation of the time bounded termination property would imply that the time lock value was unknown, leading to a contradiction.

- *Committed/Refunded*: Since its asset was not locked, SC_2 can be refunded. But since Δ_1 is known and finite, $SC1$ will eventually be able to move to $Refunded$, not leading to a violation of the time-bounded termination property.

Out of those three sets of states, the only one of interest is *Committed/Committed*, as the other two involve SC_1 being able to call REFUND. Since Δ_1 is known and bounded, SC_1 will be able to call REFUND at the expiration of Δ_1. Thus, from now on, the potential violation of the time-bounded termination property only concerns SC_2. From the set of state *Committed/Committed* two set of states are possible:

- *Committed/Redeemed*: Since Bob has extracted h from Alice's COMMIT transaction tx, she can call the REDEEM function of SC_2 with the secret s. By doing so she triggers the asset transfer from SC_2 to her address. As she has revealed s to Bob, if Bob's adapter is correct it will automatically call the REDEEM function of SC_1, triggering the asset transfer from SC_1 to Bob's address. Since the transaction has a high probability of being included before Δ_2's expiration, the protocol satisfies the time-bounded termination property.
- *Refunded/Committed*: If Alice has been refunded, it means that Δ_1 has expired. If Bob's adapter is correct, it would have called the REFUND function of SC_2 right after the publication of the block containing Alice's refund transaction. Since $\Delta_1 \geq \Delta_2 + \Delta_{relay}$, Bob's REFUND transaction will be included with a high probability, leading to a contradiction.

Overall, an asset can remain locked at most $\gamma = \max(\Delta_1 + \lambda_a, \Delta_1 + \lambda_a)$ time. Thus the R-SWAP protocol satisfies the time-bounded termination property with parameter γ.

Lemma 3.2

The R-SWAP protocol satisfies the safety property of a time-locked atomic cross-chain swap. ♡

Proof Assume R-SWAP protocol violates the safety property of the time-locked atomic cross-chain swap specification. This safety violation implies that there exist two smart contracts SC_1 and SC_2 one being in state *Redeemed* and the other *Refunded*. Indeed, any other set of states is impossible or doesn't imply a safety violation. We also exclude the case of an asset being locked forever because of the time-bounded termination property.

SC_1 being redeemed and SC_2 refunded, imply that Bob has found the secret s because SC_1 is programmed to change state to redeemed only if provided s. This leads to a contradiction because it is impossible to calculate s from h.

SC_1 being redeemed and SC_2 being refunded imply that Alice has called the redeem function of SC_2. By doing so she has revealed the secret s. But given that the adapter checks the state of SC_2 at every new block, and given that the block time is $<< \Delta_2$ (see Appendix 3.2.9.1), at the time Alice will be allowed to call refund, SC_1 will already be in state *Redeemed*, leading to a contradiction.

Lemma 3.3

The R-SWAP protocol satisfies the liveness property of the time-locked atomic cross-chain swap problem with high probability. ♡

Proof Assume R-SWAP protocol violates the liveness property of the time-locked atomic cross-chain swap problem. A violation of the liveness property implies that upon lock time expiration both contracts are in state *Redeemed* but one or more of the participants have not received their payoffs. Since SC_1 and SC_2 are programmed to transfer the assets before the transition to state *Redeemed*, a violation of the liveness property implies that the receiver address provided during the commit transaction $tx1_{com}$ of SC_1 resp. $tx2_{com}$ of SC_2 was not Bob resp. Alice.

When calling SC_1 COMMIT function, Bob's adapter has provided the details of the PRE-COMMIT transaction $tx2_{pre}$ of SC_2. Then the COMMIT function of SC_1 has called $R_{a \leftarrow b}$ to verify $tx2_{pre}$. If the transaction is valid, SC_1 proceed to extract Bob's address from $tx2_{pre}$, set him as receiver and changes its state to *Committed*. Yet a violation of the liveness property would suppose that this address was something else, leading to a contradiction.

SC_2 COMMIT function works the same way, but instead of providing the PRECOMMIT transaction, Alice's adapter provides the COMMIT transaction details $tx1_{com}$. Prior to committing, SC_2 calls $R_{b \leftarrow a}$ to verify $tx1_{com}$. If it was a valid transaction, SC_2 extracts Alice's address from $tx1_{com}$ and calls its own COMMIT function with the address previously extracted as a parameter. Yet a violation of the liveness property would suppose that the address provided was something else, leading to a contradiction.

Note on non-deterministic blockchains It should be noted that in proof-of-work blockchain systems such as Bitcoin and Ethereum, block creation is probabilistic. However, it is proven in [36] that under the assumption of an honest majority of nodes $t \leq \frac{1-\delta}{2}(n-t)$, Bitcoin satisfies agreement and validity properties with a probability of at least $1 - e^{-\Omega(\epsilon^2 \lambda f)}$ with n number of parties mining; t out of which are controlled by the adversary, δ the advantage of

honest parties, ϵ the quality of concentration of random variables in typical executions, λ the tail-bounds parameter and f the probability at least one honest party succeeds in finding a POW in a round. Thus, for such blockchain systems, R-SWAP satisfies the properties of a time-locked atomic cross-chain swap with high probability.

3.2.10 Performance and cost analysis

In this Section, we will analytically evaluate the performances of the R-SWAP protocol. We will then provide a numerical projection for the most valued PoW blockchains, Bitcoin and Ethereum. We will also provide a numerical projection for the most valued BFT consensus blockchain, Cosmos. Finally, we will estimate the operational cost of maintaining the infrastructure required.

3.2.10.1 Theoretical analysis of R-SWAP performance

In this section, we will analytically evaluate the performances of the R-SWAP protocol.

Atomic Swap Latency definition There is no reliable global clock in most blockchain systems thus latency can only be measured from the participants' blockchain client's perspective.

Definition 3 (Atomic Swap Latency) Let $Locked$, $Refund$ and $Redeem$ be three predicates. Let p be a participant in an atomic cross-chain swap. Predicate $Locked = true$ indicates that p's asset is locked. Predicate $Refund = true$ indicates that p has received his refund and can transact it. Predicate $Redeem = true$ indicates that p has received his payoff and can transact it. Let t_l be the time when $Locked = true$. Let t_u be the time when $Refund \vee Redeem = true$. Then the atomic swap latency is defined by:

$$\Delta_{latency} = t_u - t_l \tag{3.9}$$

R-SWAP Latency Let λ be the upper bound on transaction validation and let Δ be the time lock value of an R-SWAP contract SC. As proven in Section 3.2.9, the R-SWAP protocol is time-bounded. In the case every participant is honest and abides by the protocol, the atomic swap latency of the R-SWAP protocol is less than Δ_1, the lock time of contract SC_1. In the case

Bob stopped abiding by the protocol after Alice's COMMIT transaction, she will have to wait for Δ_1's expiration before calling REFUND. Then this REFUND transaction will be confirmed in less than λ_a, the transaction validation time of blockchain BC_a. From Alice's point of view, this is a latency of less than $\Delta_1 + \lambda_a$. In the case Alice stopped abiding by the protocol after Bob's COMMIT transaction, he will have to wait for the expiration of Δ_2 before calling REFUND on SC_2. This gives a latency of $\Delta_2 + \lambda_b$ for Bob and $\Delta_1 + \lambda_a$ for Alice. Thus the overall maximum latency is $\Delta_{lantency} = 3\lambda_b + \lambda_a$.

Case \ Max latency	Alice	Bob
Alice & Bob abide by the protocol	Δ_1	Δ_1
Bob stops after Alice's Commit	$\Delta_1 + \lambda_a$	None
Alice stops after Bob's Commit	$\Delta_1 + \lambda_a$	$\Delta_2 + \lambda_b$

Figure 3.22: Table of R-SWAP theoretical latency

3.2.10.2 Numerical analysis of R-SWAP performance

The value of the latency is highly dependent on the set of blockchains involved in the swap because time locks are based on the upper bound of transaction validation time λ with probability ϵ, which is specific to each blockchain. In this section we will provide a numerical analysis of the swap latency between the two most valued cryptocurrencies in terms of market cap at the time of writing: Bitcoin and Ethereum [23].

Bitcoin's upper bound on transaction validation time Calculating Bitcoin's or any blockchain's upper bound on transaction validation time is a very complex problem. Indeed, it depends on numerous factors such as transaction fees, network traffic, current hash rate and difficulty, size of the unconfirmed transaction mempool, etc. But it has been shown in [50] that the main factors are transaction fee density (in satoshi/Byte) and network traffic.

It is known that miners order transactions in their mempool by fee density to maximize profitability [47]. By setting a sufficiently high fee density it is possible to predict that a transaction will be included in the next block with a probability p as high as 95% [69]. Then to find an estimation of Bitcoin's upper bound on transaction validation time, we must ensure that once the transaction has been included, there is enough time for $k = 6$ new blocks to be appended to the chain, i.e., to receive $k = 6$ confirmations. Block time in Bitcoin follows an exponential distribution with parameter $\theta = 0.001578$ [27]. Thus the probability of k blocks

being mined in less than x time is given by the PDF of the gamma distribution Gamma(k,θ) cumulative function.

As most blockchain clients consider $k = 6$ being a safe number of confirmations we obtain the value of Figure 3.23.

$P(X < x)$	x (in seconds)
0.95	6662.25
0.99	8307.02
0.995	8966.89
0.999	10427.60

Figure 3.23: Probability for 6 blocks to be mined in less than x seconds

Thus with a fee density high enough for the transaction to be included in the next block with a high probability $p = 0.95$, there is an upper bound on this transaction validation time $\lambda_{btc} = 10427$ seconds with a high probability. [1] Rounded up, this translates to a 3 hours validation time.

Ethereum's upper bound on transaction validation time A study based on the Ethereum blockchain data [82] showed that 99% of the blocks were produced in less than a minute. Thus with $k = 12$, twelve minutes after its inclusion in a block, a transaction should be k-valid with a high probability $p \approx 1$. Therefore with sufficient gas price, there is an upper bound on transaction validation time for Ethereum $\lambda_{eth} = (k * 60) + 30 = 750$ seconds with high probability. [2]

Latency for a swap between Bitcoin and Ethereum Consider an R-SWAP protocol execution between Bob and Alice. Alice, the initiator, has bitcoins. Bob, the participant, has ethers. (The initiator is the one that generates the secret). Given the values of upper bounds on transaction validation time, given the theoretical latency values of Figure 3.22 and given the values of time locks of Section 3.2.9.1, the estimation of the maximum latency values for this swap are presented in Figure 3.24a. (We selected $\Delta_{btc} = 8307$ seconds and $\Delta_{eth} = 750$ seconds).

Tendermint upper bound on transaction validation time Block-chain systems such as Cosmos [52] uses the Tendermint Byzantine fault-tolerant (BFT) consensus [12]. Tendermint

[1] $p = 0.95 * 0.999 = 0.949$. It is to be noted that even if the transaction has not received 6 confirmations yet, it should have received at least 5 confirmations with probability > 0.949, and should receive its 6th confirmation in the next ten minutes

[2] This is a 12min and 30s total validation time.

Case \ Max latency	Alice	Bob
All abide	10 557	10 557
Bob stops	18 864	n.a
Alice stops	18 864	1500

(a) Latency for a R-SWAP execution between Bitcoin and Ethereum

Case \ Max latency	Alice	Bob
All abide	771	771
Bob stops	1521	n.a
Alice stops	1521	14

(b) Latency for a R-SWAP execution between Ethereum and Tendermint

BFT consensus satisfies the instant finality property. With our model, this translates to having a safety factor $k = 1$. Thus, the Tendermint upper bound on transaction validation time is nothing else than the Tendermint block time. As of today, the observed block time is seven seconds [73]. Thus for Tendermint we have $\lambda_{tendermint} = 7$.

Latency for a swap between Tendermint and Ethereum Consider an R-SWAP protocol execution between Tendermint and Ethereum. The initiator (the one generating the hash lock) has ethers and the participant has atoms (Cosmos native asset). For such a swap we obtain the latency values presented in Figure 3.24b. Cosmos recently launched the Stargate update [11] which allows cross-chain transactions. The inter blockchain communication protocol used to achieve cross-chain transactions uses some sort of pegging like X-Claim [98]. As this is a work in progress, we do not have a performance evaluation of this protocol.

3.2.10.3 Cost analysis

In this section, we will analyze the operational cost of the infrastructure required to implement the R-SWAP protocol. The R-SWAP protocol is made out of three software bricks; hash time-locked contracts, blockchain adapters, and relays. Operating an external adapter is as costly as operating a blockchain node. A single hash time-locked contract can be used for a large number of swaps, making the deployment cost negligible. Thus the operational costs of the infrastructure narrow down to the cost of operating the relays.

Operational cost of Relays As explained in Section 3.2.5, a relay $R_{a \leftarrow b}$ is a piece of software, possibly a smart contract, that allows to read and verify the state of chain BC_b from chain BC_a[98]. The intuition is that a relayer r publishes every new block header of chain BC_b to the relay $R_{a \leftarrow b}$ on chain BC_a. The relay implements the block verification protocol of chain BC_b. It verifies the proof of work for PoW chains or the signature of $2/3$ of the validators for BFT consensus chains [35]. Then, once the block has been verified, a process can call the relay to verify a specific transaction of chain BC_b from BC_a. He pays a little fee that will compensate the relayer's work and expenses. Depending on the blockchain, this process can be both calculation and storage-intensive. Unfortunately, storage and calculation "on-chain" is expensive in most blockchain systems, inevitably leading to high operational costs. Current implementations such as BTCRelay [26] used about 194 000 gas to store and verify a single Bitcoin block header [3]. Gas price and ether price are subject to high volatility. Thus it is important to note that what follows can vary from one day to another but to give a rough estimate, as of today's prices [4], this would translate to 25$ per relayed block. Considering that an average of 144 Bitcoin blocks are mined every day, maintaining the BTCRelay would cost alone about 3600$/day. If the volume of R-SWAP transactions is large enough, those operational costs could be amortized by an economy of scale, each swap participant paying a small fee to the relayer. The other outlays concerning the relays is the computational power required to verify a submitted transaction. For each R-SWAP execution, each participant request at least one transaction verification to the relay. On modern implementation of BTCRealy, the cost of such an operation vary from 67000 to 102000 gas [43].

As of today, the operating cost of the relay is quite dissuasive, especially considering that current centralized exchange platforms offer fees as low as 0.1% [5]. Fortunately, recent research suggests that the operational cost of blockchain relays could be reduced by up to 92% [35], thanks to an "on-demand" approach.

3.2.11 Conclusion on R-SWAP

This section presents R-SWAP, a time-bounded atomic cross-chain swap protocol that makes use of blockchain relays and adapters to address the shortcomings of the hash locking

[3] https://etherscan.io/address/0x41f274c0023f83391de4e0733c609df5a124c3d4

[4] eth price: 1293$, gas price: 100Gwei

[5] https://www.binance.com/en/fee/schedule

technique. Previously proposed atomic cross-chain swap protocols such as [40] were subject to a potential safety violation. Indeed, the safety of such protocols could only be satisfied with the assumption that no errors occurred on the participant's blockchain client. Moreover, the smart contract auditing tasks were to be executed by the participants, leading to bad ergonomics and potential safety violations. The R-SWAP protocol makes use of blockchain relay $R_{a \leftarrow b}$ to verify transactions of chain BC_b from chain BC_a thus automating the contract auditing process. The protocol also involves distributed blockchain APIs, namely blockchain adapters that reduce the risks of a client crash compromising the swap's safety.

We formally prove that the R-SWAP protocol satisfies the properties of a time-bounded atomic cross-chain swap with a maximum latency $\Delta_{latency} = 3\lambda_b + \lambda_a$, where λ's represents a specific blockchain's upper bound on transaction validation time.

The operational cost of the infrastructure required for the R-SWAP protocol can be narrowed down to the cost of the relay. Indeed the blockchain relay is undoubtedly the piece of software that has the highest operational cost, as high as 3600$ per day for a Bitcoin relay on the Ethereum blockchain at the current price. However, recent research [35] suggests that the operational cost of relays could be reduced by 92%.

3.3 Conclusions

In this chapter, we present the concept of blockchain interoperability. We review the different definitions given in the literature and conduct a short review of the current applications and use cases. The most formal definition of blockchain interoperability from Lafourcade et al. in [54] comes to the conclusion that there can not be interoperability without a "2 in 1" system where a blockchain is contained in another. But considering our review of the state of the art, we conclude that Lafourcade's definition is too narrow and does not include all use cases of blockchain interoperability. We extend his definition to all systems where the execution of a transaction on a blockchain A is conditioned by the state of a blockchain BC. Without having a "2 in 1" blockchain, we can leverage protocols such as blockchain Oracles or Herlihy's atomic cross-chain swap [40] to achieve blockchain interoperability.

We present Herlihy's atomic cross-chain swap [40] and the implementation we realized of this protocol between the two most valued blockchain systems at that time, Bitcoin and Ethereum. We evaluate the protocol and determine its weaknesses. We have shown that the

protocol safety was highly dependent on the participant's blockchain client availability, and thus that there might be a safety violation in case of a client crash. We observe that its performances are bad compared to the existing solutions for asset exchange across blockchains. Finally, we consider that the protocol has bad ergonomics because it requires the participants to stay online during the whole process. From those conclusions, we propose R-SWAP, an atomic cross-chain swap protocol that improves Herlihy's original design by using the relaying technique and blockchain adapters. We provide a definition of the time-bounded atomic cross-chain swap and prove the correctness of the protocol. We also evaluate its performance and cost. The R-SWAP protocol solves the safety violation problem that Herlihy's swap had. Moreover, it has better ergonomics as all tasks are automated. However, with the current state of the relaying technique, the R-SWAP protocol does not match the performance and cost of the centralized solutions for exchanging assets across chains. Indeed, the cost of maintaining a relay is currently very high on proof-of-work based blockchains because of transaction fees. Recent proposal suggest that this cost could be reduced drastically with an on demand approach.

Chapter 4 Conslusion

As this thesis was conducted as a partnership between the industry and the academia, in this conclusion we are going to discuss topics that go beyond the technical aspect of our work.

4.1 Thesis general conclusion

Context During the past few years, blockchains systems have gained a lot of traction both from the industry and academia. Besides some declines inherent to every market, we have seen a significant and constant increase in the overall value retained in blockchain systems. Public blockchain systems, which are the ones we were focused on during this thesis, have proven their abilities as a store of value. In the financial and banking sphere, many argue that distributed systems such as blockchains would offer better transparency, performance, and security than the currently used centralized systems. Several central banks, such as La Banque de France [53], are conducting experimentation on central banks' digital currencies. On the other hand, the DeFi ecosystem proposes an everyday wider range of classical financial instruments on blockchain systems. One version of the story of the genesis of Bitcoin, which I will admit, might be a little romanticized, states that blockchain systems emerged from pseudonymous communities on online forums after the disillusion of the 2008 financial crisis. At that time, Bitcoin was disregarded by financial institutions and heavily criticized by politicians. But today, we can state that both the blockchain community and the institutional actors have moved toward each other. Institutional actors are considering blockchain as an infrastructure for their activities and the blockchain ecosystem is making efforts to make blockchain systems more institutional by for example establishing a legal framework. Among other efforts, members of the blockchain community in Europe have formed the Association for the Development of Crypto-Assets (ADAN), which has engaged in discussions with the European Parliament to create a legal status for crypto-assets. Binance, a top cryptocurrency exchange platform has just obtained a Digital Asset Service Provider license from the French Autorité des marchés financiers [72]. Thus, in our opinion, blockchain systems are on the right path to becoming a prevalent technology for banking and financial institutions.

Problem In order to prove their pertinence, blockchain systems still have to overcome some technological barriers. Several topics are being addressed by both the industry and academia. Some are working on the security guarantees of blockchain systems, others are working on finality, others on accountability and transparency, etc. But the problem we chose to work on is, how are we going to integrate blockchain technologies into the existing technology stack and infrastructure of the current financial system. In layman's terms, the problem of this thesis is *how can blockchain systems talk to the outside world ?* This is the topic of blockchain interoperability. In this document, we address this topic through the prism of two different problems. In Chapter 2, we address the blockchain oracle problem, or how can we safely input data from the outside world into a blockchain system. In Chapter 3, we address the problem of atomic cross-chain swaps, or how can we synchronize transactions between heterogeneous chains between potentially malicious participants without a trusted third party. If those two problems were to be resolved, then blockchain systems could be integrated into any underlying legacy infrastructure.

Contributions In Chapter 1, we informally introduced some concepts and technologies that combined makes up a blockchain system. We presented the state of the art of the technology by reviewing applications and use cases blockchain can serve. Finally, we opened up the next chapter by introducing blockchain interoperability.

In Chapter 2, we addressed the first problem of this thesis which is, how can data from the outside world be safely inputted into blockchain systems that, by design, are self-contained environments with limited access to the exterior. To answer this problem, we proposed a formal definition of the blockchain oracle problem. We reviewed some existing solutions and related works and then proposed our own protocol that provides better properties than the state of the art. Indeed, our proposal is positioned into the asynchronous model, which is an improvement compared to the state of the art. We also prove the correctness of this protocol.

In Chapter 3, we addressed the atomic cross-chain swap problem. We presented Herlihy's [40] protocol and the implementation of it that we have realized. We analyzed the cost and performance of this protocol and concluded on its limitations. As a result of these findings, we proposed an improved protocol that overcomes the said limitations. We defined the time-locked atomic cross-chain swap problem and proposed an improved protocol that uses cross-chain verification for better security guarantees. We prove the correctness of the protocol and provide

a theoretical analysis of its cost and performance.

4.2 Perspectives and future works

From a short-term point of view, we would like to implement the PoWacle protocol in order to experimentally test its performance and properties. In chapter 2 we semi-formally presented the protocol, provided a theoretical analysis of the protocol, and proved that it satisfies better properties than the state of the art. But we would like to show that this proof holds in a real-world environment. Surely some empirical work would open up new perspectives in terms of research and business. In this document, we decided to use the median of the signed observations as the result returned to the client. But we might also test some other dispositions. We could for example decide to exclude the extreme observed values (lowest and highest) and average the remaining values. Another experimental value we would like to work on is the size of the set of oracles. We would like to experimentally find which set size provides the best performance to security ratio. We could also experiment with some other parameters such as the difficulty adjustment process. Until now, we have decided to adjust the difficulty depending on previous report times. However, we could try some more dynamic strategies, such as real-time network performance monitoring. DAA which stands for difficulty adjustment algorithm is on its own a topic of research and we could surely benefit from the works in the literature. Finally, we would like to generalize the protocol to other use cases. Indeed, in this document, we focused on price values. However, the oracle problem is not specific to the kind of values observed and thus the approach could be generalized to any kind of arbitrary data.

Broadly speaking, centralized solutions are more simple to implement, have a lower operating cost and show better performance than their distributed analogs. Blockchain systems are not an exception to this rule. Indeed, while the contributions presented in this document propose an alternative to centralized solutions for swapping assets and gathering off-chain data, their centralized analog provides better performance at a lower cost. Thus, from a mid to long-term perspective, future works should be focused on improving the cost and performance of the protocols presented in this document.

A very promising topic of research is what we call rollups in layer two solutions. Layer 2 blockchains are scalability solutions originally developed to off-load network congestion from the Ethereum network. The overall goal is to have faster and cheaper transactions than on the

main Ethereum chain while leveraging the security of Ethereums execution layer. To do so, a technique called rollups is used. The principle is to batch transactions from layer 2 chains inside a single transaction that will be published to the main chain. By reducing the transaction cost, we could improve the security of the oracle network. Indeed, if the transaction costs are reduced, we can consider having a larger set of observers and thus, a network that is more difficult to attack. The properties and security guarantees of those layer 2 chains are different from the chains we considered in this document. This results in a different environment and model. Thus in future works, we would like to confront our proposals to these new models.

Bibliography

[1] *A crypto wallet gateway to blockchain apps*. URL: https://metamask.io/.

[2] Shubhani Aggarwal and Neeraj Kumar. "Chapter Twenty - Attacks on blockchain Working model." In: *The Blockchain Technology for Secure and Smart Applications across Industry Verticals*. Ed. by Shubhani Aggarwal, Neeraj Kumar, and Pethuru Raj. Vol. 121. Advances in Computers. Elsevier, 2021, pp. 399–410. DOI: https://doi.org/10.1016/bs.adcom.2020.08.020. URL: https://www.sciencedirect.com/science/article/pii/S0065245820300759.

[3] Yackolley Amoussou-Guenou. "Governing the Commons in Blockchains". PhD thesis. 2020. URL: https://tel.archives-ouvertes.fr/tel-03173517/document.

[4] Yackolley Amoussou-Guenou et al. "Dissecting Tendermint". In: *Networked Systems - 7th International Conference, NETYS 2019, Marrakech, Morocco, June 19-21, 2019, Revised Selected Papers*. Ed. by Mohamed Faouzi Atig and Alexander A. Schwarzmann. Vol. 11704. Lecture Notes in Computer Science. Springer, 2019, pp. 166–182. DOI: 10.1007/978-3-030-31277-0_11. URL: https://doi.org/10.1007/978-3-030-31277-0%5C_11.

[5] Elli Androulaki et al. "Hyperledger fabric: a distributed operating system for permissioned blockchains". In: *Proceedings of the Thirteenth EuroSys Conference, EuroSys 2018, Porto, Portugal, April 23-26, 2018*. Ed. by Rui Oliveira, Pascal Felber, and Y. Charlie Hu. ACM, 2018, 30:1–30:15. DOI: 10.1145/3190508.3190538. URL: https://doi.org/10.1145/3190508.3190538.

[6] Andreas M Antonopoulos. *Mastering Bitcoin: Programming the open blockchain*. " O'Reilly Media, Inc.", 2017.

[7] Adam Back et al. "Enabling blockchain innovations with pegged sidechains". In: *URL: http://www. opensciencereview. com/papers/123/enablingblockchain-innovations-with-pegged-sidechains* (2014).

[8] Bandchain. *Band protocol system overview*. URL: https://docs.bandchain.org/whitepaper/system-overview.html.

[9] Rafael Belchior et al. “A Survey on Blockchain Interoperability: Past, Present, and Future Trends”. In: *CoRR* abs/2005.14282 (2020). arXiv: 2005.14282. URL: https://arxiv.org/abs/2005.14282.

[10] Juan Benet. “Ipfs-content addressed, versioned, p2p file system”. In: *arXiv preprint arXiv:1407.3561* (2014).

[11] Gavin Birch. *Cosmos stargate update overview*. 2020. URL: https://figment.io/resources/cosmos-stargate-upgrade-overview/%5C#ibc.

[12] Sean Braithwaite et al. “Tendermint Blockchain Synchronization: Formal Specification and Model Checking”. In: *Leveraging Applications of Formal Methods, Verification and Validation: Verification Principles*. Ed. by Tiziana Margaria and Bernhard Steffen. Cham: Springer International Publishing, 2020, pp. 471–488. ISBN: 978-3-030-61362-4.

[13] Lorenz Breidenbach et al. “Chainlink off-chain reporting protocol”. In: *URl: https://blog.chain. link/off-chain-reporting-live-on-mainnet* (2021).

[14] Eric A Brewer. “Towards robust distributed systems”. In: *PODC*. Vol. 7. 10.1145. Portland, OR. 2000, pp. 343477–343502.

[15] Ethan Buchman, Jae Kwon, and Zarko Milosevic. *The latest gossip on BFT consensus*. 2019. arXiv: 1807.04938 [cs.DC].

[16] John F. Buford, Heather Yu, and Eng Keong Lua. “Chapter 14 - Security”. In: *P2P Networking and Applications*. Ed. by John F. Buford, Heather Yu, and Eng Keong Lua. Boston: Morgan Kaufmann, 2009, pp. 319–340. ISBN: 978-0-12-374214-8. DOI: https://doi.org/10.1016/B978-0-12-374214-8.00014-3. URL: https://www.sciencedirect.com/science/article/pii/B9780123742148000143.

[17] Vitalik Buterin et al. “A next-generation smart contract and decentralized application platform”. In: *white paper* 3.37 (2014).

[18] Vitalik Buterin. *Why Sharding is great: Demystifying the technical properties*. Apr. 2021. URL: https://vitalik.ca/general/2021/04/07/sharding.html.

[19] Christian Cachin, Rachid Guerraoui, and Luis Rodrigues. *Introduction to reliable and secure distributed programming*. Springer Science & Business Media, 2011.

[20] Giulio Caldarelli and Joshua Ellul. “The Blockchain Oracle Problem in Decentralized FinanceA Multivocal Approach”. In: *Applied Sciences* 11.16 (2021), p. 7572.

[21] Miguel Castro and Barbara Liskov. “Practical Byzantine fault tolerance and proactive recovery”. In: *ACM Transactions on Computer Systems (TOCS)* 20.4 (2002), pp. 398–461.

[22] Vicent Cholvi et al. “Appending Atomically in Byzantine Distributed Ledgers”. In: *arXiv preprint arXiv:2002.11593* (2020).

[23] CoinMarketCap. *Cryptocurrency Prices, Charts And Market Capitalizations*. URL: https://coinmarketcap.com/.

[24] Justin Connell. *Sophisticated trading bot exploits Synthetix Oracle, funds recovered*. July 2019. URL: https://cointelegraph.com/news/sophisticated-trading-bot-exploits-synthetix-oracle-funds-recovered.

[25] *Consensus mechanisms*. URL: https://ethereum.org/en/developers/docs/consensus-mechanisms/.

[26] Consensys. *ethereum/btcrelay*. Oct. 2017. URL: https://github.com/ethereum/btcrelay.

[27] Christian Decker and Roger Wattenhofer. “Information propagation in the bitcoin network”. In: *IEEE P2P 2013 Proceedings*. IEEE. 2013, pp. 1–10.

[28] Roger Dingledine, Nick Mathewson, and Paul Syverson. *Tor: The second-generation onion router*. Tech. rep. Naval Research Lab Washington DC, 2004.

[29] K. Driscoll et al. “The real Byzantine Generals”. In: *The 23rd Digital Avionics Systems Conference (IEEE Cat. No.04CH37576)*. Vol. 2. 2004, pp. 6.D.4–61. DOI: 10.1109/DASC.2004.1390734.

[30] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. “Consensus in the presence of partial synchrony”. In: *Journal of the ACM (JACM)* 35.2 (1988), pp. 288–323.

[31] Cynthia Dwork and Moni Naor. “Pricing via processing or combatting junk mail”. In: *Annual international cryptology conference*. Springer. 1992, pp. 139–147.

[32] Michael J Fischer. “The consensus problem in unreliable distributed systems (a brief survey)”. In: *International conference on fundamentals of computation theory*. Springer. 1983, pp. 127–140.

[33] Michael J Fischer, Nancy A Lynch, and Michael S Paterson. "Impossibility of distributed consensus with one faulty process". In: *Journal of the ACM (JACM)* 32.2 (1985), pp. 374–382.

[34] Philipp Frauenthaler et al. "Cross-blockchain Token Transfer with Blockchain Relay". In: *Gas* 300 (2020), pp. 600–000.

[35] Philipp Frauenthaler et al. "Testimonium: A Cost-Efficient Blockchain Relay". In: *CoRR* abs/2002.12837 (2020). arXiv: 2002.12837. URL: https://arxiv.org/abs/2002.12837.

[36] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. "The bitcoin backbone protocol: Analysis and applications". In: *Annual international conference on the theory and applications of cryptographic techniques*. Springer. 2015, pp. 281–310.

[37] Vincent Gramoli. "From blockchain consensus back to Byzantine consensus". In: *Future Generation Computer Systems* 107 (2020), pp. 760–769. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2017.09.023. URL: https://www.sciencedirect.com/science/article/pii/S0167739X17320095.

[38] Abdelatif Hafid, Abdelhakim Senhaji Hafid, and Mustapha Samih. "Scaling blockchains: A comprehensive survey". In: *IEEE Access* 8 (2020), pp. 125244–125262.

[39] Runchao Han, Haoyu Lin, and Jiangshan Yu. "On the optionality and fairness of Atomic Swaps". In: *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*. 2019, pp. 62–75.

[40] Maurice Herlihy. "Atomic Cross-Chain Swaps". In: *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*. PODC '18. Egham, United Kingdom: Association for Computing Machinery, 2018, pp. 245–254. ISBN: 9781450357951. DOI: 10.1145/3212734.3212736. URL: https://doi.org/10.1145/3212734.3212736.

[41] Jon Huang, Claire O'neill, and Hiroko Tabuchi. *Bitcoin uses more electricity than many countries. how is that possible?* Sept. 2021. URL: https://www.nytimes.com/interactive/2021/09/03/climate/bitcoin-carbon-footprint-electricity.html.

[42] Adnan Imeri, Nazim Agoulmine, and Djamel Khadraoui. "Smart Contract modeling and verification techniques: A survey". In: *8th International Workshop on ADVANCEs in ICT Infrastructures and Services (ADVANCE 2020)*. 2020, pp. 1–8.

[43] Interlay. *interlay/BTC-Relay-Solidity*. 2020. URL: https://github.com/interlay/BTC-Relay-Solidity.

[44] interlay. *interBTC Specification documentation*. 2021. URL: https://spec.interlay.io/intro/architecture.html.

[45] Xin-Jian Jiang and Xiao Fan Liu. "CryptoKitties Transaction Network Analysis: The Rise and Fall of the First Blockchain Game Mania". In: *Frontiers in Physics* 9 (Mar. 2021), p. 631665. DOI: 10.3389/fphy.2021.631665.

[46] Southurst Jon. *Chainlink exploits lead to ETH losses-again*. Sept. 2020. URL: https://coingeek.com/chainlink-exploits-lead-to-eth-losses-again/.

[47] Shoji Kasahara and Jun Kawahara. "Effect of Bitcoin fee on transaction-confirmation process". In: *arXiv preprint arXiv:1604.00103* (2016).

[48] Aggelos Kiayias et al. "Ouroboros: A provably secure proof-of-stake blockchain protocol". In: *Annual International Cryptology Conference*. Springer. 2017, pp. 357–388.

[49] Zindros Dionysis Kiayias Aggelos. "Proof-of-work sidechains". In: *International Conference on Financial Cryptography and Data Security*. Springer. 2019, pp. 21–34.

[50] David Koops. "Predicting the confirmation time of bitcoin transactions". In: *arXiv preprint arXiv:1809.10596* (2018).

[51] Jae Kwon. "Tendermint: Consensus without mining". In: *Draft v. 0.6, fall* 1.11 (2014).

[52] Buchman Ethan Kwon Jae. "A network of distributed ledgers". In: *Cosmos, dated* (2018), pp. 1–41.

[53] *La Banque de France réalise une nouvelle expérimentation de Monnaie Numérique de Banque Centrale*. June 2021. URL: https://www.banque-france.fr/communique-de-presse/la-banque-de-france-realise-une-nouvelle-experimentation-de-monnaie-numerique-de-banque-centrale.

[54] Pascal Lafourcade and Marius Lombard-Platet. "About blockchain interoperability". In: *Information Processing Letters* 161 (2020), p. 105976.

[55] Leslie Lamport. "Time, Clocks, and the Ordering of Events in a Distributed System". In: *Commun. ACM* 21.7 (July 1978), pp. 558–565. ISSN: 0001-0782. DOI: 10.1145/359545.359563. URL: https://doi.org/10.1145/359545.359563.

[56] Leslie Lamport, Robert Shostak, and Marshall Pease. “The Byzantine generals problem”. In: *Concurrency: the Works of Leslie Lamport*. 1981, pp. 203–226.

[57] Carlos Lopez De Lara. *What’s Ouroboros? the cardano proof of stake protocol*. Nov. 2019. URL: https://medium.com/@carloslopezdelara/whats-ouroboros-the-cardano-proof-of-stake-protocol-ad4b958e152e.

[58] Arnaud Legout, Guillaume Urvoy-Keller, and Pietro Michiardi. “Understanding bittorrent: An experimental perspective”. In: (2005).

[59] Zhang Leo. *Merkle Patricia Trie explained*. Aug. 2020. URL: https://medium.com/@chiqing/merkle-patricia-trie-explained-ae3ac6a7e123.

[60] Sergio Demian Lerner. *Rsk*. Tech. rep. Tech. Rep, 2015.

[61] Loi Luu et al. “A secure sharding protocol for open blockchains”. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 2016, pp. 17–30.

[62] Nancy A Lynch. *Distributed algorithms*. Elsevier, 1996.

[63] Léonard Lys. *Blockchain-based assets and Central Banking: The meeting of two worlds*. Apr. 2020. URL: https://blog.palo-it.com/en/blockchain-based-assets-and-central-banking-the-meeting-of-two-worlds.

[64] MakerDAO. *The maker protocol white paper: Feb 2020*. URL: https://makerdao.com/en/whitepaper/.

[65] Ralph Charles Merkle. *Secrecy, authentication, and public key systems*. Stanford university, 1979.

[66] Renita M Murimi and Grace Guiling Wang. “On elastic incentives for blockchain oracles”. In: *Journal of Database Management (JDM)* 32.1 (2021), pp. 1–26.

[67] Satoshi Nakamoto. “Bitcoin: A peer-to-peer electronic cash system”. In: *Decentralized Business Review* (2008), p. 21260.

[68] DOS network. *A Decentralized Oracle Service boosting blockchain usability with off-chain data & verifiable computing power*. 2011. URL: https://s3.amazonaws.com/whitepaper.dos/DOS+Network+Technical+Whitepaper.pdf.

[69] John Newbery. *An introduction to Bitcoin Core fee estimation*. Oct. 2018. URL: https://bitcointechtalk.com/an-introduction-to-bitcoin-core-fee-estimation-27920880ad0.

[70] Nikhilesh. *Bitcoin futures volume is more significant than you think*. Mar. 2019. URL: https://www.coindesk.com/markets/2019/03/22/bitcoin-futures-volume-is-more-significant-than-you-think-bitwise-says.

[71] Shen Noether. "Ring SIgnature Confidential Transactions for Monero". In: *IACR Cryptol. ePrint Arch.* (2015), p. 1098. URL: http://eprint.iacr.org/2015/1098.

[72] *Obtaining a DASP registration/optional licensing*. 2022. URL: https://www.amf-france.org/en/professionals/fintech/my-relations-amf/obtain-dasp-authorisation.

[73] Faridi Omar. *Cosmos Network Now Has Nearly 100 Validators, 6-7 Second Block Times*. Apr. 2019. URL: https://www.cryptoglobe.com/latest/2019/04/cosmos-network-now-has-nearly-100-validators-6-7-second-block-times/.

[74] Diego Ongaro and John Ousterhout. "In Search of an Understandable Consensus Algorithm". In: *2014 USENIX Annual Technical Conference (USENIX ATC 14)*. Philadelphia, PA: USENIX Association, June 2014, pp. 305–319. ISBN: 978-1-931971-10-2. URL: https://www.usenix.org/conference/atc14/technical-sessions/presentation/ongaro.

[75] *Parachain Slots Auction ů Polkadot Wiki*. URL: https://wiki.polkadot.network/docs/learn-auction.

[76] Paradigm. *Chainlink: Detailed review on the project*. Mar. 2019. URL: https://medium.com/paradigm-fund/chainlink-detailed-review-on-the-project-9dbd5e050974.

[77] Collins Patrick. *Building and using External Adapters*. Jan. 2021. URL: https://blog.chain.link/build-and-use-external-adapters/.

[78] Joseph Poon and Thaddeus Dryja. *The bitcoin lightning network: Scalable off-chain instant payments*. 2016.

[79] *Proposer selection procedure in Tendermint*. URL: https://docs.tendermint.com/v0.32/spec/reactors/consensus/proposer-selection.html.

[80] Jamie Redman. *Report: Blockchain price oracle manipulation produces millions in losses, shows no signs of slowing altcoins bitcoin news*. Nov. 2020. URL: https://news.bitcoin.com/report-blockchain-price-oracle-manipulation-produces-millions-in-losses-shows-no-signs-of-slowing/.

[81] M.G. Reed, P.F. Syverson, and D.M. Goldschlag. "Anonymous connections and onion routing". In: *IEEE Journal on Selected Areas in Communications* 16.4 (1998), pp. 482–494. DOI: 10.1109/49.668972.

[82] Rolandkofler. *rolandkofler/blocktime*. 2017. URL: https://github.com/rolandkofler/blocktime.

[83] David Schwartz, Noah Youngs, Arthur Britto, et al. "The ripple protocol consensus algorithm". In: *Ripple Labs Inc White Paper* 5.8 (2014), p. 151.

[84] *Simplified Payment Verification*. URL: https://wiki.bitcoinsv.io/index.php/Simplified_Payment_Verification.

[85] Mathis Steichen et al. "Blockchain-Based, Decentralized Access Control for IPFS". In: *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. 2018, pp. 1499–1506. DOI: 10.1109/Cybermatics_2018.2018.00253.

[86] Nolan Tier. Mar. 2013. URL: https://bitcointalk.org/index.php?topic=193281.msg2224949%5C%5C#msg2224949.

[87] Ngan Ton. *A complete list of cryptocurrency exchange hacks [updated]*. July 2020. URL: https://blog.idex.io/all-posts/a-complete-list-of-cryptocurrency-exchange-hacks-updated.

[88] *Validators in tendermint core*. URL: https://docs.tendermint.com/master/nodes/validators.html.

[89] Buterin Vitalik. *Chain interoperability*. 2016.

[90] Gang Wang. "SoK: Exploring Blockchains Interoperability." In: *IACR Cryptol. ePrint Arch.* 2021 (2021), p. 537.

[91] Gang Wang et al. "Sok: Sharding on blockchain". In: *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*. 2019, pp. 41–61.

[92] Wang Chun Wei. “The impact of Tether grants on Bitcoin”. In: *Economics Letters* 171 (2018), pp. 19–22.

[93] Gavin Wood et al. “Ethereum: A secure decentralised generalised transaction ledger”. In: *Ethereum project yellow paper* 151.2014 (2014), pp. 1–32.

[94] Gavin Wood. “Polkadot: Vision for a heterogeneous multi-chain framework”. In: *White Paper* 21 (2016).

[95] Yang Xiao et al. “A Survey of Distributed Consensus Protocols for Blockchain Networks”. In: *IEEE Commun. Surv. Tutorials* 22.2 (2020), pp. 1432–1465. DOI: 10.1109/COMST.2020.2969706. URL: https://doi.org/10.1109/COMST.2020.2969706.

[96] Jiahua Xu and Benjamin Livshits. “The anatomy of a cryptocurrency pump-and-dump scheme”. In: *28th {USENIX} Security Symposium ({USENIX} Security 19)*. 2019, pp. 1609–1625.

[97] Victor Zakhary, Divyakant Agrawal, and Amr El Abbadi. “Atomic commitment across blockchains”. In: *Proceedings of the VLDB Endowment* 13.9 (2020).

[98] A. Zamyatin et al. “XCLAIM: Trustless, Interoperable, Cryptocurrency-Backed Assets”. In: *2019 IEEE Symposium on Security and Privacy (SP)*. 2019, pp. 193–210. DOI: 10.1109/SP.2019.00085.

[99] Alexei Zamyatin et al. “Sok: Communication across distributed ledgers”. In: *International Conference on Financial Cryptography and Data Security*. Springer. 2021, pp. 3–36.