(THESIS TITLE PAGE SAMPLE)

UNIVERSITA' DEGLI STUDI DI TORINO:

DIPARTIMENTO DI: INFORMATICA

DOTTORATO DI RICERCA IN :
INFORMATICA

CICLO: 34

TITOLO DELLA TESI: ON BLOCKCHAIN INTEROPERABILITY AND PRIVACY

TESI PRESENTATA DA: FADI BARBARA

TUTOR(S): PROF. CLAUDIO SCHIFANELLA

COORDINATORE DEL DOTTORATO: PROF. VIVIANA PATTI

ANNI ACCADEMICI: 2018/19; 2019/20; 2020/21; 2021/22; 2022/23

SETTORE SCIENTIFICO-DISCIPLINARE DI AFFERENZA*: INF/01

(*__Please note__: in case of a multidisciplinary thesis, please insert the code of your major).

On Blockchain Interoperability and Privacy

Fadi Barbàra

February 27, 2023

Contents

Chapter 1

Introduction

1.1 The Beginning

Bitcoin was born in 2008. It was not the first cryptocurrency. But it was the first decentralized and cryptographically sound cryptocurrency. It was announced in an email to a cryptography mailing list. Such kind of projects were common at the time and in the previous decade. Yet, this one was the first one that respected all the requirements needed to have a decentralized and pseudo-anonymous cryptocurrency. At least at a first glance.

Since Bitcoin is free software[1], many projects were born after people saw that Bitcoin worked and had real financial value. Some projects were just financial rip off that tried to capitalize on the financial success of Bitcoin: those kinds of projects are created every day right now and they are all self-called "bitcoin killers". But other projects were legitimate attempts to improve on some of Bitcoin's shortcomings. For example: a project like Namecoin tries to create a Blockchain-based root zone management in the Domain Name System (DNS) to compete with the centralized one maintained by the Internet Assigned Numbers Authority (IANA)[2].

It is widely known, at least after 2013 [MPJ+13], that it is possible to trace Bitcoin payments sometimes even to the real identity of the payer and for this reason other projects decided to improve the privacy of Bitcoin. For example the authors of the project Cryptonote [VS13] decided to widen the set of possible payers for each transaction in order to obfuscate the real payer. In order to do that the creator of Cryptonote substituted the normal ECDSA signature with a ring signature variant of the EdDSA signature scheme. Since the need of privacy and anonymity in an electronic cash systems is highly felt by the community, other projects tried to obtain similar results using completely different approaches. ZCash [Hop22]is another blockchain project that aims to obtain privacy and anonymity by encrypting all the blockchain and proving transactions using succinct zero-knowledge proofs.

Other projects decided to improve the scripting capabilities of Bitcoin, but not in a trivial way[3]. One of the projects that has became popular is Ethereum, but also Tezos and Algorand are currently under active development. These projects implemented the ideas of Nick Szabo of a *smart contract* able to execute code in a decentralized way [Sza97].

In general, new strands of research sparkled after the creation of Bitcoin, each of them trying to solve one of the problems related to the distributed database that we now call *blockchain*. Solutions were and are currently proposed. Some of these solutions became independent blockchains with real-word usage and financial weight. And with many blockchains, the need to exchange value between them increased. This is the topic of the next sections and the solutions around that are the topics of this PhD thesis.

[1] Bitcoin Core is currently under the MIT license

[2] https://www.iana.org/about

[3] Since the inception of bitcoin, many of the scripting capabilities used in the Bitcoin scripting language Script where deactivated for security reasons. A trivial way to improve Bitcoin scripting capability would then be to fork the Bitcoin blockchain and reactivate the deactivated scripting capabilities.

1.2 The Current State

As mentioned, the blockchain space has currently tens of thousands between coins and tokens and there has to be a way to exchange them. Generally, voices from the business world talk about that in terms of "DeFi matrix", i.e. the possibility of buying (and therefore pricing) every token or coin with any other token or coin. It is possible that in the future it will be possible to exchange information, generally in the form of smart contracts tokens, and products or services in a way that blurs the line between product and medium of exchange.

Perhaps we are already there. In fact, what am I doing when I exchange bitcoins for ethers on the Ethereum blockchain? Which one is the medium of exchange? One way to see it is that I am using bitcoins as a currency (i.e. as medium of exchange) to buy the product (the ether) that lets me use applications and services which happen to use the Ethereum blockchain as settlement layer: that is similar to use dollars or euros to buy oil to power my car, which happens to burn oil to run. Another way to see it is that I am a seller of bitcoins (i.e. the product) and I use the medium of exchange called ether: as before, this view is justified since ether (and gas in particular) is the medium of exchange used to pay for the use of service applications, similar to the dollars or euros used to perform computations on cloud computing platforms.

It can be argued that this happens since at least the invention of coupons or fidelity points, gained in one shop and spendable for other services in other shops. And I agree with that argument: it just proves that this transformation started a long time ago and we are seeing a qualitatively more technological, efficient and promising way to do that. Which proves that the probability of this trend stopping now is very low, especially since we are talking about the extension from simple currency (static representation of value) to smart contract (dynamic representation of value, i.e. information).

Therefore the majority of users will probably gravitate around a small number of currencies they will use to price any other product, either physical or digital, or token. This does not contradict the DeFi matrix thesis. Assume two tokens with ticker symbol $TK1$ and $TK2$. Assume they both are the product of a smart contract on the Ethereum blockchain whose coin has ticker symbol ETH. Assume $1\ TK1 = \alpha\ ETH$ and $1\ TK2 = \beta\ ETH$. Therefore it is easy to obtain $1\ TK1$ by just giving $\frac{\beta}{\alpha}\ TK2$, the DeFi matrix works, yet people gravitate toward a few cryptocurrencies to base the price[4].

From that argument it may appear that the future will be "multi layers" but not "multi chain'. In fact, the platform/blockchain with the higher usage will attract the most people and it is better to create *ad-hoc* systems that interact with that platform (i.e. multiple layers) than separate blockchains with different architecture: Today the blockchain that has the higher probability of having a monopoly on applications and users is the Ethereum blockchain.

Many experts have publicly stated that they are more in favor of a multi layer ecosystem rather than a multi chain one. For example (perhaps unsurprisingly) Vitalik Buterin is a strong proponent of that [But22]. Buterin's main argument is that possible forks in one blockchain could cause loss of funds for users using that blockchain as settlement layer during a cross-chain exchange. This (Buterin argues) can not happen in the multi layer-blockchain paradigm since the layer uses the blockchain for settlements: in case of a fork (which can happen only on the blockchain), the user just re-does the transaction and incurs no loss of funds. We analyzed this problem in Section 2.1.3 and Section 8.1.

But blockchains are not used as medium of exchange's settlement layer only. Coins on a blockchain and the blockchain's services are also information, so cryptocurrencies and smart contracts are ways to exchange information using cryptography. In a complex real world, such as the one we inhabit, external forces like censorship, better technology, and in general every new event that drives new assumptions on how the real word works will inevitably affect the properties needed by these distributed databases and their services. In those cases people will need to move their funds and the services associated with them from the Old Way of information exchange to the New Way. Those kinds of changes are turbulent, at least for coordination reasons.

That is why saying that the future is "multi layers" but not "multi chain", means assuming that the chain that acts as the ultimate settlement layer will never be attacked or will never crumble to external forces. But history teaches that this is not something that it is safe to assume. On the

[4]Of course is vastly more complex than that: prices between $TK1$ and $TK2$ are not fixed and they also depend on many factors different from the quote in ETH terms. For example the work of arbitrageurs, who are happy to use the ETH coin as bridge instead of exchanging $TK1$ and $TK2$ directly if that means a profit, may destabilize the price in the short run, while stabilizing it in the long run.

contrary the very fact that there are easy ways to move the funds from one chain to another will act as a disincentive or at least an obstacle to the effectiveness of an attack to the blockchain.

1.3 Organization of the thesis and contributions

This thesis is split into three parts. Part I deals with fundamental knowledge which I will use to explain the protocols I presented as a student of PhD together with their improvement with respect to the state of the art. In particular;

- Chapter 2 deals with how the consensus in a distributed system works. While this is a prerequisite to understand the whole thesis, the content of the chapter will be specifically used to explain the solutions presented in Chapters 9, 10, 11 and 12;
- Chapter 3 deals with the transaction models blockchain projects use. The content of the chapter will be specifically used to explain the solutions presented in Chapters 5, 9 and 10;
- Chapter 4 deals with how transaction signing works. The content of the chapter will be specifically used to explain the solutions presented in Chapters 9, 10 and 11;
- Chapter 5 deals with commit-reveal protocols, vastly used for interoperability. The content of the chapter will be specifically used to explain the solutions presented in Chapters 9, 10 and 11;
- Chapter 6 deals with the relation between blockchains and commerce. The content of the chapter will be specifically used to explain the solutions presented in Chapters 13 and 14;
- Chapter 7 deals with the concept of fair exchange and interoperability between blockchains. While this is a prerequisite to understand the whole thesis, the content of the chapter will be specifically used to explain the solutions presented in Chapters 9, 10, 11 and 12;
- Chapter 8, deals with the attacks to the various layer of the blockchain which can disrupt any interoperability protocol during its execution. The content of this chapter are a prerequisite to understand the whole thesis, especially to since many of the choices in the design of those solution has been done as a mitigation of potential attacks.

On the other hand, chapters in Part II and Part III are based on solutions papers I published or that are current in a state of evaluation in a journal. In particular:

- Chapter 9 proposes a protocol on a protocol called DMix which acts as a decentralized mixer presented in [BS20a]. It improves the state of the art by adding decentralization and mitigations to the (currently-known) privacy threats (summarized in Table 8.2);
- Chapter 10 proposes a protocol on a multi-users capable Hash-Time Lock Contracts (HTLCs) for atomic swaps as submitted and currently in minor revision state in [BS22b], which building on DMix uses its principles on two different blockchains obtaining atomic swaps that solves some of the problems of HTLCs (detailed in Section 5.3)
- Chapter 11 proposes a protocol on time-lock based decentralized-exchange protocol published in [BMS21]. As far as I know, the first protocol that proposes a synchronicity-method to obtain a modular atomic swap with a decentralized matching algorithm;
- Chapter 12 is based on my paper on the exchange of tokenized bitcoins presented in [BS22a]. Differently from other exchange methods, I use the Stateless SPV construct here in order to avoid time-locks and the consequent loss of time to obtain a peer-to-peer exchange that bypass privacy-threatening procedures embedded into comment bitcoin tokenization protocols.
- Chapter 13 proposes a protocol on how to integrate blockchains and the delivery of physical products with no intermediaries in an anonymous and privacy preserving way as presented in [BS20b]: this is an improvement over the state of the art since it does not need any smart-contract capability and can be implemented in blockchains with non-Turing complete languages such as Bitcoin;

- Chapter 14 proposes a protocol on distributed or decentralized payment processing presented in [BSS21]: differently from decentralized-exchanges, it uses aggregators and contains automation algorithms (e.g. automatic taxes allocations or investment decisions using DeFi composability) which are under the complete control of the merchant that deploys it.

Part I

Blockchains Building Blocks

Chapter 2

Consensus

The consensus mechanism is the most important part in blockchains: as a distributed database, a blockchain needs to have a mechanism to ensure that all nodes containing the database copies agree (i.e. reach consensus) on the current state of the database in order to update their copy they maintain locally. This is especially important in case multiple blockchains need to interact with each other. In fact, the inconsistency of even one of the two databases can create financial losses for users who are participating in an atomic swap. This section seeks to explain the underlying structure of blockchain system decisions that are the basis for understanding the financial dangers due to forks (explained in section 2.1.3) and attacks that aim to hack the consensus mechanism 8.1.

Regarding permissionless blockchains, Proof-of-Work (PoW) is the name of the consensus mechanism used both in Bitcoin and (at the time of this writing) in Ethereum. Other Blockchains such as Tezos and Algorand, have a different consensus mechanism called Proof of Stake. Goal of this chapter is to explain their differences and their common traits. Knowledge of how the consensus mechanism works will be needed when we explain interoperability issues between two blockchains.

In the blockchain space, the consensus mechanism is used to process transactions. Generally each miner for each *epoch* (roughly defined as the time between a block and the next one) performs the following steps:

1. Receive messages (which generally are called "transactions" due to the fact that historically those messages were currency transaction in Bitcoin) and stores them in memory in a "pool", which is called *mempool*
2. Picks some of these messages from the mempools, generally the ones with the higher fees
3. Verifies that the picked messages follow the consensus rules
4. Creates a template "block" of messages and additional consensus related data (see Table 7.3 to see how the header of a block is serialized)
5. Performs the Leader Election step (Section 2.2)) and if it is the leader then the miner publish that block

Studies on possible ways of verifying messages received from several parts of a distributed system begun in the 1970s. The Byzantine-generals problem is an abstraction of the possible dishonesty in one or more "processors" (or "replicas") of a system with redundancy. In Lamport's seminal work on the subject [LSP82], the assumption is that all the nodes/processors of the replicated system are previously registered and identified. Using the terminology used today in blockchain systems, the first consensus algorithms of a distributed system assumed a *permissioned* setting.

As mentioned in [LSP82], once the Byzantine-generals problem in a permissioned setting has been solved, the solution works "whether one is implementing a reliable computer using redundant circuitry to protect against the failure of individual chips, or a ballistic missile defense system using redundant computing sites to protect against the destruction of individual sites by a nuclear attack". What was missing before the birth of Bitcoin was the possibility to solve the Byzantine-generals problem in a *permissionless* setting, i.e. in a context where nodes are not identified and may or may not participate without notice.

Since then the study of consensus algorithms has developed and in complex systems such as today we speak more properly of "consensus mechanisms". A consensus mechanism is the set of algorithms that produce consensus in a distributed system. These algorithms include message-passing, leader election, consensus algorithm, fault tolerance mechanisms etc.

In this section we explain how these intermediate steps of a consensus mechanism work. We will not give a complete explanation though: for details about consensus mechanisms see [AW04].

The section is so composed. In Section 2.1 we explain how the nodes communicate between them and in Section 2.2 we explain how the nodes elect a leader: this particular subsection will explain the differences between Proof of Work and Proof of Stake. In Section 2.3 we explain how the nodes can recover from a failure. In Section 2.4 we explain how the nodes can be sure that the distributed system is in a final state regarding a block.

2.1 Message Passing

In a distributed system and in a blockchain in particular, nodes share two distinct types of information, namely *transaction* messages and *block* messages. The purpose of a message changes depending on the transaction model used (see Section 3.1 and Section 3.2), while blocks' information has the same goals for any model.

In the following we briefly explain message propagation by explaining the network topology and the propagation method, further details can be found e.g. in [DSW14]. This explanation will let us introduce the concept of *forks* in the last subsection, which are the basic problem of each interoperability method based on time-lock puzzles as we will see later.

2.1.1 Network Topology

Every blockchain network can be seen as a graph. A new node A use the DNS seed (whose IP address is hardcoded into the software, see the `chainparams.cpp` file[1] in the Bitcoin case and EIP-1459[2] in the case of Ethereum) to discover the other nodes $B_1, \dots, B_n$, where n is the number of open connections A wants to maintain. The nodes B_i will be the only source of information for A. Note that wile *joining* the network is obviously explicit, *leaving* the network is not: A leaves the network silently and the set of other nodes connected to A linger for several hours before purging A from their known-nodes list.

In the absence of evidence to the contrary, we assume that the graph of the blockchain network is connected, namely that for each pair of nodes A and B there exist w intermediate nodes $C_1, \dots, C_w$ such that a message from A propagates through $C_1, \dots, C_w$ to arrive at B. If this does not happen, then the network is said to be *partitioned*. A partitioned network is dangerous, since the partitions track essentially different states that result in incompatible "histories". We will see that different states lead to what is commonly referred to as *fork*.

2.1.2 Propagation Method

Consensus-related messages in blockchain networks are about transactions and blocks. These messages are by far the most frequent. For this reason, nodes do not communicate transactions and blocks directly, but operate a series of meta-communications to avoid A the node that learns of a new transaction or block and has verified their validity. Before transmitting the data, A announces to $B_1, \dots, B_n$ their availability. We call this message *inv*, since it is an Inventory message[3].

A node B_i that receives an *inv* message has three possibilities. If B_i is already aware of the data it will discard this message. On the other hand if B_i does not now about the new data (i.e. B_i does not have the data locally), it will send a *getdata* message to A. A then answers with the data.

The first two cases assume that the information given by A does not contradict the information already received by B_i. In case the message contains new information, but which B_i believes to be contradictory to the information it already has, B_i discards the message. Figure 2.1 highlight how message requests are handled by other nodes.

[1] https://github.com/bitcoin/bitcoin/blob/master/src/chainparams.cpp#L121
[2] https://eips.ethereum.org/EIPS/eip-1459
[3] See https://developer.bitcoin.org/reference/p2p_networking.html

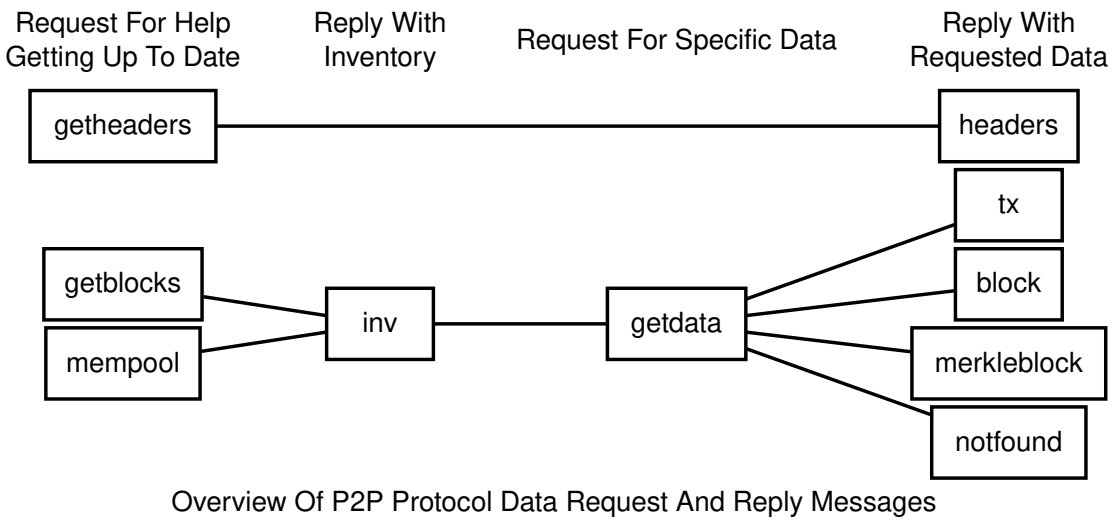


Figure 2.1: Overview of P2P protocol data request and reply messages. Source: https://developer.bitcoin.org/reference/p2p_networking.html

2.1.3 Forks

The word "fork" means different things in different context. In non-technical discussions, forks are generally referred to as a result of disagreements between developers and managers, and therefore for "political" causes. Well known examples are the Bitcoin XT fork[4] and the Bitcoin Cash fork[5]. In those cases we speak of *hard forks*. Another kind of forks are the *soft forks*: this is the preferred method used in Bitcoin to introduce new features. Both of these forks are planned in advantage and effectively create different project: these kind of forks have a big overlap with the concept of fork in e.g. versioning systems such as Git.

The concept of fork in distributed systems is different, even though the outcome is the same: inconsistent states between the different copies of the database which result in incompatible histories and therefore in different finalized or accepted transactions or states. These forks are unplanned and they are generated by the asynchronicity of the network.

Every day there are small (temporary) forks that are created and resolved automatically: the existence of delay between miners and non-honest mining strategies (such as selfish mining [ES18], see Section 2.2) is one of the reasons that leads to blockchain forks.We say that a node A experiences the existence of *a* fork by participating in the network and receiving two conflicting blocks $\mathcal{B}_i$ and $\widehat{\mathcal{B}}_i$. Naturally, it is impossible for A to know about *every* fork in the network, since we saw nodes don't accepts nor relay incoming blocks that conflict with the block it believes to be the chain's head.

When the forks is caused by only one block, it is easy to resolve a fork for a node A by simply replacing the conflicting block with the block it believes to be the chain's head (the concept of "head of blockchain" depends on the specific algorithm used, as we will see in Section 2.2). The problem arises when rejected blocks are numerous: in this case we can talk informally of *reorgs*, as in reorganizations. In a reorg, many blocks are substituted for other new blocks, with different transactions.

Leaving aside the logistical issues consequent to a reorg, this is particularly problematic in the case of a cross-chain atomic swaps. For a detailed analysis of the problems related to forks, see Section 8.1.

[4]See for example https://en.bitcoin.it/wiki/Bitcoin_XT for a brief (and biased) description. For another biased discussion (but on the opposite side) see https://blog.plan99.net/the-resolution-of-the-bitcoin-experiment-dabb30201f7. Because the debate (generally called the *scaling debate*) is outside the scope of this thesis, which focuses on interoperability and not on scalability of blockchains, we will not discuss this issue.

[5]We won't discuss this fork too for similar reasons as in footnote 4.

2.2 Leader Election

At this stage of the consensus mechanism, once transactions by non-mining nodes have been received, the miners set up to update the distributed ledger with some of the transactions in the mempool.

In decentralized and distributed systems, the participated choice of the appointed node is called Leader Election. Informally, the goal is for each node to eventually decide that either it is the leader or it is not the leader, with the additional constraint that *exactly one processor* decides that it is the leader. Formally (see [AW04] Section 3.1):

Definition 2.2.1 *An algorithm is said to solve the leader election problem if it satisfies the following conditions*

- *The terminated states are partitioned into elected and not-elected states. Once a processor enters an elected (respectively, not-elected) state, its transition function will only move it to another (or the same) elected (respectively, not- elected) state.*
- *In every admissible execution, exactly one processor (the leader) enters an elected state and all the remaining processors enter a not-elected state.*

In the following we see how the most important leader election systems used in blockchains work.

2.2.1 Proof of Work

From Definition 2.2.1 we can see that Leader Election is a bit of a misnomer in the Bitcoin PoW protocol. That is because in a decentralized system such as bitcoin a miner is more of a "proposer" of a block. A simple way to understand why miners are proposers and not leaders is the fact that there can be multiple proposers at the same time, so there is no partition of "elected" vs. "non-elected" for each round/block. For example, assume that after a given block B, there are two proposers for block $B+1$, let us call them proposer P_1 and proposer P_2. In this case, the whole set of miners will eventually add blocks on top of the block proposed by either P_1 or P_2, but not both, following the rules explained in this section[6]. The way leader election work in the PoW algorithm used in Bitcoin is explained below.

Bitcoin's PoW is heavily based on Back's work on Hashcash [Bac02]. Hashcash's goal is to block denial of service in e-mail sending. Using current terminology, the goal of Hashcash is to block spamming actors in mass sending of e-mails. To do that, Back proposes a protocol in which the client solves a problem and sends both the e-mail and the problem solution to the server to get the mail sent. If the solution is correct, the server sends the email, otherwise it does not. Assuming the problem is quick to solve, from a minimum of 1 to a maximum of 10 seconds per mail, an honest user experiences no degradation at the service level, while a dishonest user (defined as a spammer who sends $\tilde{1}000$ and e-mails at a time) has losses in terms of time (since sending the whole set of e-mails would take around 2 hours and a half instead of some seconds) and costs (electricity used for computing). This would make the act of spamming unprofitable.

Essentially, Back proposes a cost function efficiently verifiable, but parameterisably expensive to compute. The client is the user that evaluate the function, the server is the participant that verifies that. More formally:

Definition 2.2.2 *An* interactive *Hashcash protocol is a three steps protocol such that:*

- $\mathcal{C} \leftarrow \textbf{Chal}(s, w)$ *is the server challenge function, where s is a bitstring (e.g. the e-mail as in the example above) and w is a witness*
- $\mathcal{T} \leftarrow \textbf{Mint}(\mathcal{C})$ *is the mint function based on the challenge; the result is called* token
- $\mathcal{V} \leftarrow \textbf{Value}(\mathcal{T})$ *is the token verification function*

The **Chal** *and* **Value** *functions are performed by the server, while the* **Mint** *function is performed by the client.*

[6] The distinction between proposer and leader is meaningful only for PoW consensus mechanism.

This protocol formulation, however, cannot work in non-interactive contexts, for example when you don't want to overload the server with a double function or for network requirements you want to decrease the number of messages. In these cases a non-interactive version is used:

Definition 2.2.3 *An* non-interactive *Hashcash protocol is a two steps protocol such that:*

- $\mathcal{T} \leftarrow$ **Mint**(s, w) *is the mint function*
- $\mathcal{V} \leftarrow$ **Value**$(\mathcal{T})$ *is the token verification function*

The **Value** *function is performed by the server, while the* **Mint** *function is performed by the client.*

More generally, "Hashcash is a non-interactive, publicly auditable, trapdoor-free cost function with unbounded probabilistic cost" [Bac02], where:

- **Publicly Auditable**: A cost-function is publicly auditable if **Value** can be efficiently computed (with respect to **Mint**) by any third party without access to any trapdoor or secret information.
- **Trapdoor-Free**: A trapdoor-free cost-function is one where no participant (either client or server) has any advantage in minting tokens
- **Unbounded Probabilistic Cost**: a cost-function has an unbounded probabilistic cost if the probability of taking significantly longer than expected decreases rapidly towards zero.

This generalization is important, since it is possible to create different functions that always represent, in methodology, a proof-of-work. An example of a different method of leader election is EthHash [Fou22a], the leader election algorithm currently used in the proof-of-work of the Ethereum blockchain.

In the following we explain how the **Mint** and **Value** functions work in Bitcoin. Using the notation in [Bac02], let $\mathcal{H}(\cdot)$ be the `SHA256` (collision-free) hash function, so that its output is always a 256-bit string, and define the $\overset{\text{left}}{=}_w$ operation as:

$$\begin{aligned} x \overset{\text{left}}{=}_0 y \quad & [x]_1 \neq [y]_1 \\ x \overset{\text{left}}{=}_w y \quad & [x]_i = [y]_i \forall i = 1 \dots w \end{aligned}$$

In this operation, w represents the required minimum number of left-most 0s of the result of the hash for which two bytes-arrays x and y are considered equal.

Then, the **Mint** and **Value** functions are implemented as:

$$\begin{cases} \mathcal{T} \leftarrow \textbf{Mint}\,(s, w): & \text{find } x \in_n \{0,1\}^* \text{ st } \mathcal{H}(s \| x) \overset{\text{left}}{=}_w 0^{256} \\ & \text{return } \mathcal{T} = (s, x) \\ \mathcal{V} \leftarrow \textbf{Value}(\mathcal{T}): & \mathcal{H}(s \| x) \overset{\text{left}}{=}_w 0^{256} \\ & \text{return } \mathcal{V} \end{cases} \tag{2.1}$$

In the case of bitcoin, s represents the serialization of the block. See Section 7.4.2 for a detailed discussion on Bitcoin blocks.

The number x is called *nonce* and w is called *difficulty*. The reason is that the higher the value w, the smaller the minimum integer number whose byte representation has w's zeros, and therefore the more difficult "finding it" becomes[7] . We explain the concept better in the next section since it gives PoW the capability of being adaptive with respect to the estimated hashpower.

As seen in Equation 2.1, **Value** is efficient to compute with respect to **Mint** having both s and x: the equation is a good cost-function for the purpose.

[7]Recall that collision free hash functions' output is supposedly uniformly distributed with a fixed length of 256-bit, as defined in Definition 5.1.2. Therefore the maximum number is 2^{256}, the probability of "hitting" a specific number a is always $\frac{1}{2^{256}}$ and the probability of "hitting" a number less than b is $\frac{b+1}{2^{256}}$ (since you have to count the all-zeros string). Assuming you write e.g. the number 3 as

$$\underbrace{0 \dots 0}_{254 \text{ zeros}} \; 11$$

it is easy to see that the higher the required left-most zeros w, the smaller the maximum number and consequently the probability of "hitting" a number below it.

Note that in principle two miners can find a solutions for the same block B, e.g. miner M_1 finding solution (s_1, x_1) and miner M_2 finding solution (s_2, x_2). In this case both miners will propose their block: in a following phase (see Section 2.4) it is decided which of the two blocks is integrated in the chain definitively.

Difficulty

As mentioned above the process of leader selection is adaptive thanks to the parameter w that decides the difficulty of finding the nonce x. The difficulty is calculated every $2,016$ blocks. this is the expected number of blocks assuming a block every 10 minutes in 2 weeks So, if d_i is difficulty at epoch i, then the update is[8]:

$$d_{i+1} = d_i * \frac{expected\ time}{real\ time} = d_i * \frac{14\ days}{real\ days}$$

For example, if the real number of days is 10, then

$$d_{i+1} = d_i * \frac{14}{10} = 1.4 * d_i$$

an increment of 40%. The number of days is computed *deterministically* via the use of timestamps (one of the values in each block): each miner performs this computation locally; the honest miners will reach the same result. This is important, since the difficulty value is part of headers[9].

Variations

The easiest way to variate from the Hashcash algorithm is by changing the hash function $\mathcal{H}(\cdot)$ used. Generally the change is related to properties of the functions. For example, Ethash is the proof-of work algorithm used in Ethereum. The algorithm has been designed to be memory hard, meaning that costs significant amount of memory to evaluate, see e.g. [BRZ18]. Goal of this choice is to be ASIC-resistant, thereby discouraging the willingness to build specific electronic components suitable solely and exclusively for hash computation. On the one hand, this inherently lowers the security of the blockchain as it decreases the difficulty, on the other hand ASIC-resistance theoretically favors decentralization as it is not necessary to equip oneself with special equipment to actively participate in consensus[10].

Other proof-of-work systems follow a similar pattern, but they do not variate in the specific goal, but only in the hash functions (or collection of hash functions) used, so we will not cover them here.

2.2.2 Proof of Stake

Differently from the Proof of Work case, the consensus algorithms going under the name of "Proof of Stake" do not have a clear parent.

Proof of Stake (PoS) as a consensus algorithm has been informally presented in a bitcointalk post in 2011 [Qua11]. While reception in the forum was mild (the proposal was to change the proof of work of bitcoin into something different), the proposal sparked a new wave of interest into consensus algorithms, sparking a thorough discussion with multiple analysis on the benefits, pros and cons of both PoW and PoS which undoubtedly are making the whole ecosystem more secure and healthy.

One of the first blockchain projects using PoS was Peercoin [KN12]. As a real life project, it brought up many problems with naive implementation of PoS, such as the "rich get richer" and the "long term" attacks. More details on these problems can be found in [HTC+21]: we will not explain the multiple facets of the issues since to us the leader election phase is interesting only for fault tolerance (Section 2.3) and finality (Section 2.4).

Recall the discussion on difficulty D in Section 2.2.1. Instead of being based on D, PoS uses a different way to created a target, $f(D, stake)$ where $f(D, \cdot)$ is a function that depends on both the

[8] See https://github.com/bitcoin/bitcoin/blob/7fcf53f7b4524572d1d0c9a5fdc388e87eb02416/src/consensus/params.h#L81 for the specific line of code in the Bitcoin Core official repository

[9] https://developer.bitcoin.org/reference/block_chain.html#target-nbits

[10] At the practical level other practical difficulties have made Ethereum's decentralization process difficult. For example, the large (in terms of GBs) ever-expanding blockchain requires storage management that is out of reach for a lot of users. ASIC-resistance, as the fork topic in Footnote 4

difficulty D and the stake *stake* of the validator (the validators perform the same role as the miners do in PoW). Furthermore, all PoS algorithms can be explained as a *lottery* for which a validator has been given a ticket: only if the ticket is "good", the winner propose the block and if accepted by the other validators the block is appended to the blockchain. In the following we present three different PoS algorithms: Single-Lottery PoS, Multi-Lottery PoS and Compound PoS. The main difference is in how f, and therefore the "lottery", is implemented. Different fs give different consequences as seen in [HTC+21]. In all the three algorithms, we will use ellipsis (...) to denote the different variables used in the algorithm, especially in the hashing function H. They generally include the serialized block in different forms, depending on the actual project. Since it is not our goal to analyze the specific on the algorithm, but just the overall working, we omit this discussion.

Single-Lottery PoS

In Single-Lottery PoS [Nxt] the validator is given only a ticket. the ticket is based on a variable called *time* and it is computed as

$$time = basetime \cdot \frac{H(\mathbf{pk}, \ldots)}{stake} \tag{2.2}$$

where **pk** is the public key of the validator, H is a hash function, *stake* is the stake of the validator and *basetime* is a constant.

in this type of PoS-based leader election algorithm, the lower *time* the higher the chance of winning and propose the block. therefore, the higher the *stake* the higher the chance the validator wins.

Multi-Lottery PoS

In Multi-Lottery PoS [Bla, Hom] the validator is given multiple ticket, differently from 2.2.2. The tickets is based on the variable *time* which represents the timestamp of the block. As in Section 2.2.2, the higher the *stake*, the more hashes will be valid. Validity of and it is computed as

$$H(time, \ldots) < D \cdot stake \tag{2.3}$$

In this type of PoS-based leader election algorithm, the lower *time* the higher the chance of winning and propose the block, as in Section 2.2.2.

Compound PoS

This is the PoS algorithm that will be used in Ethereum after the merge phase [Fou22b]. There are some differences between this PoS and the other in Section 2.2.2 and Section 2.2.2. First, both the proposal and validation of blocks are rewarded, instead of the proposer only: the rationale is that validators are basically proposers whose proposal has not been chosen and they should be rewarded anyway for their contribution to secure the network. The block-choice is done randomly. Second, compound-PoS caps the amount of stake every identity can have: every identity/node has *exactly* 32 ethers. Third, the identities are randomly and dis-jointly partitioned into 32 shards and verify transaction in parallel. Sharding has been a topic of research in the Ethereum community from at least 2016 with the creation of Plasma [PB17]. We will not comment on the sharding part of the consensus algorithm, since we are only interested in the leader election phase.

For each mining epoch, miners receive $3 \cdot \mathbf{base} \cdot \mathbf{vote}$ incentives for each identity. The constant **vote** is based on the percentage of time that the identity now has been online and has actively submitted votes. Attesters have a $1/8 \cdot \mathbf{base} \cdot \mathbf{N}$ incentive, where N is the total number of identities assigned to all miners.

Here again we can see how having more stakes (and thus more identities) can benefit an entity that controls more nodes since it can participate in selection with more blocks.

2.2.3 Proof of Authority

Proof of Authority (PoA) is a simple leader election algorithm specified in [Szi17] originally created for single or multiple-nodes testnets. It has been used for private or permissioned projects in settings where all parties know and trust each other since the scheme is based on the idea that blocks may only

be minted by trusted signers. This has a chance of working when nodes are under similar legal laws so that trust is ultimately outsourced to a government.

In PoA every node is indexed in a list of authorized signers. A node can sign only if it belong to the list "and didn't sign recently", as explained in the EIP [Szi17]. Since the list of authorized signers is shared by all parties, the leader election is deterministic. We see why PoA can not be used in consensus methods for permissionless blockchains in Section 2.3.

2.3 Fault Tolerance

Fault tolerance is not directly connected with interoperability concepts, but it must be included into every consensus mechanism which uses a distributed database (and therefore in any blockchain) which needs to stay consistent. In fact, while deciding on a value (i.e. reaching consensus) would be trivial if the participating nodes, processes and network were always reliable, real systems are subjected to possible failures: processes crashing or network partitions are only two of the many things a consensus method has to defend against.

To better explain failures, we divide them into two kinds. The first, "benign", kind occurs when nodes crashes but do not perform wrong operations. The second kind deals with behavior of nodes (e.g. sending different messages when supposed to send the same message to all nodes): these faults are called *Byzantine*.

Recall the difference between a *synchronous* and *asynchronous* system: in the former it is expected that *all* messages will be delivered and reach destination within a deadline δ. The latter kind of systems can not keep this assumption: the internet is the epitome of an asynchronous system, and therefore a blockchain using the internet for message passing (see Section 2.1) is inherently an asynchronous system. This difference is crucial: while it is possible to have consensus for synchronous systems with faulty processes (see Chapter 5 of [AW04]), it is widely known that reaching *deterministic* consensus is not possible for asynchronous systems in case of at least one faulty process, as expressed in the FLP impossibility theorem [FLP85] which we formally introduce in Theorem 7.1.3.

The key property that lets both Proof-of-Work (Section 2.2.1) and Proof-of-Stake (Section 2.2.2) to act as consensus mechanism is the fact that both of them are non-deterministic. This does not happen with Proof-of-Authority (Section 2.2.3): it is easy to see that in case of lack of communication from a node, nodes can not reach consensus.

For example, in Proof of Work, originally it was widely believed that the Bitcoin system would perform globally well if at least 51% of the hashpower was controlled by honest miners [Nak08]. Recently new attacks at the network level have shown that in reality a smaller percentage (one third) of rogue players is enough for the blockchain to suffer globally [ES18, NRS20]. The latter proportion is in line with the general result on the topic (see e.g. [AW04], Theorem 5.7).

2.4 Eventually Stable Consensus

Given how the leader is proposed in Proof of Work based blockchains, the long term acceptance of a block is not deterministic but probabilistic. In this sense, a *stable* consensus is asymptotically reached. As seen in the computation of the Bitcoin whitepaper [Nak08], a user can be 99% sure that a block will be forever stored in the Bitcoin blockchain after 6 new blocks have been mined, i.e. on average after one hour. Numerous proposals has been done to solve this perceived problem, see [DSW14] for an example. Depending on how the Proof of Stake leader-election algorithm works, this is true also for Proof of Stake based blockchains. Interestingly, PoA based blockchains do not have this problem.

Consequently it is not immediate to securely use a blockchain for day-to-day payments: imagine waiting an entire hour for a coffee! This is not just an annoying problem, but also a security concern. In fact, if the steps of an atomic swap are done too fast, without waiting for a stabilization of the consensus, parties risk their funds "disappearing" from a blockchain, with problems similar to those explained in Section 2.1.3.

Time-locks duration (Section 5.2) and waiting periods (Chapter 11) may seem tedious or second-order parameters, but they actually are the very key (sometimes not metaphorical) that makes an atomic swap effectively *atomic*.

Chapter 3

Transaction Models

After seeing how consensus works, let's look at how database update functions work. In the blockchain framework, these functions are called *transactions* since the first ones represented identity changes in the ownership of funds.

Currently there are two major paradigms regarding the transaction model. The first is the one called *Unspent Transaction Model* (or UTXO) where transactions *spends* previous transactions. In this sense the balance of a user is the sum of all unspent transactions received. The second method is the *Account Model*, where each transaction represents a change in database state. In this model, the balance of a user is the sum of assets value associated with his account. This model is more intuitive because it has a clear analogy with the concept of bank account, which we are generally used to.

In the following we see the potentialities and the tradeoffs of these two models. It is important to analyze these two models because they give rise to different management of interoperability methods, as we will see in Section 5.2.

3.1 The UTXO model

3.1.1 Inner Working

The UTXO model is the model used by Bitcoin, the first blockchain. That's why it is (historically) the most important one. Figure 3.1 describes how a transaction in Bitcoin works.

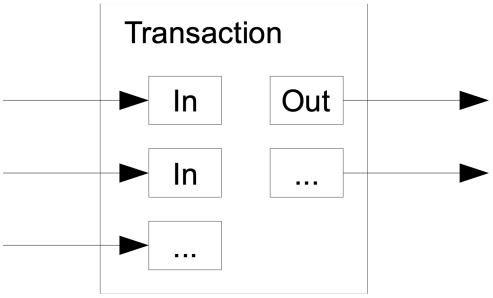


Figure 3.1: Using this figure from the original Bitcoin paper (see page 5 of [Nak08]), Nakamoto described its transaction method.

Each transaction is divided into *Inputs* and *Outputs*. This way you can combine and split previous transactions and get all the values a transaction needs. The combination of inputs and outputs creates the *Transaction Hash*, or *Transaction Id*, which acts as a way to index and recall transactions. Figure 3.2 has a more detailed example of how a transaction works. With reference to Figure 3.2 we now see how to interpret transactions.

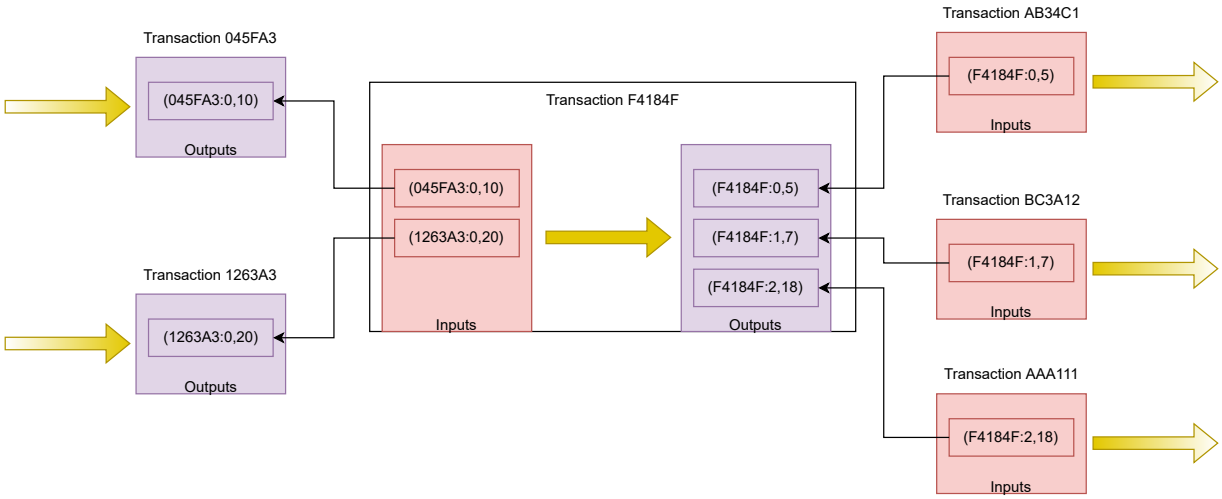


Figure 3.2: The life of transaction with hash F4184F.

A transaction can have one or more than one input and one or more than one output. For example, transaction F4184F[1] in Figure 3.2 has two inputs and three outputs. The inputs of transaction F4184F are derived from the outputs of transaction 045FA3 and transaction 1263A3 respectively. Thus, I say the transaction F4184F *spends* the outputs of transactions 045FA3 and 1263A3. In other words, before transaction F4184F was created and added to the blockchain, the outputs of transactions 045FA3 and 1263A3 were transaction outputs yet unspent, or Unspent Transaction Outputs. Hence the name of the transaction model.

After transaction F4184F is posted, the three outputs `F4184F:0`, `F4184F:1` and `F4184F:2` are available to be spent. As seen in Figure 3.2, for example the output `F4184F:1` becomes the input for transaction BC3A12. This ends the life cycle of a transaction.

Finally, note that since outputs are spent based on an indexing linked to the hash of a transaction, a double spending attempt (i.e. trying to spend a previously spent output) is recognized by all honest nodes since the reference transaction appears as previously spent. Such an attempt is not relayed by any honest node (see Section 2.1.2) and therefore has little chance of propagation.

3.1.2 Usage

So far we have seen what a succession of transactions looks like in a UTXO transaction model, but we have not seen *how* it is actually possible to spend the outputs.

In a UTXO model each output has associated spending conditions. Different spending conditions create different types of outputs. Spending conditions are practically enlisted in a script we can call *OutputScript*. To redeem the output for using it in a transaction as an input, the user has to provide another script we can call *InputScript* such that the evaluation of the whole script *InputScript* + *OutputScript* returns a `True` result.

Below I explain the different types of outputs used in Bitcoin, since all of these will be used to create interoperable protocols. In fact protocols presented in Chapter 9, Chapter 10 and Chapter 11 need a technical knowledge of how the Bitcoin scripting language Script works to be understood.

Bitcoin has undergone a major soft fork relatively recently. Since November 2021, specifically since block 709,632, some basic components of the protocol have changed dramatically. Among these changes are the switch from signatures via ECDSA to Schnorr-type signatures (see Sections 4.2.1 and 4.2.2) and the introduction of new transaction types. Below I briefly list the transaction types providing the

[1] In the course of this chapter I use transition names encoded with six alphanumeric characters. In doing so I am inspired by the hexadecimal representation of hash digests, which have much more than six characters. For example, in Bitcoin, each transaction is indexed by (the result of) a SHA256 hash. Its output is 256 bits, or 32(= 256/8) bytes. Since in hexadecimal representation two characters are necessarily used for each byte, the length (in terms of characters) of a hexadecimal string representing the result of a SHA256 hash is 64, more than ten times the length used in this example.

scripts needed. To understand how Script scripts[2] works I provide a brief example.

Assuming you want to evaluate if $1 + 2$ is equal to 3, a Script script will be written like that:

```
1 2 OP_ADD 3 OP_EQUAL
```

where `OP_ADD` is the *opcode* for addition and `OP_EQUAL` is the *opcode* for checking the equality. An *opcode* is a Bitcoin Script command or function[3]. So in particular `OP_ADD` is a function that takes two inputs and outputs a number, i.e. the sum of the input and `OP_EQUAL` is a function that takes two inputs and outputs a boolean, `True` if the inputs are equal, `False` otherwise.

The evaluation of the script `1 2 OP_ADD 3 OP_EQUAL` can be visualized this way. At Step 0, we have an empty stack. We visualize that this way:

```
script   |  stack
---------+--------
1        |
2        |
OP_ADD   |
3        |
OP_EQUAL|
```

Step 1 of the evaluation is to put 1 on the stack:

```
script   |  stack
---------+--------
2        | 1
OP_ADD   |
3        |
OP_EQUAL|
```

After Step 2 of the evaluation we have this situation:

```
script   |  stack
---------+--------
OP_ADD   | 2
3        | 1
OP_EQUAL|
```

Step 3 of the evaluation puts `OP_ADD` on the stack. This opcode does two things: it removes the last two elements put on the stack (in a LIFO fashion), in this case the numbers 1 and 2. Then the opcode puts on the stack the result of the operation.

```
script   |  stack
---------+--------
3        | 3
OP_EQUAL|
```

Step 4 of the evaluation puts the number 3 from the script column on the stack. Step 5 acts similarly to Step 3: it puts `OP_EQUAL` on the stack. This opcode removes the last two elements put on the stack and puts on the stack the result of the equal operation, in this case `True` since 3 is equal to 3. This is the result after the full evaluation:

```
script   |  stack
---------+--------
         | True
```

In the following we give a quick explanation of the most important transactions in UTXO based projects. These transactions are, unless otherwise specified, common to all projects, such as Bitcoin, Bitcoin Cash and ZCash. It is important to keep in mind that these transactions are possible on multiple blockchains in order to achieve interoperability methods that fit on multiple blockchains.

[2]Unfortunately, the scripting language used by Bitcoin is called Script. This is a Forth-like, stack-based scripting language, processed from left to right in a LIFO (last-in, first-out) fashion. The script is intentionally not Turing-complete for security concerns.

[3]A list of all Script opcodes can be found here: https://en.bitcoin.it/wiki/Script#Opcodes

P2PK Pay-to-Pubkey is the simplest way of making a transaction. It is currently removed as a method due to the fact that it releases the public key in cleartext. In fact, its *OutputScript* is simply:

```
<Public Key> OP_CHECKSIG
```

Trivially, the *InputScript* needed for the whole script to resolve to `True`, is `<sig>`, i.e. signing the transaction with the private key whose public key is `<Public Key>`.

Due to the risks of post-quantum computing and the consequential capability of deriving the private key from the public key, this method of payment is no longer allowed since it leaves the public key in cleartext in public.

P2PKH To solve the problems mentioned before, the Pay-to-PubkeyHash type transaction is used. In this type of transaction, the user who creates the output uses as *OutputScript*:

```
OP_DUP OP_HASH160 <Public KeyHash> OP_EQUAL OP_CHECKSIG
```

Here `<Public KeyHash>` is the double-hashed[4] public key. In practice, to redeem this transaction the redeemer has to provide two inputs:

1. The right public key
2. The right signature of the transaction

That way the *InputScript* is :

```
<sig> <Public Key>
```

It's easy to see that the combination of *InputScript* and *OutputScript* resolves to `True` if variables are correct by knowing that `OP_DUP` is the opcode that duplicates the last element of the stack, this way:

```
script  |  stack            script  |  stack
--------+--------    -->    --------+--------
OP_DUP  | 3                        | 3
                                   | 3
```

and `OP_CHECKSIG` is the operator that verifies if `<Public Key>` "opens" the signature `<sig>`.

I stress a difference between two Bitcoin opcodes since it is useful to explain how some Bitcoin scripts for interoperability work. Bitcoin Script provides two opcode for checking signatures: `OP_CHECKSIG` and `OP_CHECKSIGVERIFY`. The difference between those two is that the first one returns a boolean in any case, while the latter returns `False` in case of error and nothing otherwise. This distinction is important since any Bitcoin script has to evaluate to `True`. Therefore a script such as

```
<sig> <Public Key> OP_CHECKSIGVERIFY
```

is not a valid script since it leaves an empty stack, instead of a stack like this:

```
script  |  stack
--------+--------
        | True
```

Finally let Tx_2 be a transaction that spends a P2PKH transaction Tx_1. As we saw, the *InputScript* of Tx_2 contains the signature of the output of Tx_1. The transaction id of Tx_2 takes into account this signature. Therefore the signature has to be stored on chain for miners to be able to verify the transaction. We will see how the Segregated Witness introduction changes the way the signature is stored on the blockchain.

[4]Specifically the public key is hashed with a SHA-256 hash function and its result is re-hashed with the RIPEMD-160. The reason why two hashes are being used is that it is resistant to a wider range of attacks (e.g. finding a preimage attack on SHA256 does not immediately make all the public-key hashes less secure) and the reason why the second hash is the RIPEMD-160 one is for length (and therefore data) reason: less data in a transaction means less data on a blockchain, which is cheaper for the user and lighter to process for nodes.

P2PSH Script has many opcodes which let you create many types of scripts. For example, it is possible to require knowledge of more conditions than those listed above. Among the most famous are the knowledge of the pre-image of a hash (which we will see and use in Section 5.2) and certain temporal conditions based on the number of blocks. As mentioned before though, the longer a script is, the slower it is to verify. This can create several damages to the network of a blockchain. For example, the Bitcoin network has been the victim of hard-to-verify spam transactions. This led developers to separate standard transactions (i.e. the ones I consider in this section) from non-standard ones [BMS19].

Since it is impossible to predict all possible combinations of opcodes that do not pose a threat to the bitcoin network, BIP-16 [And13] established a standard transaction that allows embedding of any script without the risk of damaging the bitcoin network.

The *OutputScript* of a P2SH transaction is:

```
OP_HASH160 <20-byte-hash of script> OP_EQUAL
```

where `OP_HASH160` is the double hash function we saw in the P2PKH transactions and `OP_EQUAL` is the equality opcode.

To redeem a P2SH transaction, a user has to supply two scripts. I call the first *P2SHout*: this is the script whose hash is equal to `<20-byte-hash of script>`. I call the second script *P2SHin*: this is the script such that *P2SHin* + *P2SHout* would evaluate to true. A *InputScript*+*OutputScript* in this case is of the form:

$$\underbrace{P2SHin + P2SHout}_{InputScript} + OutputScript$$

When the *InputScript* is parsed by the miner, the whole *P2SHout* is considered as the input of `OP_HASH160`

To make clearer the dynamics of the scripts used, I give an example. Suppose the *P2SHout* is

```
OP_SHA256 eb2e10773d403f9972939ede70382fc05c43260b1beb0d5163f3b39dbc8f964b OP_EQUAL
```

The hash of this script is `b3b3ae4c1c7db109e72000a9fdd7275e1ba4cf32`.

This script evaluate to true if somebody provides x such that $SHA256(x) =$`b2e10773...`. The *P2SHin* script is then[5]:

```
389274691823
```

since $SHA256(389274691823) =$`b2e10773...` Note that anyone can enter x, this script does not check who can put in the missing data[6].

The hash of the *P2SHout* script is `b3b3ae4c1c7db109e72000a9fdd7275e1ba4cf32`. Therefore the *OutputScript* is[7]:

```
OP_HASH160 b3b3ae4c1c7db109e72000a9fdd7275e1ba4cf32 OP_EQUAL
```

The *InputScript* in this case is:

```
389274691823 OP_SHA256 eb2e10773d403f9972939ede70382fc05c43260b1beb0d5163f3b39dbc8f964b OP_EQUAL
```

This concludes the transaction.

[5] You can check that the *P2SHin* + *P2SHout* script will work by using the `btcdeb` utility and doing: `btcdeb '[389274691823 SHA256 eb2e10773d403f9972939ede70382fc05c43260b1beb0d5163f3b39dbc8f964b EQUAL]'`

[6] This is an intrinsic security hole: Assume Alice knows x and sends a transaction with x in it. Any miner can read the transaction Alice sent, and a miner Mallory can create a new transaction with x and choose not to include Alice's transaction in the block, who gets her coins subtracted from Mallory's transaction

[7] Note that this is the output of a real transaction I made on testnet as part of a research project. You can see whole hexadecimal value of the script at https://api.blockcypher.com/v1/btc/test3/txs/c8cca62403882df10ba79ebdc7c8327e3a85faad0208ed818cc39343003236a6?limit=50&includeHex=true.

P2WPKH This transaction type works only for the Bitcoin blockchain, since both ZCash and Bitcoin Cash did not activate Segregated Witness [LW16a]. This transaction is the Segregated Witness version of the P2PKH transaction.

Segregated Witness, also abbreviated in SegWit, has been a change in the Bitcoin community which is currently implemented as a soft fork (see Section 2.1.3). SegWit is beneficial in multiple instances. For example it provides a fix for transaction malleability and increase the capacity of a blockchain by increasing the amount of transactions per block. For an informal presentation of the SegWit scheme see [Wui16].

As for this thesis, the main change is due to the different structure of the transaction. In a P2WPKH SegWit introduces the removal of the *InputScript* and introduces the witness fields. At a practical level, the transaction is composed as follows:

```
witness:      <sig> <Public Key>
InputScript:    (empty)
OutputScript: 0 <Public KeyHash>
```

where `<sig>` is the user's signature, `<Public Key>` is the user's public key, `<Public KeyHash>` is the hash of the public key. The `0` in scriptPubKey indicates the following push is a version 0 witness program; the length of `<Public KeyHash>` indicates to miners that this is a P2WPKH transaction. Moving the script from *InputScript* to witness means saving only the public key hash in blockchain.

P2WPKH transaction can be nested into a P2SH transaction. In this case:

```
witness:      <sig> <Public Key>
InputScript:  <0 <Public KeyHash>>
OutputScript: HASH160 <20-bytes-hash of script> OP_EQUAL
```

Although less efficient (the *InputScript* is not empty) the payment address is fully transparent and backward compatible.

P2WSH This transaction type works only for the Bitcoin blockchain, since both ZCash and Bitcoin Cash do not activate Segregated Witness [LW16a]. This transaction the Segregated Witness version of the P2SH transaction. similarly to the case of P2WPKH, the witness fields are introduced and the *InputScript* is move in it.

P2TR This transaction type works only for the Bitcoin blockchain and it is one of the results of the Taproot soft-fork [WNT20]. These transactions can be seen as version 1 of the SegWit kind of transactions

```
witness:      <sig> <Public Key>
InputScript:    (empty)
OutputScript: 1 <Public Key>
```

where `<sig>` is the user's signature, `<Public Key>` is the user's public key. The `1` in scriptPubKey indicates the following push is a version 1 witness program. Note that the `OutputScript` contains the `<Public Key>` , not its digest. As explained in [WNT20], the legacy way of putting a hash digest instead of the public key directly was done to protect against ECDLP breaks or quantum computers. Unfortunately, this protection is very weak since transactions are not protected while being confirmed (a miner verifying a transaction sees both the signature and the public key and it can reverse public key at this moment). Furthermore a large portion of the bitcoin currency's supply is not under such protection, such as all unspent transactions currently under P2PK addresses[8]

[8]For more information and specific numbers about that, see Peter Wuille Twitter thread https://twitter.com/pwuille/status/1108085284862713856.

```
{
     "hash": "0x16d4e7a8f4...",
     "nonce": 0,
     "blockHash": "0x508b487333...",
     "blockNumber": 1,
     "transactionIndex": 0,
     "from": "0x5003B35Ae4...",
     "to": "0x8f48a1AFE4...",
     "value": 1000000000000000000,
     "gas": 2000000,
     "gasPrice": 50000000000,
     "data": " ",
     "input": "0x",
     "v": 28,
     "r": "0x3338d84446...",
     "s": "0x7f00dde6b8..."
 }
```

Listing 3.1: A transaction in the account based model

3.2 The Account model

3.2.1 Inner Working

The account-based transaction model is orthogonal to the UTXO-based model. In this model each address is considered the holder of the money, which is not associated and indexed by the hash of the transaction. There is no concept of input and output, but the concept of account becomes the representation of the entity that owns the funds. In this system, coins are transferred from *Entity A* to *Entity B*. It is not possible to combine funds from different entities natively, although it is possible to do so using smart contracts. This is why it is not necessary to have many types of transactions in an account model: any type of non-standard transaction (here meant as a transaction more complex than a simple transfer of coins between two accounts) can be mediated by a smart contract. I expand on this concept below.

Using the nomenclature used in Ethereum, in an account based blockchain there are two types of accounts: Externally Owned Accounts (EOA) and Contracts Accounts. Each EOA has an ether balance. It can send transactions which are either a transfer of ethers (if less than the balance) or a trigger of a contract code: in the course of this section I call *transfers* the former and *smart contract calls*, or simply *calls*, the latter. Furthermore, an EOA is controlled by private keys and has no associated code. A contract has an ether balance too, but it has no private key and it has associated code whose execution is triggered by calls received from other contracts or EOAs. Yet, as of now, a contract can not initiate a transaction[9]

3.2.2 Usage

Given the distinction between transfers and calls, in this section I see how transfers and contract calls work in Ethereum.

Transfers

The best way to represent a transaction in an account model is using a `key:value` data structure: in Listing 3.1 I present a transfer in the case of Ethereum-like blockchain. As can be seen in the transfer fields, every transfer needs two addresses: the first one to tell where the transfer starts (the `from` field), the second one to tell where it ends (the `to` field). The value of the transfer is specified in

[9]This may no longer be true in the future. At the time this thesis is being written, there is an ongoing debate in EIP-2938 and EIP-4337 about the concept of *Account Abstraction*. If things change in this direction, one of the consequences would be that contracts could initiate transactions without a trigger from an EOA.

```
{
    "to": "0x11f4d0A3c12e86B4b5F39B213F7E19D048276DAe",
    "data": "0xc6888fa1000000000000000000...0003"
}
```

Listing 3.2: A contract call in the account based model

the `value` field. The `gas` and `gasPrice` field are related to the amount of *gas* the user is comfortable with paying for the transfer. Since there is no concept of "already spent inputs" in an account model, each address can surely send more than one transfer. So it follows that it is necessary to know if an address sends coins already due to others. This concept is expressed by the `nonce` field that I explain below.

Every transfer from every address is indexed. Whenever the address `0xb60e8d...` makes a transfer, the client updates the `nonce` field. Each node of the blockchain keeps track of the `nonce` field for each address, so if the address `0xb60e8d...` makes two transfer with the same `nonce`, it will be detected.

The last three values of the transfer (`v`,`r` and `s`) are the values of the signature of the transfer. In particular, they refer to the ECDSA signature scheme, the Ethereum current digital signature scheme. The `r` and `s` values are the signature, while `v` is the recovery value. I give more details in Section 4.

Contract Calls

A contract call is conceptually similar to a transfer: a similar transaction as the one in Listing 3.1 is created and mined when users do a contract call. I highlight here the few differences.

A contract call can be either a transaction to a function of an already existing contract or a deployment of a new one. In the latter case , the `to` field has always the same all-zeros address `0x000...000`. This is the particular way in which a user of the Ethereum compatible blockchain deploys a new smart contract. On the other hand, in case of a call to a function of an existing smart contract, the `to` address field is filled with the smart contract address. The particular function called is passed with the `data` field, which in those cases is not empty, see Listing 3.2 for an example.

Chapter 4

Digital Signatures

Digital signatures schemes are probably the most important piece of the blockchain stack. The main, and well-known, reason is that (digital) signatures enable the sending of blockchain transactions. In fact, both UTXO and Account based blockchain rules require users to sign their transactions to prove knowledge of the private-key linked to the address. Signing and verifying messages and transactions makes distributed and authenticated data-updating possible. And that's only one part of the many possibilities this cryptographic primitive enables.

In the following, I explain the numerous ways a digital signature scheme can be expanded to obtain interoperability. The reader should consider this chapter as a small introduction on the state of the art on digital signatures schemes for blockchain interoperability. Chapters 9 and 10 will present practical applications of some of the concepts introduced here.

The rest of the chapter is so organized: in Section 4.1 I briefly introduce the notation I use throughout the chapters related to this topic; in Section 4.2 I introduce the most important signatures regarding our topic, namely the ECDSA, the Schnorr and the BLS digital signature schemes. In Section 4.3, I describe the first (chronologically) variation used to expand blockchain transaction capabilities, namely multisignatures; in Section 4.4 I introduce Threshold Signatures and its improvements over naive multisignatures.

4.1 Notation and Definitions

I introduce the notation by giving a brief summary of how a general digital signature scheme works. I explain that in the context of Elliptic Curve Cryptography (ECC), following the Standards for Efficient Cryptography 2: Recommended Elliptic Curve Domain Parameters [Bro10]. For more details in a general context the interested reader can see Chapter 6 of [Gol04].

Let $\mathbb{F}_p$ a field whose order is p. The elliptic curve domain parameter over $\mathbb{F}_p$ are a sextuple so defined:

$$T = (p, a, b, G, n, h) \tag{4.1}$$

where p specifies the finite field $\mathbb{F}_p$, $a \in \mathbb{F}_p$ and $b \in \mathbb{F}_p$ specify the elliptic curve $\mathcal{C}(\mathbb{F}_p)$ with equation:

$$\mathcal{C} : y^2 \equiv x^3 + ax + b \pmod{p} \tag{4.2}$$

If the field $\mathbb{F}_p$ is obvious from context, I drop it and write $\mathcal{C}$ only. $G_{\mathcal{C}} = (x_G, y_G)$ is a point on $\mathcal{C}$. I call $G_{\mathcal{C}}$ the *base* (or *generator*) point of $\mathcal{C}$: such a name is motivated by the fact that G is used to create the public key starting from the private key. If the curve $\mathcal{C}$ is obvious, I drop the subscript and I write G. $n_{\mathcal{C}}$ is the order of G.

Finally, h is a number called *cofactor* and $h = |\mathcal{C}|/n$. This number is always an integer since n, being the order of G, is also the cardinality of the subgroup generated by G and by Lagrange theorem h must be an integer[1].

[1]Since all groups I will deal with in this thesis are finite, such a statement is valid. Of course if one of the groups is not finite, h could be potentially infinite. Since we are not dealing with such edge cases and since I are interested in practical applications of the elliptic-curve concepts, I do not delve in such details.

We can put a special operation on $\mathcal{C}$ which acts as a sum. I abuse the notation and denote the operation on $\mathcal{C}$ with $+$, so that $(\mathcal{C}, +)$ becomes a group[2]. In the context of an elliptic curve, I use letters such as $x, a, b, c, \ldots$ as elements of $\mathbb{F}_p$ and capitalized letters like $X, A, B, C, \ldots$ as points on $\mathcal{C}$. using this notation then, I can call C the sum of points A and B:

$$C = A + B \tag{4.3}$$

and I can define an external product as multiple iteration of the sum in (4.3) of a point on itself as:

$$xA = \underbrace{A + \ldots + A}_{x \text{ times}} \tag{4.4}$$

In the context of blockchains a (digital) signature scheme can be generally defined in the following way

Definition 4.1.1 *A signature scheme on an elliptic curve with parameters T (as defined in 4.1) is a triple* $(\mathbf{KeyGen}, \mathbf{Sign}, \mathbf{Ver})$ *such that:*

- *The key generation* $\mathbf{KeyGen}(1^l)$ *is a PPT algorithm that outputs a key-pair* (x, X) *where* $x \in \mathbb{F}_p$ *is the secret key and* $xG = X \in \mathcal{C}$ *is the public key*
- $\forall (x, X) \leftarrow \mathbf{KeyGen}(1^l)$ *and* $\forall m \in \{0,1\}^*$ *the PPT algorithm* **Sign** *and the deterministic verification algorithm* **Ver** *satisfy*

$$Pr[\mathbf{Ver}(X, m, \mathbf{Sign}(x, m)) = 1] > 1 - \mu(l)$$

Here l is the security parameter, $\mu(l)$ a negligible function in l (e.g. $\mu(l) = 2^{-l}$) and m is a message.

Generally, I write $\sigma \leftarrow \mathbf{Sign}(x, m)$. Note that the digital signature schemes used in blockchain projects are length-restricted signature schemes using the "hash-and-sign" paradigm (see e.g. Section 6.2 of [Gol04]). Consequently, the specification of a signature scheme should also specify a hash function.

4.2 Signatures

In this section I explain the functioning of different digital signature algorithms. The goal is the presentation of the algorithm, its advantages and disadvantages with respect to the uses in the blockchain. Obviously, I cannot present all existing digital signature schemes. Here I decided to present only those that have a direct use in blockchain projects involved in interoperability protocols that I will present in Part II of this thesis. This also explains the detailed treatment: it is necessary to understand the inner working of the different signature schemes in order to appreciate the differences in Multi (Section 4.3) and Threshold (Section 4.4) Signature Schemes

4.2.1 ECDSA Digital Signatures

I start with this algorithm because it is the first to be used, as it is part of the Bitcoin protocol. The same scheme is also used in Ethereum.

Parameters definition

The $\mathcal{C}$ curve (see Equation 4.2) used in the ECDSA protocol is the `secp256k1` defined in [Bro10], Section 2.4.1. As per [Bro10], the sextuple $T = (p, a, b, G, n, h)$ is:

$$\begin{aligned} p &= \texttt{FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFFC2F} \\ &= 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1 \end{aligned}$$

The parameters a and b of the curve $\mathcal{C} : y^2 = x^3 + ax + b$ over $\mathbb{F}_p$ define:

[2] This explains also why I could say that the point G *generates* a subgroup of $\mathcal{C}$.

$$\mathcal{C} : y^2 = x^3 + 7 \tag{4.5}$$

i.e., the curve in Equation 4.2 of parameters $a = 0 \in \mathbb{F}_p$ and $b = 7 \in \mathbb{F}_p$. The base point G in compressed form is:

$$G = \texttt{02 79BE667E F9DCBBAC 55A06295 CE870B07 029BFCDB 2DCE28D9 59F2815B 16F81798} \tag{4.6}$$

and its uncompressed form is: [3]

$$G = \texttt{04 79BE667E F9DCBBAC 55A06295 CE870B07 029BFCDB 2DCE28D9 59F2815B 16F81798}$$
$$\texttt{483ADA77 26A3C465 5DA4FBFC 0E1108A8 FD17B448 A6855419 9C47D08F FB10D4B8} \tag{4.7}$$

You can see from Equation 4.6 and Equation 4.7 that the difference is that the compressed form of G only specifies x_G (the first eight octet-strings) while the uncompressed form of G also specifies y_G (having sixteen octet-strings).
It is important to note that it is possible for any point to have two equivalent representations; here I explain this using G as an example: this is important when talking about public key encryption which in turn is related to the data found on the blockchain. More details can be found in [Bro09] Sections 2.3.3 and Sections 2.3.4.

It is possible to switch from the uncompressed form of G to its compressed form by removing the octet-strings relative to y_G and changing the prefix from `04` to `02` or `03` based on whether the number y_G is even or odd respectively. Since the curve is asymmetrical with respect to the x-axis but symmetrical with respect to the line parallel to the x-axis and passing through the point $(0, b) = (0, 7)$, the equality is an index of the sign, i.e. if y_G is even, then $-y_G$[4]. The reverse is possible using Equation 4.5 (and changing the prefix from `04` to `02` or `03`, again according to evenness).

Finally the order n of G and its cofactor are:

$$n = \texttt{FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE BAAEDCE6 AF48A03B BFD25E8C D0364141}$$
$$h = 1$$

Algorithm

After having seen what the parameters used in Bitcoin and Ethereum are, and where the objects are taken from, we now see how the triplet defined in Definition 4.1.1 works in ECDSA. I follow the protocols described in the ANSI X9.62 ECDSA [JMV01].

KeyGen This algorithm is described in Section 6.1 of [JMV01]. Let A be an entity. A's key-pair is associated by context[5] with the parameters of T (see Equation 4.1).

The key pair generation algorithm **Gen** is a two steps algorithm:

1. Select a random or pseudorandom integer $d \xleftarrow{\$} \mathbb{Z}_n$
2. Compute $Q = dG$.

The keypair is (d, Q) where d is referred to as the *secret key* and Q is referred to as the *public key*.

Signing Probabilistic signature generation and signature verification is described in Section 7 of [JMV01]. There is another *deterministic* way of signing based on RFC 6979 [Poe13]. I start by describing the probabilistic method assuming domain parameters T, a message m and keypair (d, Q). The signer performs:

[3] it is not clear why this particular G has been chosen. Unofficially, that elliptic curve point has interesting properties. For an informal discussion see this link

[4] The sign "$-$" is to be understood as the inverse operation of the sum on elliptic curves and not the normal subtraction between numbers. In this case then $-y_G$ is the second component of the point $-G$, and *not* literally the number $-y_G \in \mathbb{Z}$.

[5] Parameter association can be assured either cryptographically (e.g. with certificates) or by context (e.g. when all entities use the same domain parameters). The latter is the case for every blockchain project.

1. Select a random or pseudorandom integer $k \xleftarrow{\$} \mathbb{Z}_n$
2. Compute $kG = (x_k, y_k)$
3. Compute $r = x_k \pmod n$. If $r = 0$ then go to step 1
4. Compute $k^{-1} \pmod n$
5. Compute $e = \mathcal{H}(m) \pmod n$, where $\mathcal{H}(\cdot)$ is a hashing function[6]
6. Compute $s = k^{-1}(e + dr) \pmod n$. If $s = 0$ then go to step 1

The signature is (r, s).

Verifying A receiver of a signature (r, s) has to verify if it is valid for message m and public key Q. In order to do that, the receiver performs the steps explained in this protocol:

1. Verify that both r and s are in $\mathbb{Z}_n$
2. Compute $e = \mathcal{H}(m) \pmod n$, where $\mathcal{H}(\cdot)$ is a hashing function[7]
3. Compute $w = s^{-1} \pmod n$
4. Compute $u_1 = ew \pmod n$ and $u_2 = rw \pmod n$
5. Compute $X = u_1G + u_2Q$
6. If $X = \mathcal{O}$[8], then reject the signature. Otherwise, $X = (x_X, y_X)$ and compute $v = x_X \pmod n$
7. Accept the signature if $v = r$

The reason why the verification works is that $(r, s) = (r, k^{-1}(e + dr)) \pmod n$ and therefore

$$k \equiv s^{-1}(e + dr) \equiv s^{-1}e + s^{-1}rd \equiv we + wrd \equiv u_1 + u_2d \pmod n$$

Given that $u_1G + u_2Q = (u_1 + u_2d)G = kG$, then $v = r$.

Analysis

Randomicity: The random generation of k can create problems that decrease the security of the ECDSA system. In fact if the implementation is not well done and the number k is used more than once with messages $m_1, m_2, m_1 \neq m_2$ from the same entity A, the receiver of the signatures can get the private key by verifying that (note that if the receiver has m_1 and m_2 then he is able to get also e_1 and e_2 because of step 5 in the signing algorithm):

$$\frac{s_2e_1 - s_1e_2}{r(s_1 - s_2)} = d \tag{4.8}$$

Therefore RFC 6979 [Poe13] standardizes a deterministic way of generating k that replaces the 1 step with:

$$k = \mathcal{H}'(m)$$

where $\mathcal{H}'$ is a hashing function different from $\mathcal{H}$, but from the same family of $\mathcal{H}$[9].The use of RFC 6979 has no influence on the signing algorithm, as seen below.

Malleability: ECDSA signatures are inherently malleable since if (r, s) is a valid ECDSA signature for a given message and key, then $(r, n - s)$ is also valid for the same message and key, where $r\,, s\,, n$ are defined as in the algorithm description. A straightforward consequence of malleability is that

[6] The result of a hash is usually in bytes, but it is always possible to convert bytes to numbers. For this reason I assume that $\mathcal{H}(\cdot)$ is a hashing function with built-in conversion, see Section 5.1 for more details

[7] See footnote 6

[8] $\mathcal{O}$ is the neutral point of the curve $\mathcal{C}$, i.e. the point such that $A + \mathcal{O} = A$ for each A. Given its position it is also called the *point at infinity*

[9] For more details see Section 3.2 of [Poe13] where we see that k is created using the hash of the message made with $\mathcal{H}$ concatenated with other constants, and Section 5.1 of this thesis that explains the concept of hash-function family.

a third party can alter an existing valid signature for a given public key and message into another signature still valid for the same key and message, without access to the secret key. While not a widespread vector of attack, malleability is at least one of the reasons the MtGox exchange had to file for bankruptcy and many people lost funds [DW14]. Since then, the Bitcoin network mitigated this problem assuming only one of those variants as valid, namely the one with *lowest possible* s (see [LW16b]): under these assumptions and other non-standard ones (more details in Section 4.2.2), it is possible to prove ECDSA is non-malleable [Fer18]. Similarly, the Ethereum network considers as invalid any s-value greater than $n/2$, as explained in EIP-2 implemented with the Homestead hard fork [But15].

4.2.2 Schnorr Digital Signatures

Schnorr signatures are a long-known protocol to sign, but it has received little attention so far. One of the reasons this has happened is that until 2008 the signing scheme was under patent. This has led to sporadic use and multiple attempts at standardization that have never been completed [10].

The signature scheme is presented by Schnorr in a numeric field (e.g. $\mathbb{Z}_p$, with p prime), but there is nothing to prevent it from being applied to elliptic curves since the underlying problem, the difficulty of computing the discrete logarithm, works in elliptic curves as well.

Below I define the parameters and the algorithm as they are defined in Bitcoin [WNR20]. The version of Bitcoin can be considered final since it came into effect with the introduction of Taproot in November 2021.

Parameters definition

The parameters used to create the Schnorr signatures are the same as those in Equation 4.1. In this way it is possible to keep the same assumptions about the security of the curve and the hashes. From [WNR20]:

> By reusing the same curve and hash function as Bitcoin uses for ECDSA, we are able to retain existing mechanisms for choosing secret and public keys, and we avoid introducing new assumptions about the security of elliptic curves and hash functions.

In this regard, it is important to note that the introduction of Schnorr signatures *does not replace* the ECDSA signature scheme, but the signature scheme is introduced as a preferred alternative. In fact, if the Schnorr signature scheme replaced the ECDSA signature scheme, unspent transactions created with ECDSA (i.e. all transaction before Taproot) would no longer be valid and redeemable. This would effectively be a theft of funds or create a hard fork. Consequently, the keypair is created in the same way as in the ECDSA algorithm, and I use d as the private key and Q as the public key.

Finally, note that the algorithm uses *tagged hashes*. This is a common method to create a family of hashes starting from a hash function, similar to the family of hashes used in Footnote 9 of this chapter. For example, from hash $\mathcal{H}(\cdot)$ it is possible to create hashes $\mathcal{H}_1(\cdot)$ and $\mathcal{H}_2(\cdot)$ such that[11] $\mathcal{H}_1(x) \neq \mathcal{H}_2(x), \forall x$ by doing

$$\mathcal{H}_i(\cdot) = \mathcal{H}(tag_i||\cdot),\ i = 1, 2$$

where $||$ is the concatenation of bytes and tag_i is a byte-string. In the following I will talk about $\mathcal{H}_1(\cdot)$ or $\mathcal{H}_2(\cdot)$ assuming that there exist a tag.

Algorithm

KeyGen The key creation algorithm for Schnorr signatures is identical to the key creation algorithm for ECDSA signatures.

[10] see https://crypto.stackexchange.com/a/50202 for a discussion of attempts to standardize the Schnorr signature scheme for elliptic curves

[11] See Definition 5.1.2 to understand how inequality is intended.

Signing As per [WNR20], on input (d, m, aux) where d is the private key, m is the message and aux is auxiliary random data, the signing algorithm is:

1. Compute $t = d \bigoplus aux$
2. Compute $k = \mathcal{H}_1(t||Q||m)$
3. Compute $R = k \cdot G$ and $r = x_R$[12]
4. Compute $e = \mathcal{H}_2(R||P||m)$
5. Compute $s = k + ed$

and the signature is $sig = (r, s)$. Note that this is different from the original Schnorr paper [Sch91] where the signature is $sig = (e, s)$. This difference allows optimizations that result in improved performance especially in the case of signature verification, which we see below.

Verifying As per [WNR20], on input (Q, m, sig)

1. Let $r = sig[0]$
2. Let $s = sig[1]$
3. Compute $e = \mathcal{H}_2(r||P||m)$
4. Compute $R = s \cdot G - e \cdot P$
5. Accept the signature if $x_R = r$

This is different from the signature verification method illustrated in Schnorr's original paper [Sch91]. The change is due to the fact that this method gives the ability to verify transaction signatures in batches instead of sequentially. This new optimization results in a single signature of many messages, instead of one signature per message, improving the complexity class of the operation.

Analysis

Randomicity The version of these signatures allows not having to use randomness in its default implementation. This is a greater security than the default algorithm used by ECDSA that as we have seen previously if not implemented properly creates a possibility of attack that in the worst case can recover the private key. With this default algorithm to create the signatures, the randomness is created before the signature and given as input to the signing algorithm, instead of being created during the signature itself.

Of course, it is possible to modify the default algorithm to make it analogous to the ECDSA algorithm (signatures in this case would still be considered valid), but this would be a weakening of its security.

Batch Verification Unlike the ECDSA-based signature method, the Schnorr variation allows you to do one verification when you have multiple signatures, as already mentioned. This is the typical case of verifying multiple signatures when validating an entire block of transactions. In case all transactions within the block are valid, the verification algorithm succeeds. This happens when a miner has to verify the block created by another miner (where the probability that the block is invalid is low). It therefore makes sense to have verification optimization at this level.

Security The security is "higher" than that of ECDSA. In fact, as described in BIP340 [WNR20], Schnorr signatures are strongly unforgeable under chosen message attack (SUF-CMA) in the random oracle model assuming the hardness of the elliptic curve discrete logarithm problem (ECDLP) [PS00a, KMP16] and in the generic group model assuming variants of preimage and second preimage resistance of the used hash function [NSW09] [13].

[12] See Section 4.1 to understand how points are represented.

[13] While the security is proved with the version $sig = (e, s)$, the proofs are still valid with the version $sig = (r, s)$. This is possible since a bijection map from (R, s) to (e, s) is efficiently computable converting the SUF-CMA attacker of the (e, s) variant one for the (R, s) variant. Further details can be found in https://github.com/bitcoin/bips/blob/master/bip-0340.mediawiki#cite_note-2

4.2.3 BLS Digital Signatures

BLS signatures [BLS04] are currently discussed in the Ethereum community[14] as a possible choice for transaction signing. The work started for Ethereum 2.0 in relation with the transition to proof of stake consensus algorithm with sharding. As explained by Drake [Pra18] ECDSA signatures are too slow to verify in the event of a vote with more than 300 thousands voters. This is a practical situation that will happen in a Casper finality voting. BLS signatures have been identified as a short/medium term solution[15] since their short size signatures and easiness signature aggregation. BLS signatures will not substitute the current ECDSA digital signature scheme since, as with the Taproot change in Bitcoin, it would represent a stealing of the funds of the users.

As in the previous sections I give a brief explanation of how the signature work and the choice of parameters. The process of creation of the standard is currently under development since there was not a standard before the proposal of its use in the Ethereum blockchain. Unless otherwise specified, the notation, algorithm and parameters are taken from the RFC on BLS signatures [BGW+20].

Pairing Operation

Differently from the previous digital signature schemes I described, the BLS digital signature scheme uses a different primitive in order to obtain the signature. Pairing (or Bilinear Maps) is a elliptic curve operation that has received attention due to its flexible and practical functionalities. It isn't proposed only in digital signatures but this operation is currently in practical use for other cryptographic protocols such as identity based encryption or attribute based encryption [66613].

Goal of the subsection is to present the pairing operation and give a brief history of its creation. I will indicate the pairing with $e(\cdot)$ or `pairing`.

Definition 4.2.1 *Let $(G_1, +), (G_2, +)$ and $(G_T, \cdot)$ be three groups, non necessarily different. Then a pairing (or bilinear map) is a binary operation $e : G_1 \times G_2 \to G_T$ such that*

1. $e(P + R, Q) = e(P, Q) \cdot e(R, Q)$
2. $e(P, Q + S) = e(P, Q) \cdot e(P, S)$

I only consider pairings between groups of large prime order r, which are non-degenerate, i.e. for which there exists $P \in G_1$ and $Q \in G_2$ such that $e(P, Q) \neq 1$ and that are efficient.

It's easy to see that the conditions 1 and 2 are linearity with respect to the first and second variable respectively, hence the name *bi*linear maps. Also, note that I used two different operations for the groups, but that is not needed. In the following I give an example with simple numbers.

Example 4.2.1 *Let the bilinear map be defined as: $e(x, y) = 2^{xy}$. Then, I can see:*

$$e(3, 4 + 5) = 2^{3 \cdot 9} = 2^{27}$$
$$e(3, 4) \cdot e(3, 5) = 2^{3 \cdot 4} \cdot 2^{3 \cdot 5} = 2^{12} \cdot 2^{15} = 2^{27}$$

In this case $G_1 = G_2 = (\mathbb{Z}, +)$ and $G_T = (\mathbb{Z}, \cdot)$, where '$+$' and '$\cdot$' are the usual sum and product operations in $\mathbb{Z}$.

We can see from this definition a simple property. By seeing that

$$e(P, 2Q) = e(P, Q) \cdot e(P, Q) = e(P, Q)^2$$
$$e(3P, Q) = e(P, Q) \cdot e(P, Q) \cdot e(P, Q) = e(P, Q)^3$$

I can conclude that for $a, b \in \mathbb{Z}$:

$$e(aP, bQ) = e(P, Q)^{ab} \tag{4.9}$$

[14] The Ethereum blockchain is not the only project that is interested in this digital signature scheme. Other projects are Algorand, the Chia Network and DFINITY. This is also evidenced by the fact that researchers from these projects are listed as co-authors of the RFC that standardizes BLS signatures, see [BGW+20]

[15] It cannot be a permanent or long term solution since BLS signatures are not quantum resistant. Differently from Bitcoin, address reuse is highly common due to its account-based transaction model. This way, public keys are reused often and the user is at risk of loosing funds if a commercial quantum computer is created. This doesn't happen in the Bitcoin blockchain, because the public key is showed only after spending a transaction and never reused, effectively leaving the address empty. See Section 3.1.2.

Parameters Definition

In pairing based digital signature schemes the parameters used are a little different from the digital signatures schemes that rely on the addition of elliptic curve points. So for this reason, the mapping of parameters to a sextuple T as defined in Equation 4.1 is not straightforward, even if possible. Given the little theoretical utility of such reparameterization, I decided to describe the parameters intuitively.

In the BLS digital signature scheme as used by Ethereum, the curve $\mathcal{C}$ is the BLS12-381, whose name comes from the fact that the embedding degree $k = 12$ and $r = 381$ a prime (see [BN06a]). More formally

$$
\begin{aligned}
p &= \texttt{1A0111EA 397FE69A 4B1BA7B6 434BACD7 64774B8 4F38512BF 6730D2A0 F6B0F624 1EABFFFE} \\
&\quad \texttt{B153FFFF B9FEFFFF FFFFAAAB} \\
r &= \texttt{73EDA753 299D7D48 3339D808 09A1D805 53BDA402 FFFE5BFE FFFFFFFF 00000001} \\
a &= 0 \\
b &= 4 \\
k &= 12 \\
E_1 &= E(F_p) = y^2 = x^3 + b \\
F_{p^2} &= F_q[i]/(x^2 + 1) \\
E_2 &= E(F_{p^2}) = y^2 = x^3 + 4(i + 1)
\end{aligned}
$$

As in the case of Schnorr signatures in Bitcoin, the BLS digital signatures will not replace the old ECDSA which continues to be used along side the new scheme. See the Analysis to learn more about the reasons behind the choice of flanking BLS signatures scheme to ECDSA.

There are two variants of this digital signature scheme: Minimal-signature-size and Minimal-pubkey-size. The former variant puts signatures as points in G_1 and public keys as points in G_2: the name comes form the fact that E_1 is "smaller" (has lower cardinality than)E_2, and consequently G_1, G_2 (being subgroups of E_1 and E_2 respectively) have different representation.

The latter variant (Minimal-pubkey-size) puts public keys as points in G_1 and signatures as points in G_2.

Algorithm

```
1     SK = KeyGen(IKM)
2
3     Inputs:
4     - IKM, a secret octet string. See requirements above.
5
6     Outputs:
7     - SK, a uniformly random integer such that 1 <= SK < r.
8
9     Parameters:
10    - key_info, an optional octet string.
11      If key_info is not supplied, it defaults to the empty string.
12
13    Definitions:
14    - HKDF-Extract is as defined in RFC5869, instantiated with hash H.
15    - HKDF-Expand is as defined in RFC5869, instantiated with hash H.
16    - I2OSP and OS2IP are as defined in RFC8017, Section 4.
17    - L is the integer given by ceil((3 * ceil(log2(r))) / 16).
18    - "BLS-SIG-KEYGEN-SALT-" is an ASCII string comprising 20 octets.
19
20    Procedure:
21    1. salt = "BLS-SIG-KEYGEN-SALT-"
```

```
2. SK = 0
3. while SK == 0:
4.     salt = H(salt)
5.     PRK = HKDF-Extract(salt, IKM || I2OSP(0, 1))
6.     OKM = HKDF-Expand(PRK, key_info || I2OSP(L, 2), L)
7.     SK = OS2IP(OKM) mod r
8. return SK
```

Listing 4.1: Deterministic Secret Key generation starting from a secret octet string IKM. Taken from Section 2.3 of [BGW+20]

I present here the algorithms in a formal way, as presented in the RFC, but leaving out the technicalities that weigh down the discussion without giving a theoretical contribution. For example, in Listing 4.1 you can see that to create the private key the algorithm uses specific random bytes that encoded in ASCII generate the string `BLS-SIG-KEYGEN-SALT-` and HMAC-based[16] Key Derivation Functions specified in other RFCs. In this discussion I will semi-empirically assume that "a pseudo-random function exists" that creates the private key. To see a semi-formalization of the algorithm, see [Pra18].

KeyGen This algorithm is similar to the previous keygen algorithms, but having two variants I have to differentiate between the two. In particular, I put G as the generator elliptic point. When the signature variant is minimal-signature-size, G is the distinguished point G_2 that generates the group $\mathbb{G}_2$ (see previous paragraph). When the signature variant is minimal-pubkey-size, G is the distinguished point G_1 that generates the group $\mathbb{G}_1$.

The key pair generation algorithm **KeyGen** is:

1. Select a random or pseudorandom integer $d \xleftarrow{\$} \mathbb{Z}_n$
2. Compute $Q = dG$.

The keypair is (d, Q) where d is referred to as the *secret key* and Q is referred to as the *public key*.

Sign This operation is called `CoreSign` in the RFC to differentiate it from the aggregation signature algorithm. As usual, the signing algorithm takes a message m and the private key d. The algorithm for signing is:

1. $H = \texttt{hash\_to\_point}(m)$
2. $R = d * H$
3. $sig = \texttt{point\_to\_signature}(R)$

`hash_to_point` is a hash function that takes an arbitrary string as input and returns a point on an elliptic curve[17]. This RFC draft [FHSS+22] explains how to implement such a function. `point_to_signature` is a function that project the input (an elliptic-curve point) to the right curve, depending on the variant: if the variant is minimal-signature-size, it projects the point to the curve E_1; if the variant is minimal-pubkey-size, it projects the point to the curve E_2.

Verify This operation is called `CoreVerify` in the RFC to differentiate it from the aggregation verification algorithm. As usual, the verifying algorithm takes a message m and the public key Q and a signature sig. The algorithm for verifying is:

1. $R = \texttt{signature\_to\_point}(sig)$
2. $x_Q = \texttt{pubkey\_to\_point}(Q)$

[16] HMAC stands for hash-based message authentication code. It is a cryptographic authentication technique that uses a hash function and a secret key generally used to achieve authentication or to verify that data is correct and authentic.

[17] Since the map is composed of long but easy to understand operations involving very large constants, I have omitted this technicality. The interested reader may look at Section 8 and Appendices E2 and E3 of [FHSS+22]

3. $H = \texttt{hash\_to\_point}(m)$
4. $C_1 = \texttt{pairing}(H, x_Q)$
5. $C_2 = \texttt{pairing}(R, G)$
6. Accept the signature if $C_1 = C_2$

Similarly to the signing algorithm, the verification algorithm uses multiple sub-routines. The pairing operation has been already defined in Definition 4.2.1, and so is the `hash_to_point` function. The functions `signature_to_point` is the inverse of `point_to_signature`, accounting for the two variants of the BLS signature. Finally, the `pubkey_to_point` is defined in RFC and is similar to `signature_to_point`: it projects the point to the "right" curve; curve E_2 in the minimal-signature-size variant and curve E_1 in the minimal-pubkey-size variant.

Analysis

As mentioned, the main use case of BLS signatures in the Ethereum domain is signature-aggregation for sharding, a performance matter. This is different from the reasoning of the Bitcoin case, where ECDSA was flanked by Schnorr signatures for security reasons.

Since the reason is performance, the choice of the curve matter. The BLS signature uses a Barreto-Naehrig curve, specifically the BLS12-381, a family of curves which are deemed paring-friendly since the pairing operation can be optimized substantially [BN06a]. Specific details on the choice of the parameters can be found in the "BLS aggregation performance" section of [Pra18].

4.3 Multi-Signatures

With multisignatures (also called "multisig") I mean those protocols that let multiple people co-sign the same message by appending multiple signatures[18]. More formally, a message is considered k-of-n multi-signed when at least k users among n have signed it, and appended their signatures.

While such a method has been used for thousands of years to manage access to physical spaces[19], its use in computer science is relatively recent. In the following I explain how multisignatures work in Bitcoin (Section 4.3.1) and Ethereum (Section 4.3.2). After that I present the major interoperability application of this system.

4.3.1 Bitcoin

Transactions with multiple signatures in the case of Bitcoin are very simple and that is why they appeared relatively early. In fact to make a k-of-n ($k \leq n$) multisignature inside Bitcoin users add multiple signatures, using the following *OutputScript* (see 3.1).

```
k {pubkey}...{pubkey} n OP_CHECKMULTISIG
```

For example in the case of a 2-of-3 multisignature, to sign the input users place two signatures side by side in the *InputScript* and the `OP_CHECKMULTISIG` opcode is able to execute the necessary code to verify that both signatures are valid. Of course, it is also possible to put three signatures in this case, and the code is able to correctly interpret the sequence.

Multisignatures were standardized in BIP 11 [And11b].

[18] The name is not technical and many different assumptions exist regarding what is a multisignature. Sometimes a multisignature is any kind of signature that involves more than one party to sign: in this case threshold signatures (Section 4.4) would be a kind of multisignature too. In this thesis I stick with the blockchain-related nomenclature based on the amount of signatures that appear on blockchain.

[19] Orthodox Christian Monks at Mt. Athos are known to use a multi-key based special lock for historically important rooms

4.3.2 Ethereum

In the Ethereum blockchain, multisignatures are implemented using a smart contract. the reason is that it is not possible to sign the same transaction with multiple accounts, since Ethereum (and any Ethereum-like blockchain) has an account based transaction model. Using a smart contractor to make this kind of signatures has required multiple iterations and it still is not as secure as in Bitcoin[20].

An example of a multi-signature contract gone wrong is the contract involved in the the Parity hack [Pal17]. Currently many ethers are locked in the smart contract and non-spendable: to be able to spend them a hard fork of the net is required.

As of today, one of the most famous multisignature smart contracts on Ethereum is this smart contract template managed by Gnosis [Saf].

4.4 Threshold Signatures

I mentioned in Section 4.2 that different signature schemes require more or less complex threshold schemes depending on their inner working. Here we give a treatment of some of these threshold schemes. I decided to include to include the details of the protocols since they give an idea of why the protocol I created and will present in Chapters 9 and 10 do work. In fact, in those protocols users have to create an address they control together and the only way to do that privately (i.e. without make external observers know it is an address controlled by multiple parties) is by using a threshold multiparty protocol

4.4.1 Overview

Threshold signatures are part of the general study of threshold cryptosystems. A (t, n)-threshold cryptosystem (also written as t-of-n threshold cryptosystem) is a cryptographic system that distributes a secret between n parties and requires t of them to reveal the secret. Generally, the secret is encrypted with a public key pk and the $t + 1$ parties can decrypt it with a secret key sk recomputed during the decryption phase. Importantly the key is never exposed to the t parties and the same $n + 1$-uple

$$(sk_1, sk_2, \ldots, sk_n, pk)$$

can be reused in other instances. Here, $sk_1, sk_2, \ldots, sk_n$ represent the *shares* of the keys of the n participants. Similar to multisignatures, the threshold signatures are a kind of digital construct that let multiple people sign the same message. More formally:

Definition 4.4.1 *Given a signature scheme, $\mathcal{S} = (\mathbf{KeyGen}, \mathbf{Sig}, \mathbf{Ver})$, a $(t, n)-$threshold signature scheme $\mathcal{TS}$ for $\mathcal{S}$ is a scheme that lets a group of at most n players and at least $t + 1$ players jointly generate a signature. $\mathcal{TS}$ consists of two protocols:*

- **ThreshKeyGen**, *a distributed key generation protocol;*
- **ThreshSig**, *the distributed signing protocol.*

Immediate consequence of Definition 4.4.1 is that a groups of size t or fewer players cannot jointly generate a signature. The specifics of the **ThreshKeyGen** and **ThreshSig** routines depends on the signature scheme $\mathcal{S}$ used and will explained in detail in the respective sections. Note that there is no verification algorithm the reason is that a signature produced by $\mathcal{TS}$ can be verified by $\mathbf{Ver} \in \mathcal{S}$

The work on threshold signatures has been pioneered by Desmedt [Des87] for a kind of group signatures. Numerous other works continued his intuition (see e.g. [Sho00, DK01, LP01]). Yet, after the early 00s, the enthusiasm for this new primitive has nearly died down.

We will see in Section 4.4.3 how the research on threshold signature re-started in a blockchain-centered perspective. But first, we see *how* this primitive is useful in blockchain applications (Section 4.4.2)

[20] The fact that the implementation of a multi-signature is based on a smart contract instead of a specific opcode makes the procedure inherently less secure and prone to smart contracts hacks.

4.4.2 Uses in Blockchains

The big "aesthetic" difference is that by using a threshold signature the number of signatures which are stored is always 1. Indeed we saw that in a k-of-n multisignature the number of signatures stored on the blockchain is at least k. On the other hand, in a t-of-n threshold signatures the number of signatures stored on the blockchain is always 1.

This has a two main consequences for blockchain protocols. The first one is related to the amount of data that is stored on the blockchain. In fact, storing 1 signature instead of k is a nearly k-fold improvement that helps for the scaling of the blockchain. The second one is related to the privacy of the user. In fact, the threshold signatures are inherently a way to hide the number of signatures (i.e. the number of participants) involved in a certain exchange. In Chapter 10 I propose a way to achieve peer-to-peer atomic swaps which are also privacy preserving using threshold signatures.

Threshold signatures have different requirements than those of multi signatures. We saw that multi signatures are relatively easy or difficult to implement based on the transaction model. On the other hand threshold signatures problems are related to the digital signature scheme used. In the following I analyze threshold signatures for three digital signature schemes that are currently used in blockchain projects.

4.4.3 Formalization

ECDSA

Threshold signatures in ECDSA are historically more difficult to design than in other digital signature schemes such as EdDSA and Schnorr [AHS20]. The main reason is the non linearity of the signature in s. Recall that the ECDSA signature is the couple (r, s) where (see Section 4.2.1 for more details):

- r is the point is the first component of the point $K = kG$, with k random number and G generator point of the curve
- $s = k^{-1}(e + dr)$, where $e = \mathcal{H}(m)$

Before the advent of Bitcoin, the most efficient ECDSA (n, t)-threshold signature scheme was the work by Gennaro et al. [GJKR01]. Still, that work can not be considered efficient for real-case uses. Indeed the scheme requires at least $n = 2t + 1$ players in order to implement a security threshold of t players[21]. A practical consequence of that is the impossibility to use the scheme described in [GJKR01] for n-of-n sharing, i.e those sharing where all parties are required to sign. Moreover, the $2t + 1$ players requirement can be costly a weakness. To see that, note that it is easier for an attacker to bribe t players out of $n = 2t + 1$ than t out of $n = t + 1$ (the case where $n = t + 1$ is considered threshold optimal since in that case only one honest party is required against an adversary who compromises up to t parties)[22]. Mackenzie and Reiter [29] proposed a 2-of-2 scheme as a way to solve this issue. Although it is a trivial implementation of the n-of-n case, it cannot be generalized to the case where $n > 2$. The same goes for the works in [DKLS18, Lin17]. For a more detailed analysis of the different proposal of ECDSA threshold signatures see [AHS20].

In this section I describe the work of [GG18]. I chose this particular scheme since it is a simple yet efficient signature scheme. Still it is cumbersome and it needs many sub-protocols and previous MPC results: I think that it is a good way to highlight the higher difficulty of creating threshold ECDSA scheme with respect to a Schnorr-based threshold scheme or BLS-based threshold scheme. Since there are many sub-protocols, I will describe them with a pseudo-code algorithm without going into details. In any case, I give a pointer for more details for each sub protocol.

The protocol is split into two main sub-protocols: *Key generation* and *Signature generation*. As mentioned before, verification is done in the same way as the "normal" ECDSA scheme.

[21] By that I mean that t or less players cannot create a signature.

[22] To see that, note that $\binom{2t+1}{t+1} > t + 2 = \binom{t+2}{t+1}$, meaning that there always more ways to corrupt $t + 1$ players out of $2t + 1$ than to corrupt $t + 1$ out of $t + 2$ players. For example, if $t = 2$, and therefore at least 3 players to sign and $2t + 1 = 5$ players in total, then there are 10 ways to corrupt 3 players out of 5 and 4 ways to corrupt 3 players out of 4. If t increases, the difference is higher. For example, with $t = 3$, $\binom{7}{4} = 35$ and $\binom{5}{4} = 5$.

Algorithm 1 Pailler Cryptosystem

Require: $a, b, n \in \mathbb{Z}$, $c_1, c2 \in \mathcal{E}$
Require: $E_{pk}(\cdot)$ s. t. encription function with pubkey pk
Require: $+_E : c_1 +_E c_2 = E_{pk}(a + b \bmod N)$ sum of cyphertexts
Require: $\times_E : n \times_E c1 = E_{pk}(na \bmod N)$ sum of cyphertexts
  function KeyGen
    $P, Q \leftarrow \mathbb{Z}$
    $N = PQ, \quad \lambda(N) = lcm(P-1, Q-1)$
    $\Gamma \in \mathbb{Z}^*_{N^2}$ s. t. $\exists k, \Gamma^{kN} = 1_{\mathbb{Z}^*_{N^2}}$
    return {Private Key; $\lambda(N)$, Public Key: (N, Γ)}
  end function
  function Encrypt$(m, (N, \Gamma))$
    $x \leftarrow \mathbb{Z}^*_N$
    $c = \Gamma^m x^N \bmod N^2$
    return c
  end function
  function Decrypt$(c, \lambda(N), N)$
    $S = \{u \in \mathbb{Z}_{N^2} : u = 1 \bmod N\}$
    $L : S \to \mathbb{Z}_N, L(u) = (u-1)/N$
    $m = L(c^{\lambda(N)})/L(\Gamma^{\lambda(N)}) \bmod N$
  end function

Key Generation This is a three phases protocol:

- Phase 1. Each party P_i selects $u_i \in_R Z_q$; computes $[KGC_i, KGD_i] = \text{Com}\,(u_i G)$, where KGC_i is the committing-string of party P_i and KGD_i is the committing-string of party P_i[23] and broadcast KGC_i. Each Player P_i broadcasts E_i the public key for Paillier's cryptosystem (see Algorithm 1).

- Phase 2. Each Player P_i broadcasts KGD_i and Y_i be the value decommitted. Then, player P_i performs a (t, n) Feldman Verifiable Secret Scheme (Feldman-VSS, see Algorithm 2 to see how it works) of the value u_i, with Y_i as the "free term in the exponent". The public key is set to $Y = \sum_i Y_i$. Each player adds the private shares received during the n Feldman VSS protocols. The resulting values x_i are a (t, n) Shamir's secret sharing of the secret key $x = \sum_i u_i$. Note that the values $X_i = x_i G$ are public.

- Phase 3. Let $N_i = p_i q_i$ be the RSA modulus associated with E_i. Each player P_i ZK-proves that he knows x_i using Schnorr's protocol [Sch91]. Each P_i also proves that he knows p_i, q_i using any proof of knowledge of integer factorization (see e.g. [PS00b])

Signature Generation After the Key Generation procedure, parties P_i can generate signatures. Assume M a message and m its hash. Since this threshold cryptosystem is a t-of-n protocol, I can assume that the set S of signing party is a subset of of t parties among $\{P_1, \ldots, P_n\}$. Since the secret is shared with a (extension of) Shamir Secret Sharing, each party $P-i$ can "locally remap" a its share of secret shared between n parties to a share of a secret between t parties. To do that, let $\lambda_{i,S}$ be the i-th Lagrangian coefficient of party P_i in S. Then, the share x_i is remapped to $w_i = \lambda_{i,S} x_i$. All parties can compute $W_i := w_i G$ by doing $W_i = \lambda_{i,S} X_i$ for each P_i in S[24]. The signature generation is a five-phases procedure:

- Phase 1. Each party P_i chooses uniformly randomly $k_i, \gamma_i \in \mathbb{Z}_q$; computes $[C_i, D_i] = \text{Com}\,(\gamma_i G)$

[23] Practically we can think of Com as a hash-based commitment for a hash function H. In this case $KGC_i = H(u_i G, r)$, where r represent random coin-tosses, and $KGD_i = (u_i G, r)$. For more details see Construction 5.1.1.

[24] with the symbol ":=" I mean "definition". In this sense, W_i is defined as g^{w_i} and can be checked by doing a different computation that involves only public parameters, namely by computing $X_i^{\lambda_{i,S}}$

Algorithm 2 Feldman VSS (Shamir's SS extension)

Require: $\mathcal{G}$ group , $g \in \mathcal{G}$
Require: P_0 the holder of secret σ and dealer of shares
Require: $\{P_1, P_2, \ldots, P_t\}$ are t parties P_0 send shares to
  function CREATESHARES(σ, q) ▷ Performed by P_0
    $\forall i = 1 \ldots t, a_i \leftarrow \mathbb{Z}_q$
    Initialize $p(x) = \sigma + a_1 x + a_2 x^2 + \cdots + a_t x^t \bmod q$
    Publish $v_i = g^{a_i}$ in $\mathcal{G}$ for all $i \in [1, t]$ and $v_0 = g^\sigma$ in $\mathcal{G}$
    send $\sigma_i = p(i) \bmod q$ to $P_i, \forall i \in [1, t]$
  end function
  function CHECKSHARES($\sigma_i, v_i \forall i \in [0, t]$) ▷ Performed by $P_i, \forall i \in [1, t]$
    Compute $H = \sigma_i G$
    Compute $\pi = \sum_{j=0}^{t} v_j^{i^j}$
    if $H = \pi$ **then**
      return $True$
    else
      return $False$
    end if
  end function

and broadcast C_i. Let $k = \sum_{i \in S} k_i$ and $\gamma = \sum_{i \in S} \gamma_i$. By construction:

$$k\gamma = \sum_{i,j \in S} k_i \gamma_j \bmod q$$
$$kx = \sum_{i,j \in S} k_i w_j \bmod q$$

- Phase 2. Every pair of players P_i, P_j engages in two multiplicative-to-additive share conversion sub-protocols
  - P_i, P_j run MtA (see Algorithm 3) with shares k_i, γ_j respectively. Let α_{ij} [resp. β_{ij}] be the share received by player P_i [resp. P_j] at the end of this protocol, i.e.

    $$k_i \gamma_j = \alpha_{ij} + \beta_{ij}$$

    Player P_i sets $\delta_i = k_i \gamma_i + \sum_{j \neq i} \alpha_{ij} + \sum_{j \neq i} \beta_{ji}$. Note that the δ_i are a (t, t) additive sharing of $k\gamma = \sum_{i \in S} \delta_i$
  - P_i, P_j run MtAwc (see Algorithm 4) with shares k_i, w_j respectively. Let μ_{ij} [resp. ν_{ij}] be the share received by player P_i [resp. P_j] at the end of this protocol, i.e.

    $$k_i w_j = \mu_{ij} + \nu_{ij}$$

    Player P_i sets $\sigma_i = k_i w_i + \sum_{j \neq i} \mu_{ij} + \sum_{j \neq i} \nu_{ji}$. Note that the σ_i are a (t, t) additive sharing of $kx = \sum_{i \in S} \sigma_i$
- Phase 3. Every player P_i broadcasts δ_i and the players reconstruct $\delta = \sum_{i \in S} \delta_i = k\gamma$. The players compute $\delta^{-1} \bmod q$.
- Phase 4. Each Player P_i broadcasts D_i. Let Γ_i be the values decommitted by P_i who proves in ZK that he knows γ_i s.t. $\Gamma_i = \gamma_i G$ using Schnorr's protocol [34]. The players compute

$$R = \delta^{-1} \left[\sum_{i \in S} \Gamma_i \right] = \left[\left(\sum_{i \in S} \gamma_i \right) k^{-1} \gamma^{-1} \right] G = \gamma \gamma^{-1} k^{-1} G = k^{-1} G$$

and $r = H'(R)$

Algorithm 3 Multiplicative To Additive (MtA)

Require: $a, b, K \in \mathbb{Z}_q$, $c_1, c2 \in \mathcal{E}$
Require: $N > K^2 q$
Require: $E_I(\cdot)$ s. t. encription function with pubkey of party $I \in \{A, B\}$
Require: $x = ab$ where A has a and B has b
Require: $+_E : c_1 +_E c_2 = E_{pk}(a + b \bmod N)$ sum of cyphertexts
Require: $\times_E : n \times_E c1 = E_{pk}(na \bmod N)$ sum of cyphertexts
procedure MtA
A computes $c_A = E_A(a)$
A creates zero-knowledge range-proof π_a that $a < K$
A sends c_a, π_a to B
B select randomly $\beta' \leftarrow \mathbb{Z}$ and sets $\beta = -\beta' \bmod q$
B computes $c_B = b \times_E E_A(b) +_E E_A(\beta')$[25]
B creates zero-knowledge range-proof π_b that $b < K$
B sends c_b, π_b to A
A verifies π_b, decrypts c_b and obtains α'
A sets $\alpha = \alpha' \bmod q$
end procedure ▷ This way $x = ab = \alpha + \beta$

- Phase 5. Each player P_i sets $s_i = mk_i + r\sigma_i$. Note that

$$\sum_{i \in S} s_i = m \sum_{i \in S} k_i + r \sum_{i \in S} \sigma_i = mk + rkx = k(m + xr) = s$$

i.e. the s_i are a (t, t) sharing of s.

  - (5A) Player P_i chooses $\ell_i, \rho_i \in_R Z_q$ computes $V_i = s_i R + \ell_i G, A_i = \rho_i G$, and $\left[\hat{C}_i, \hat{D}_i\right] = \text{Com}\left(V_i, A_i\right)$ and broadcasts $\hat{C}_i$. Let $\ell = \sum_i \ell_i$ and $\rho = \sum_i \rho_i$.
  - (5B) Player P_i broadcasts $\hat{D}_i$ and proves in ZK that he knows s_i, ℓ_i, ρ_i such that $V_i = s_i R + \ell_i G$ and $A_i = \rho_i G$. If a ZK proof fails, the protocol aborts. Let $V = -mG - rY + \sum_{i \in S} V_i$ (this should be $V = \ell G$) and $A = \sum_{i \in S} A_i$.
  - (5C) Player P_i computes $U_i = \rho_i V$ and $T_i = \ell_i A$. It commits $\left[\tilde{C}_i, \tilde{D}_i\right] = \text{Com}\left(U_i, T_i\right)$ and broadcasts $\tilde{C}_i$.
  - (5D) Player P_i broadcasts $\tilde{D}_i$ to decommit to U_i, T_i If $\sum_{i \in S} [T_i] \neq \sum_{i \in S} U_i$ the protocol aborts.
  - (5E) Otherwise player P_i broadcasts s_i. The players compute $s = \sum_{i \in S} s_i$. If (r, s) is not a valid signature the players abort, otherwise they accept and end the protocol.

Schnorr

Compared to ECDSA digital signature scheme, Schnorr-based threshold signatures are much easier to design and implement. In fact the signatures are linear in the s variable and therefore they can be easily added to obtain a single signature from different shares of signatures of a single message. This is evidenced by the fact that there are several proposals that work on a practical level (see e.g. [NKDM03, BN06b, BCJ08, MWLD10]).

It is possible to create aggregate Schnorr signatures in a "naive" fashion simply by summing the signatures of different participants. More formally, using the notation previously used for Schnorr signatures in Section 4.2.2, for the case of n participants $\{P_1, \ldots, P_n\}$ needing to sign a message m, I assume there exists the set of public keys $L = Q_1, \ldots, Q_n$ where $Q_i = d_i G$. In this setting, the "naive" threshold signature works as follows:

1. Each P_i randomly generates and communicates to $\{P_j\}_{j \neq i}$ a nonce $R_i = r_i G$
2. Each P_i computes $R = \sum_{j=1}^{n} R_j$ and $c = H(\tilde{Q}, R, m)$ where $\tilde{Q} = \sum_{j=1}^{n} Q_j$ is the sum of the public keys of all participants, m is the message to be signed and H is a hash function

Algorithm 4 Multiplicative To Additive With Check (MtAwc)

Require: $a, b, K \in \mathbb{Z}$, $c_1, c2 \in \mathcal{E}$
Require: $N > K^2 q$
Require: $E_I(\cdot)$ s. t. encription function with pubkey of party $I \in \{A, B\}$
Require: $x = ab$ where A has a and B has b
Require: $G = g^b$ is public
Require: $+_E : c_1 +_E c_2 = E_{pk}(a + b \bmod N)$ sum of cyphertexts
Require: $\times_E : n \times_E c1 = E_{pk}(na \bmod N)$ sum of cyphertexts
procedure MtAwc
A computes $c_A = E_A(a)$
A creates zero-knowledge range-proof π_a that $a < K$
A sends c_a, π_a to B
B select randomly $\beta' \leftarrow \mathbb{Z}$ and sets $\beta = -\beta' \bmod q$
B computes $c_B = b \times_E E_A(b) +_E E_A(\beta')$[26]
B creates zero-knowledge range-proof π_b that $b < K$
B creates zero-knowledge proof π'_b that B knows (b, β') s. t. $G = g^b, c_B = b \times_E E_A(b) +_E E_A(\beta')$
B sends c_b, π_b, π'_b to A
A verifies π_b, decrypts c_b and obtains α'
A sets $\alpha = \alpha' \bmod q$
end procedure ▷ This way $x = ab = \alpha + \beta$

3. Each P_i computes a partial signature $s_i = r_i + cx_i$ and communicates it to $\{P_j\}_{j \neq i}$
4. After receiving $\{s_j\}_{j \neq i}$, each P_i computes $s = \sum_{j=1}^{n} s_j$
5. The signature is (R, s)

Having $\tilde{Q}$ the verification is straightforward since it is equal to the verification of "normal" Schnorr signatures.

Unfortunately this simple method is prone to attacks. As noted multiple times (see e.g. [Lan96, MH96, MOR01]), this simplistic protocol is vulnerable to a rogue attack. In particular a malicious participant (e.g. P_1) can craft its public key as:

$$Q_1 = d_1 G - (\sum_{j=2}^{n} Q_j)$$

and forge signatures for $\{P_1, \ldots, P_n\}$ since the public key of the group of participants in this case would be is:

$$\begin{aligned} \tilde{Q} = \sum_{j=1}^{n} Q_j &= Q_1 + (\sum_{j=2}^{n} Q_j) \\ &= d_1 G - (\sum_{j=2}^{n} Q_j) + (\sum_{j=2}^{n} Q_j) \\ &= d_1 G \end{aligned} \tag{4.10}$$

meaning the aggregate signature can be done (i.e. forged) using only the private key of P_1.

In the research field related to Bitcoin, where Schnorr signatures were first proposed as an alternative to ECDSA, several variants have been developed to obtain Schnorr-based threshold signatures. These include [MPSW19, NRSW20, NRS].

In particular MuSig [MPSW19] uses an initial pre-processing round between participants where they commit public keys to avoid rogue attack. The final protocol therefore has still a total of two rounds, since the committing one is done as pre-processing and not as part of the signature scheme. Yet, together with other works that attempted to reduce the number of rounds of interaction from three (as in the seminal work of Bellare and Neven [BN06b]) to two as in e.g. [BCJ08, STV+16], MuSig

has been proved insecure nonetheless. More formally, the work of Drijvers et al [DEF+19] proved that previous works which used less than three rounds are not secure (i.e. there is no algebraic reduction) under the discrete-logarithm or one-more discrete-logarithm problem[27].

To solve this problem, two proposals have been presented. The first one, proposed by the authors themselves in [DEF+19], despite having in total two rounds of interaction, outputs a non-ordinary Schnorr signature. Thus, it is not possible to use this method since it is not a standard signature. The second, proposed by Nick et al [NRSW20] and called MuSig-DN, is a modification of MuSig and has as output a standard signature. but this is obtained using zero-knowledge proofs to prove a deterministic derivation of the nonce to all cosigners. The MuSig-DN method has only two rounds of interaction, but is less efficient than three-round MuSig.

As of now, the last proposed protocol for threshold signatures based on Schnorr signatures for Bitcoin is called MuSig2 [NRS]. In the following I present the key-generation and signature-generation of this protocol. As in the case with threshold-ECDSA, the verification is equal to the verification of the "normal" version of the signature scheme.

Key Generation Assuming each participant P_i already has its own private key d_i, then the key-generation is the creation of a public key $\tilde{Q}$ for all participant. Assuming $L = Q_1, \ldots, Q_n$ is the set of public keys, the public key of the group of participants is $\tilde{Q} = \sum_{j=1}^{n} a_j Q_j$ where $a_i = H_{agg}(L, Q_i)$[28]. This algorithm is called KeyAgg.

Signature Generation In this protocol, note that to obtain a scheme secure under concurrent sessions, *each* signer P_i sends a list of $\nu \geq 2$ nonces $R_{i,1}, \ldots, R_{i,\nu}$ instead of a single nonce R_i. The participant P_i then uses a linear combination $\widehat{R}_i = \sum_{j=1}^{\nu} b^{j-1} R_{i,j}$ of the ν nonces. The use of multiple nonces instead of one is peculiar to the MuSig2 scheme, and ν is a parameter of it. We see the details below.

Recall that there are two rounds. In the first round each participant does the operations Sign and then the participants exchange the data with SignAgg.

1. Sign: For each $j \in \{1, \ldots, \nu\}$, each signer P_i generates random $r_{1,j} \leftarrow \mathbb{Z}_p$ and computes $R_{1,j} = r_{1,j}G$. P_i then outputs the ν nonces $(R_{1,1}, \ldots, R_{1,\nu})$.

2. SignAgg: The aggregator receives outputs $(R_{1,1}, \ldots, R_{1,\nu}), \ldots, (R_{n,1}, \ldots, R_{n,\nu})$ from all signers. The it aggregates them by computing $R_j := \sum_{i=1}^{n} R_{i,j}$ for each $j \in \{1, \ldots, \nu\}$ and finally it outputs $(R_1, \ldots, R_\nu)$

In the second round each participant does the operations Sign′, SignAgg′ and Sign″.

1. Sign′: P_i uses KeyAgg to compute $\widetilde{Q}$ and stores its own key aggregation coefficient a_i. Upon reception of the first-round output $(R_1, \ldots, R_\nu)$, the signer computes $b = H_{non}\left(\widetilde{Q}, (R_1, \ldots, R_\nu), m\right)$. Then it computes

$$R := \sum_{j=1}^{\nu} b^{j-1} R_j$$
$$c := \mathrm{H}_{sig}(\widetilde{Q}, R, m)$$
$$s_i := ca_i x_i + \sum_{j=1}^{\nu} b^{j-1} r_{i,j} \bmod p$$

   and outputs s_i.

2. SignAgg′: P_i receives outputs $(s_1, \ldots, s_n)$ from $P_{j\,j\neq i}$ and aggregates them by outputting the sum $s := \sum_{i=1}^{n} s_i \bmod p$.

[27] Even if the mentioned works where presented with a security proof under at least one of the mentioned assumptions, in [DEF+19] the authors show flaws in those proofs. This flaws are related to the simulation of signing queries in combination with a rewinding argument. For more details see Theorems 1 and 2 of [PS00a]. To see a high-level explanation of the flaws in the MuSig case, see Appendix C of [NRS]

[28] Recall the definition of *tagged hash*, where from a hash function H is possible to create a family of hash function by putting $H_i(\cdot) = H(tag_i||\cdot)$: in this case, the tag is agg. See Section 4.2.2

3. SIGN'' : P_i returns the signature $\sigma := (R, s)$.

As you can see, even though there are two rounds of communication, the procedure for obtaining a Schnorr-based threshold signature is considerably simpler than that for ECDSA.

BLS

While there is a way to aggregate the signatures in the RFC [BGW+20], the method is used to aggregate different messages and it only perform n-of-n aggregation. In the following I show a way to perform a threshold BLS-signature scheme. I use a Distributed Key Generation (DKG) protocol by Gennaro et al. [GJKR06] to generate the shares of the secret keys and the public key. The signature generation protocol is a straightforward application of Shamir's secret sharing algorithm using Lagrange interpolation. In the following I show the key generation and signing algorithms.

Key Generation The method presented by Gennaro et al. [GJKR06] is an improvement over the Pedersen's DKG Protocol. The authors show that latter protocol cannot guarantee the correctness of the output probability-distribution in the presence of an adversary. To show that they present a strategy for an adversary to manipulate the distribution of the resulting secret (in this case, the aggregate private key d) to something different from the uniform distribution. The danger comes from the fact that a key not extracted from a uniform distribution is potentially more likely to be "guessed" by an adversary.

As usual, assume n parties who need to create an aggregated public key such that t of them can sign a message m. The key generation protocol is presented below: Steps 1-4 deal with the creation of the shares for the secret key (which is never actually computed); Steps 5-6 deal with the creation of the public key (which is computed and shared).

1. Each party P_i CREATESHARES with a Pedersen-VSS (see Algorithm 5) of a random value z_i as a dealer

2. Each party P_j uses CHECKSHARES from Algorithm 5 to verify the shares he received from the other parties. If CHECKSHARES returns $False$ for an index i, then P_j broadcasts a *complaint* against P_i

   (a) Each party P_i who, as a dealer, received a *complaint* from party P_j broadcasts its values s_{ij}, s'_{ij}

   (b) Each party marks as disqualified any party that either

   - received more than t complaints in Step 2, or
   - answered a complaint in Step 2a with values still result in a $False$ result of CHECKSHARES

3. Each party then builds the set of non-disqualified parties $QUAL$

4. The distributed secret value x (not explicitly computed by any party) is:

$$x = \sum_{i \in QUAL} z_i \bmod q$$

   Each party P_i sets his share of the secret as $x_i = \sum_{j \in QUAL} s_{ji} \bmod q$ and the value $x'_i = \sum_{j \in QUAL} s'_{ji} \bmod q$

5. Each party $P_i, i \in QUAL$ exposes $y_i = z_i g \bmod p$ via Feldman-VSS (see Algorithm 2)

6. The shared public key is $y = \sum_{i \in QUAL} y_i \bmod p$

Algorithm 5 Pedersen VSS (Shamir's SS extension)

Require: $\mathcal{G}$ group , $g, h \in \mathcal{G}$ ▷ No relation between g and h must be known
Require: P_0 the holder of secret $\sigma = a_0$ and dealer of shares
Require: $\{P_1, P_2, \ldots, P_t\}$ are t parties P_0 send shares to
  function CREATESHARES(σ, q) ▷ Performed by P_0
    $\forall i = 1 \ldots t, a_i \leftarrow \mathbb{Z}_q$
    Initialize $p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_t x^t \bmod q$
    Initialize $p'(x) = b_0 + b_1 x + b_2 x^2 + \cdots + b_t x^t \bmod q$
    Publish $C_i = a_i g + b_i h$ in $\mathcal{G}$ for all $i \in [1, t]$
    send $s_i = p(i) \bmod q$ and $s'_i = p'(i) \bmod q$ to $P_i, \forall i \in [1, t]$
  end function
  function CHECKSHARES($\sigma_i, [v_i, \forall i \in [1, t]]$) ▷ Performed by $P_i, \forall i \in [1, t]$
    Compute $S = s_i g + s'_i h$
    Compute $\pi = \sum_{j=1}^{t} C_j i^j$
    if $S = \pi$ **then**
      return $True$
    else
      return $False$
    end if
  end function

Signature Generation Given a message m, every party $P_i \in QUAL$ starts by creating a BLS signature using its own secret share z_i:

$$sig_i = H(m) * z_i$$

After all $P_j \in QUAL$ have created their signatures, the aggregated signature is computed as follows:

$$sig = \sum_{i \in QUAL} sig_i$$

That works since:

$$\begin{aligned} sig &= \sum_{i \in QUAL} sig_i = \sum_{i \in QUAL} H(m) * z_i \\ &= H(m) \sum_{i \in QUAL} z_i = H(m)x \end{aligned}$$

As always, the verification algorithm is the same as the "normal" BLS signatures.

Chapter 5

Commit-Reveal with Hashes

One of the most used atomic swap method is currently the Hash Time-Lock puzzle which, as the name suggests, is based on the commit-reveal protocol implemented with hashes. Goal of this section is to present the concept of commit-reveal protocols with hashes, present the state of the art with respect to HTLCs and present our improvement to the state of the art.

Note that the goal of the scheme described here is cross-blockchain interoperability instead of amount-hiding, i.e. interoperability instead of privacy, and I will see how those two concepts are different.

The Chapter is so divided. In Section 5.1 I define commit/reveal schemes properly and we will see that it is possible to create one in a cross-chain protocol using hash-lock contracts. In Section 5.2 we present one of the most popular mechanisms used for peer-to-peer interoperability, namely Hash Time-Lock Contracts. Finally in Section 5.3 I analyze HTLCs' problems. Some of the problems presented in Section 5.3 can be solved by increasing the maximum participant from 2 to *many*. This has been done in two steps. The first step is to prove the feasibility of funds-exchange between multiple parties. I describe how to obtain it in Chapter 9. The chapter describes a protocol I created that uses threshold signatures to enable control of an address by multiple parties with no trace on the blockchain: the address is indistinguishable from a "normal" address. I called the protocol DMix [BS20a], since in practice it acts as a decentralized mixer.

The second step extends DMix into a full-fledged interoperatibility scheme providing a HTLC between multiple parties at the same time. The protocol I created to do that is called MP-HTLC [BS22b] and it is presented in Chapter 10.

To understand both protocols a thorough explanation of how hash functions works and how it is possible to leverage their construction to obtain interoperability schemes is needed, and that is the topic of this chapter.

5.1 Commitment Scheme

Loosely speaking a commit-reveal scheme is a two phases protocol between at least two parties. In the first phase one participant, called the *sender*, commits itself to a value preserving secrecy and binding. In the reveal phase the sender reveals its values to the receiver(s): the receiver(s) knows that the value is the "right" one thanks to the binding property.

There are many ways to create commitment schemes. In the following I prove that it is possible to create commit-reveal schemes with (collision-free) hashing functions. This proof will let us talk and use hash-locks in a blockchain environment and in particular about hash time-lock values.

I proceed in three steps: in the first step I present collision-free hashing functions, then I prove that it is possible to create a one-bit commitment using those functions, and finally I extend the construction to arbitrary number of bits. For the definition I use [Gol01] as reference.

Let's introduce general hashing functions, i.e. the superset of hashing functions that carry no hardness requirement, such as requirements on collision. I define S_n^m the set of strings mapping n-bit into m-bit strings. I abuse notation and I identify S_n^m with its results. Finally let H_n^m be a random variable uniformly distributed over the set S_n^m. The conditions for S_n^m to be a family of hashing functions are:

Definition 5.1.1 *A hashing function is a function such that*

1. *S_n^m is a pairwise-independent family of mappings: For every $x \neq y \in \{0,1\}^n$, the random variables $H_n^m(x)$ and $H_n^m(y)$ are independent and uniformly distributed in $\{0,1\}^m$.*
2. *S_n^m has succinct representation: $S_n^m = \{0,1\}^{poly\,(n,m)}$.*
3. *S_n^m can be efficiently evaluated: There exists a polynomial-time algorithm that on input a representation of a function h (in S_n^m) and a string $x \in \{0,1\}^n$ returns $h(x)$.*

Note that 1 does not imply that if $x \neq y$ then $H(x) \neq H(y)$.

One construction of a general hashing family is the affine-transformation of a n-dimensional binary vector a into a m-dimensional binary vector b obtained via a matrix multiplication with A:

$$\underbrace{\begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}}_{\substack{m\text{-dimensional} \\ \text{binary vector}}} = \underbrace{\begin{pmatrix} A_{11} & A_{12} & \dots & A_{1n} \\ A_{21} & A_{22} & \dots & A_{2n} \\ \vdots & \vdots & & \vdots \\ A_{m1} & A_{m2} & \dots & A_{mn} \end{pmatrix}}_{\text{affine matrix}} \underbrace{\begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix}}_{\substack{n\text{-dimensional} \\ \text{binary vector}}}$$

From that we see that no special assumption is required. On the other hand, if I want that $\forall x, y\,, x \neq y \Rightarrow H(x) \neq H(y)$ minus a negligible function, then I have to assume the existence of one-way functions, i.e. functions efficiently computable but whom inverse function is impossible to compute. To see why this assumption is needed, note that the existence of fixed-length collision-free hash functions imply fixed-length (one-time) signature schemes and that (one-time) signature schemes imply the existence of one-way functions. To see the last implication assume that a $(\mathbf{Gen}, \mathbf{Sign}, \mathbf{Ver})$ signature scheme such that the key-generation algorithm $\mathbf{Gen}(1^\lambda, r) = (\cdot, pk)$ is invertible with (non negligible) probability $\epsilon(\lambda)$ and I call G^{-1} its inverse. Then an adversary $\mathcal{A}$ can easily obtain the signing key by computing $\mathbf{Gen}(G^{-1}(pk))$ and therefore produce innumerable forgeries.

By assuming the existence of one-way functions, I can define collision-free hash functions (see [Gol04], Def 6.2.5).

Definition 5.1.2 *A collision function is a function such that*

1. *(admissible indexing - technical): For some polynomial p, all sufficiently large n, and every s in the range of $I\,(1^n)$, it holds that $n \leq p(|s|)$. Furthermore, n can be computed in polynomial-time from s.*
2. *(efficient evaluation): There exists a polynomial-time algorithm that, given s and x, returns $h_s(x)$.*
3. *(hard-to-form collisions): I say that the pair (x, x') forms a collision under the function h if $h(x) = h\,(x')$ but $x \neq x'$. I require that every probabilistic polynomial time algorithm, given $I\,(1^n)$ as input, outputs a collision under $h_{I(1^n)}$ with negligible probability. That is, for every probabilistic polynomial-time algorithm A, every positive polynomial p, and all sufficiently large n,*

$$\Pr\left[A\left(I\left(1^n\right)\right) \text{ is a collision under } h_{I(1^n)}\right] < \frac{1}{p(n)}$$

where the probability is taken over the internal coin tosses of algorithms I and A.

This concludes the first step.

Now let h_s be a collision-free hash function, such that $|h_s(x)| = l|s|\,\forall x$.

Construction 5.1.1 *Let the sender S and the receiver R be two participants. A commit reveal protocol between S and R is a two-phase protocol such that*

1. *Commit phase*
   - *to receive a commitment to a bit (s security parameter), s uniformly selects $r \in \{0,1\}^*$*
   - *S commits to a value $v \in \{0,1\}$ by sending $h_s(v \,||\, r)$ where $||$ is bit concatenation.*

2. *Reveal phase*
   - *S reveals r*
   - *R accepts 0 if* $h_s(0 \mid\mid r) = \alpha$ *and 1 if* $h_s(1 \mid\mid r) = \alpha$ *where* α *is R's view of the commit phase.*

Note that in principle R could potentially accept both values. But that would mean that $0 \mid\mid r$ and $1 \mid\mid r$ produce a collision, which happens with probability $< 1/2^{l|s|}$ by Definition 5.1.2. Incidentally that also proves why I need collision-free hash functions instead of general hash functions as defined in Definition 5.1.1. I prove that in Proposition 5.1.1.

Proposition 5.1.1 *If* h_s *is a collision free hash function, then the protocol in Construction 5.1.1 constitutes a one-bit commitment scheme.*

Proof 5.1.1 *(**Secrecy**) As said in the beginning of the section, I have to prove secrecy and binding. The secrecy requirement follows from the fact that* h_s *is a general hash function. Therefore given* $H_*^{l|s|}$ *the random variable over the set of strings* $S_*^{l|s|}$ *then*

$$\mathbb{P}\{H_*^{l|s|} = h\} < \frac{1}{2^{l|s|}}.$$

by Definition 5.1.2.

Specifically let $U_{l|s|}$ *denote the random variable uniformly distributed over the set of strings of length* $l(|s|)$*. Then* $U_{l(|s|)}$ *and* $H_*^{l(|s|)}$ *are identically distributed. Therefore it's impossible to find* $\tilde{r}$ *such that* $U_{l(|s|)}$ *and* $H_*^{l(|s|)}$ *are computationally distinguishable and therefore I have secrecy* $\forall \tilde{r} \in \{0,1\}^*$*. (**Unambiguity**) Unambiguity follows from the fact that* h_s *is collision-free, and specifically from Point 3 of Definition 5.1.2.*

This proof concludes the second step.

The last step is the expansion from a single bit v to an arbitrary-length bit vector $\tilde{v}$. This step is actually easy, once I note that the output of h_s is fixed-length, regardless of the length of the input. More formally, let $r, \tilde{r} \in \{0,1\}^*$, $v \in \{0,1\}$ and $\tilde{v} \in \{0,1\}^*$. Furthermore let $H = h_s(v \mid\mid r)$ and $\tilde{H} = h_s(\tilde{v} \mid\mid \tilde{r})$. I want to prove that the random variables $H, \tilde{H}$ are computationally indistinguishable. Assume that $\exists (v, r), (\tilde{v}, \tilde{r})$ such that $H, \tilde{H}$ are computationally distinguishable, i.e.

$$\mathbb{P}\{H = h\} - \mathbb{P}\{\tilde{H} = \tilde{h}\} > \frac{1}{2^s}.$$

Then from 3 of Definition 5.1.2,

$$\mathbb{P}\{h_s(v \mid\mid r) = h\} < \frac{1}{2^s}.$$

Consequently

$$\frac{1}{2^s} < \mathbb{P}\{H = h\} - \mathbb{P}\{\tilde{H} = \tilde{h}\} < \frac{1}{2^s} - \mathbb{P}\{\tilde{H} = \tilde{h}\}. \tag{5.1}$$

And since $\mathbb{P}\{\tilde{H} = \tilde{h}\} > 0$ Equation 5.1 is a contradiction.

Therefore since $H, \tilde{H}$ are computationally indistinguishable, I can use Proposition 5.1.1 and conclude that Construction 5.1.1 is a commit reveal scheme even if the value committed is $\tilde{v}$.

I conclude this section by explaining how h_s can be used in a blockchain-based commit-reveal scheme. First of all, let $x(v, r) = v \mid\mid r = x$, so that I can see any input of h_s as a concatenation of arbitrary length bit vectors.

Construction 5.1.2 *A Hash-lock is a blockchain commitment scheme for which*

- *commitment works as Construction 5.1.1, the sender S sends* $h_s(x)$ *to R on a channel and embeds* $h_s(x)$ *in a transaction tx such that whoever wants to redeem tx has to provide x (among other requirements).*

- *Revealing. There are two possible ways of revealing. Either S sends x to R, or S puts x in a transaction tx' (potentially $tx = tx'$) so that every blockchain user can see the revealing.*

Note that I don't specify any blockchain. With this construction, a Hash-lock can be used either in a blockchain, in two or more blockchains (as in Atomic Swaps, see Section 5.2), or off-chain as in payment channels like Lightning Network (before Taproot).

5.2 HTLC

The first description of how an HTLC works is by Tier Nolan [Nol13]. In this description Nolan describes a series of steps after which two participants A and B exchange two coins in two different blockchains. The Figure 5.1 shows a diagram of the operation of an HTLC

5.2.1 Inner working

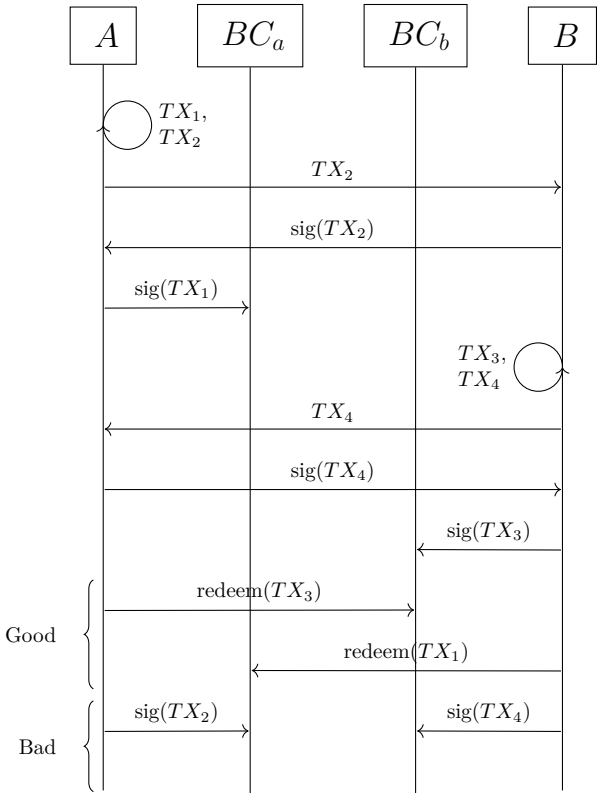


Figure 5.1: Scheme of the inner working of a HTLC between two blockchains.

With reference to Figure 5.1, let Alice A and Bob B be the clients of two participants who want to exchange two coins in two different blockchains. In particular, suppose that A wants to trade $1coin_A$ in blockchain BC_A and conversely B wants to trade $1coin_B$ in blockchain BC_B. As defined in Definition 7.5.1, the exchange is defined as "successful" if one of the two conditions occurs:

1. Either A obtains $1coin_B$ in BC_B AND B obtains $1coin_A$ in BC_A
2. Or A still has $1coin_A$ in BC_A AND B still has $1coin_B$ in BC_B

In the second case, I assume either A or B has been malicious and the protocol aborted. In those cases still, no party lose any fund. While not explicitly assumed in [Nol13], the assumption is that A and B can have a channel for communication that stays open throughout the protocol. Assume also A and B agreed on the chains, the coin to exchange and the rate α such that $1coin_A = \alpha 1coin_B$. In the following I assume $\alpha = 1$.

In the first phase, A picks a random number x and creates a transaction I call TX_1 whose verification code (see Section 3.1) can be expressed in:

> TX_1:={Pay 1$coin_A$ to `<B's public key>` if (x for $H(x)$ known and signed by B) or (signed by A & B)}

Note the first branch of the the if statement. The redeemer has to put the preimage of $H(x)$ beyond the normal signature requirement: that is the hash-based commitment in the HTLC. Before sending TX_1 to the network, A needs some reassurance that it will not loose the funds. To do that, A creates a new transaction TX_2 such that:

> TX_2:={Pay 1$coin_A$ from TX_1 to `<A's public key>`, locked 48 hours in the future signed by A}

and then sends it to B. If B signs TX_2 and returns it to A, then A broadcasts TX_1 to the network of Blockchain BC_A. This concludes the first phase

Before moving on to the next phase, I remark two things. The first one is that Alice is the only participant who transferred the coins, yet it does not risks nothing (least of losing time) thanks to transaction TX_2. At the same time, TX_2 represents Alice insurance for the future and, since it's locked for 48 hours[1]: even if Alice broadcasts TX_2 at this time, the transaction would not be taken from the mempool because of the time-lock. Consequently Bob knows that coins in TX_1 can not be redeemed by Alice for 48 hours.

After Bob's see that TX_1 has been broadcasted, it creates transaction TX_3 such that:

> TX_3:={Pay 1$coin_B$ to `<A's public key>` if (x for $H(x)$ known and signed by A) or (signed by A & B)}

Bob then proceeds to create a transaction TX_4 similar to transaction TX_2 for the same reasons, sends it to Alice and wait for Alice to sign it and return it. For clarity, TX_4 is of the form:

> TX_4:={Pay 1$coin_B$ from TX_3 to `<B's public key>`, locked 24 hours in the future signed by B}

If Bob receives a correctly signed TX_4, then it broadcasts TX_3 to the network of blockchain BC_B. This concludes the second phase.

At this point in time, both Alice and Bob sent (i.e. broadcasted to their respective network) the coins. Now they both wait. As per transaction TX_4, Alice has 24 hours to redeem transaction TX_3: if Alice waits more than that, Bob assumes Alice is malicious/faulty and broadcasts transaction TX_4 to redeem its coins. On the other hand, if Alice redeems transaction TX_3 (phase three), it leaves a trail of the value of x on the public blockchain. In that case, Bob can pick x and use it to redeem transaction TX_1, before another 24 hours pass ($24 + 24 = 48$), concluding phase four or the protocol.

5.2.2 Implementation

From an implementation point of view, I must distinguish between the implementation for UTXO-based blockchains (see Section 3.1) and the implementation for Account-based blockchains (see Section 3.2). In fact, while in the first case it is possible to use "normal" transactions, in the second case it is necessary to create a smart contract for it.

UTXO case

In blockchains of this type it is easy to create an HTLC since on a technical level each transaction is a verification smart contract. In fact, even pay-to-pubkey transactions (the oldest and simplest) are a smart contract whose output script is:

```
<pubkey> OP_CHECKSIG
```

In the case of a transaction capable of handling an HTLC instead, I can implement the script output of the TX_1 transaction with:

[1] This artificial limit can be changed at creation time. I left the number used in the original proposal.

```
1
2     function setHash(bytes32 _hashValue) public inState(State.Created) onlyOwner {
3     hashValue=_hashValue;
4     state = State.HashInserted;
5     }
6
7     function isSha256Preimage(bytes memory _candidate) public returns (bool) {
8     //bytes memory candidate_bytes = abi.encode(_candidate);
9     return sha256(_candidate)==hashValue;
10    }
11
12    function PayOut(bytes memory _candidate) public inState(State.Active) returns (
          bool) {
13        if (isSha256Preimage(_candidate)){
14            for(uint i = 0 ; i<expectedPartiesNumber; i++) {
15            BitcoinersIndexed[i].transfer(EtherersAmounts[EtherersIndexed[i]]);
16        }
17        return isSha256Preimage(_candidate);
18    }
```

Listing 5.1: Snipped of code for HTLC in Solidity. for the full contract see https://github.com/disnocen/mp-htlc/blob/master/Interop.sol

```
IF
  2 <key A> <key B> 2 OP_CHECKMULTISIGVERIFY
ELSE
  <key B> OP_CHECKSIGVERIFY OP_SHA256 <H(x)> OP_EQUALVERIFY
ENDIF
```

Account case

While the goal in the account-based blockchains is the same as in the UTXO-based case, the way to achieve it is different. Note that account-based models generally support smart contract, and this is certainly the Ethereum case. Furthermore, since I will use the HTLC primitive for atomic swaps between Ethereum and Bitcoin, it make sense to use the same SHA256 hash function used by Bitcoin. The snippet of the actual implementation is on Listing 5.1 taken from our implementation for the [BS22b] paper.

5.2.3 Security

The HTLC protocol can be informally be proven secure provided I use the following assumptions:

1. the protocol is secure if the only request is its atomicity
2. we are in the presence of rational agents A and B
3. the amount of coin exchanged is less than the necessary amount needed to reorganize one of the chains.

Note that the third assumption is highly dependent from the consensus method of the blockchains. For example, for blockchains with strong finality (as in the case of some Proof-of-Stake based projects) it is impossible to revert or reorganize the chain after a certain number of blocks. On the other hand some probabilistic-finality based chains (as in Proof of Work based blockchains) can suffer from 51% attacks which can be executed "cheaply" (see for example [Nes19]) and therefore the waiting time needed to accept a transaction as finalized must take that into account. However, if we accept those three assumptions, the informal proof given in [Nol13] suffice:

- if either A of B acts maliciously during phase 1, then no transaction has been broadcast and therefore no participant has coins at risk;
- If B acts maliciously after phase 1, i.e. during phase 2, then A can use refund transaction after 48 hours to get his coins back; since A is rational, it has no incentive to act maliciously in this phase

- If A acts maliciously after phase 2, i.e. during phase 3, then B can use refund transaction after 24 hours to get his coins back; since B is rational, it has no incentive to act maliciously in this phase
- After phase 3: since B is rational, it has no incentive to act maliciously in this phase

However, the previously enumerated assumptions do not ensure a correct execution of the protocol in case either A or B are not rational, which does happen in real life. In [TYME21] the authors propose an attack to the HTLC which assumes either A or B are willing to lose their funds if the other participant loses its funds too. In [NKW21] the authors propose an attack where the malicious party, e.g. A, is willing to leave the potential profits to the miners if they can impede B to correctly redeem A's transaction. In the latter attack, A gets its coins while B does not, while in the former both A and B lose their coins. Both these attacks can be done on Bitcoin blockchain. Other attacks, such as in [WHF19], the proposed attack needs the expressiveness of the Solidity language and it can not be implemented on Bitcoin like blockchains. For further detail on how the atomicity property of HTLCs can be broken, see Section 2 of [TYME21].

5.3 HTLC Problems

The HTLC protocol is a peer-to-peer protocol that can be used for the exchange of funds. In addition to the security issues described above, it also has practical problems that currently discourage its use on a large scale. I divided them into two categories. In the first category I put the problems that are related to the time-lock itself. Those kinds of problems cannot be solved since they are intrinsic to the time-lock mechanism: solving them would change the concept of lock and change the protocol. In the other category I put the implementation-related problems. Those kinds of problems are potentially solvable by tweaking some non-essential aspect of the HTLC algorithm. In Chapters 9 and 10 we will see how I tried to solve these implementation-related problems in the course of my PhD. I present the problems in the categories in the subsections below.

5.3.1 Time-lock related problems

Lockup Griefing As described in [HLG20], *lockup griefing* emerges because in the original HTLC coins are locked in smart contracts until the trades executes or the timelock expires. Since timelocks are generally a few hours or days long to ensure the security of the swap, one party can trick the other into locking coins in the smart contract with no intention of executing a trade, effectively creating a denial-of-service attack. In this case the other party has its fund blocked and can not use them to e.g. initate another swap.

First mover advantages We have already seen that A can carry out attacks against B and these attacks are consequence of the fact that A is the initiator and the holder of the secret x. The advantage described here, on the other hand, has financial reasons.

As described in [XAD21], in the event of a price change unfavorable to A, he may decide not to trade anymore. To do this, he lets the 48 hours run out and uses TX_2 to redeem his money. This way A can treat the protocol as if it were an *American call option*.

Without going into too much detail, an option is a financial instrument that gives the buyer the opportunity to exercise a right to buy or sell the underlying asset. In particular, a *call option* gives the holder the right to buy the asset by a certain date, called *maturity date*, at a certain price called *strike price*. Obviously, the holder will exercise this right only if it suits him, that is, if the current price is greater than the strike price. The *put option* gives the holder the right to sell an asset by a certain date at a certain price: the holder exercises his right only if the current price is lower than the strike price. Finally, there are two ways to exercise this right. The holder of an *American option* can exercise his right *within* the maturity date, while the holder of a *European option* can only exercise his right after the maturity date has expired.

Seeing the HTLC protocol under this lens, A essentially has the right to buy 1 $coin_B$ (the asset underlying the option) by paying it 1 $coin_A$. In the event that, at the end of 24 hours (the maturity date), 1 $coin_B$ is worth less than 1 $coin_A$, A can choose not to show x and not redeem TX_3. Conversely, if at the end of 24 hours the price of 1 $coin_B$ is greater than 1$coin_A$, A has a big incentive in exercising

its right to redeem TX_3. B does not have this right because its shares during the third and fourth phases depend on the shares of A. Finally, this is an American option because technically A can exercise its right before the deadline/maturity date of 24 hours.

This is not really an attack since in any case B can redeem its coins using transaction TX_4 and the atomicity of the protocol is respected, but the protocol gives an unfair advantage to A.

5.3.2 Implementation related problems

Slowness The problem just described comes up in a different version even in the case where both participants are honest. The use of time-locks in fact makes the whole execution slow. This is mostly due to the consensus protocol used and to the fact that it is generally not possible to get the endpoint of a block in a sufficiently short time.

This is one of the main reasons why it is not possible to use HTLC for decentralized markets: the user experience is too poor and the markets that tried a similar approach did not have enough liquidity. As a result, HTLC-based methods do not generally make use of platforms.

Scripting language To implement this protocol you need a scripting language capable of creating and verifying hashes and hash functions. This requirement is not so strict: both Bitcoin Script (the scripting language used in Bitcoin) and Solidity and Vyper (the two most popular scripting languages used in Ethereum and EVM compatible blockchains) have this capability.

But not all blockchain projects have this ability. For example, Monero and CryptoNote blockchain descendants do not have a scripting language. That's why it's not possible to use an HTLC-based protocol to create atomic swaps with these blockchains.

Privacy The disadvantage of using a scripting language is a privacy loss of some degree. In fact, in a successful exchange between A and B it is possible for an external observer to find the pair of transactions (TX_1, TX_3) in the two chains using the pre-image x. This pre-image is randomly decided by A so it can not be guessed by B, but becomes an indexing method for cross-chain atomic swaps. Below I briefly explain how this can happen.

Let TX_5 and TX_6 be the transactions of A and B respectively that redeem TX_3 and TX_1 respectively. Then, via TX_5, A releases pre-image x on blockchain BC_B. This information is public, otherwise B would not be able to see and use it. When B uses x in transaction TX_6 to redeem TX_1, then B releases x on BC_A as well. Consequently, the information x is publicly present on both BC_A and BC_B.

Let Eve, or E, be an outside observer who for some reason is following A transactions. Then E sees A coins move into TX_1 and sees that the next transaction, TX_6, spends these coins by releasing the x pre-image. Given the standardization of the HTLC protocol, E is sufficiently sure that there has been an exchange with another chain. While it is true that there are approximately $12,000$ different coins, the main blockchains/L1s are relatively few and it is possible on a practical level to look at these to find in which other blockchain the same pre-image is located. Also note that it is not necessary to do a complete scan of the other blockchains: given the time limit of the exchange, it is possible to look only at the transactions in the blocks created in this time interval.

For this reason a simple HTLC cannot be used as a method of obtaining privacy. we will see in Chapter 9 and Chapter 10 how I solved these problems.

Chapter 6

Blockchain and Commerce

6.1 Introduction

Since around the last years of the 80s, commerce has gradually shifted to e-commerce. First using catalogs and phone-numbers to order products, then by using internet-based portals. It is easy to see how Ebay and Amazon have paved the way to the current way of making purchases. Multiple business reports highlight that e-commercing is a real trend, which suggests it is highly probable that using e-commerces will be the *de facto* standard way to make purchases.

Some reports speculate that the new wave of commerce will be aided by Augmented or Virtual Reality (abbreviated to AR and VR respectively). This is evidenced by partnerships between established and complementary brands.

An example is the Meta-Luxottica partnership[1]. Meta is a multinational company which controls big advertising and e-commerce venues (both Meta owned companies Facebook and Instagram currently give advertising and direct-purchase capabilities to both vendors and buyers). Furthermore, Meta is currently developing a web3-based platform called Metaverse. On the other side of the partnership, Rayban is a widely known glasses brand owned by Luxottica, especially famous for its sunglasses. Recently Rayban proposed new products under the category *smart sunglasses*. Beside being sunglasses, they are capable of taking photos and automatically sharing them on Instagram. Similar capabilities were previously developed by Google Glass, but the project never really succeeded.

It is easy to envision a future were similar capabilities can be used by agent A to take a photo of item I belonging to agent B and automatically make an image-based search on the internet to buy it (products such as Google Lens already give users this capability, but they are smartphone based currently, which creates some friction in the process). If merged with AR and VR capabilities, smart glasses can help the buyer make decisions by testing the product, maybe in a web3-based platform such as Metaverse. Moreover, the products can be strictly digital, such as NFTs that represent a tool for a web3 videogame, or a file with some important information we don't want to be reshared (a concept sometimes called *soulbound*)[2].

While the latter speculation is a medium-to-long-term possibility, the former one (i.e. buying and selling using a blockchain or web3 platform) is currently operated by some markets even if they comprise a fraction of the whole commerce. But since it is something emerging, it is important to understand and develop tools to facilitate in a way that benefits both producers and consumers.

6.2 New Interoperability Needs

If products have to transit from central hubs to homes distributed over the territory, it is important to have a good tracking system which is open and capable of supporting digital products natively. Furthermore, in order to remove the obvious centralization problems deriving from non-open standards (as in the case of the products and partnerships mentioned in the previous section), we need this tracking system to be as permissionless and open as possible.

[1] See the press release here: https://www.essilorluxottica.com/highlights2021/ray-ban-stories

[2] While there is not currently an academic paper on the matter, the concept is highly debated in forums, for a summary see for example https://vitalik.ca/general/2022/01/26/soulbound.html

Multiple blockchain projects are (nearly) like that. For example, both Bitcoin and Ethereum are permissionless, open and borderless. It can be argued that it is possible to have private and (pseudo) anonymous transactions on them also. Therefore it seems natural to use these settlement layers to keep records of deliveries.

If that is the case, then, we have at least one real life process which is enabled by interactions with a blockchain. In other words, we have one case of Blockchain and Reality interacting with each-other: this is why I called Part III of this thesis *Blockchain-Reality Interoperability*.

In other words, we have one case of Blockchain and Reality interacting with each-other: this is why blockchain-based commerce can be seen through the lens of blockchain interoperability and my research on it touched on this theme. In fact, Part III of this thesis deals with the protocols that I created with the goal of mitigating the problems described in this chapter.

6.3 Practical Considerations

The new exchange system currently developed by private companies, while undoubtedly useful, has some undesired consequences. One of those consequences is the *de facto* loss of privacy and security. Today, services that centrally collect and store users' data have a far wider reach for sharing that data than they had before the internet. Third-party services can use data both for legitimate and non-legitimate purposes, and an aggressive sharing of data increases the probability of non-legitimate uses. Examples of non-legitimate uses are unfair prices or insurance premiums, stigmatization of people and, in the worst case, the unfair punishment of people in non-honest states [GGV20]. Furthermore, any service that stores data centrally is potentially the target of malicious attacks. In this regard, then, private data is a liability for both the user giving it and the service collecting it. In this send, it would be better for all the parties involved not to have the data in the first place.

Nowadays there is not a private or anonymous method or protocol for the shipping of some good from the merchant to the customer: generally the shipping still forces customers to share their personal data (e.g. their address or their identity) with the service they are buying from. The result is that people still have to trust services not sharing their data with other data collectors even though they use a blockchain based payment system.

Proposed methods to exchange goods using a blockchain for both the exchange of value and the shipment agreement still require private data sharing or additional trust, as we see in Section 6.4. Note that the underlying assumption is for delivery packages to be delivered by non-humans devices, e.g. drones[3]. In this case, the delivery itself has to as much non-interactive as possible, while still maintaining a trackability and enforcing non-deniability of receipt.

To accomplish that, some methods rely on tracking with the results posted on a blockchain [EJN17]. Generally, this involves external objects (typically a GPS) to signal the position of the package. In those settings, the GPS operates like some form of trusted oracle. This is both a trust and a privacy problem. In fact, both the company supplying the product and the client receiving it have to trust that nobody tampered with the GPS. Furthermore, a throwaway GPS sensor can be more expensive than the purchased item the transporter is carrying: this makes transportation costs higher than the item's cost. Therefore GPS-based tracking are not feasible for inexpensive items.

In the following we present current solutions found in the literature and we highlight their problems. Chapter 13 deal with a solution to these problems I proposed during my PhD.

6.4 Current solutions

While there are multiple papers about the use of a blockchain system on a supply chain (see e.g. [JSK19] for a survey), I decided to analyze only those papers that explicitly study the use of a transporter.

6.4.1 Proof of Delivery

In [HS18], Hasan et al. analyze what they call a Proof of Delivery system to trade and track sold items between two parties. The system relies on five agents. The first three agents are directly involved

[3]See for example this video on how packages are currently in some parts of the world: https://youtu.be/qcsszdkjlXg?t=18

with the shipment of the item and they are the Seller, the Buyer and the Transporter. The others are external parties not directly involved in the exchange, they overview the process. Those are the Arbitrator and the Smart Contract Attestation Authority. The Arbitrator is a trusted third party involved in case of a dispute and solve the issues off the chain. The smart contract authority is responsible to attest that the smart contract complies with the terms and conditions signed by the involved parties in the agreement form. Each involved party puts an equal deposited collateral which he risks to lose if he behaves maliciously.

In the solution proposed by the authors the third parties (the arbitrator and the SC authority) are not prevented from colluding with one of the parties. In particular, the arbitrator handles the data off chain, so there is no transparent way to inspect his judgment. Furthermore, both the systems require that all parties, arbitrator and the smart contract authority included, know both the physical address and blockchain address of the buyer, so privacy is not guaranteed.

6.4.2 Lelantos

The solution proposed by Al Tawi et al. [AEYG17] also uses a smart contract deployed by the Lelantos system itself to manage the shipment. A single smart contract is used by all customers, merchants and couriers. A customer C is able to redirect shipment between different couriers by using a specific smart contract function. C sends new delivery addresses in encrypted form using the long term public key of the currently designated delivery courier. The public keys are voucher ed by Lelantos itself.

The customer C does not declare in advance which couriers he will use. Furthermore, C will not contact any Currier before the shipment. While this process achieve anonymity for the customer C, the Lelantos protocol is interactive and requires both C and all the delivery couriers to pay attention to the delivery smart contract.

6.5 Standing issues

While there are different solutions related to this problem, I identified some issues which are not addressed in the literature:

- effective decentralization and trustlessness
- tracking-related privacy
- protocol interactivity
- multi-token commerce

The following subsections introduce and explain these problems. In Chapter 13 I present a protocol that solves them.

6.5.1 Effective Decentralization and Trustlessness

The protocols presented earlier all suffer from at least one point of centralization. The Proof of Delivery protocol [HS18] introduces agents which, although there are more than one, can easily collude, e.g. the Smart Contract Attestation Authority can easily collude with the Arbitrator. This introduces an obstacle for both the buyer and the seller. In fact, the buyer must necessarily trust the seller since in fact the protocol is not trustless. The seller must trust the buyer and in fact part of the protocol provides for the possibility of disputes being resolved through arbitrators.

Moving part of the protocol off-chain is another reason why both the seller and the buyer are in a position where they have to trust a third party, represented by the arbitrator and the smart-contract authority in this case.

Even in the Lelantos protocol [AEYG17] there is in fact a centralization of the transport service. It's all decided by the smart contract manager.

This does not happen in P2T, the protocol I devised to solve the problems described in this section, because the transporter represents an (untrusted) third party that intervenes in the protocol after the sales agreements between seller and buyer. The transport protocol is detached from the sales protocol and allows all parties to operate the transport in a trustless manner, as explained in Chapter 13.

6.5.2 Tracking-related privacy

In the Proof of Delivery protocol there is not mention of user privacy, so we can assume that the authors of this protocol do not considered it. On the other hand, in Lelantos the authors do acknowledge that the buyer has some privacy, since it can retain a high degree of anonymity[4].

The privacy-related main issue is due to the possibility of transaction-tracking by external observers. Both proof of delivery and Lelantos do not obfuscate any transaction. Indeed, both protocols use a known smart contract whose goal is only to process those transactions. Obviously, an external observer will know a great deal of what is happening by just monitoring the smart contract. Note that this is not intrinsic in the Ethereum technology, since the protocol could be tweaked by requiring a new smart contract for each instance.

This issue is solved in P2T by using a new deterministically derived address for the seller based on a base-address publicly published, the item bought by the buyer and a shared one-time key. Note that since the seller address is derived deterministically, both the seller and the buyer can derive it without any additional interaction. This helps attain a lower lever of interactivity as seen in the next section.

6.5.3 Protocol interactivity

Both the Proof of Delivery and the Lelantos protocol rely on constant interaction between parties to complete the protocol. In an environment that wants to be as private and anonymous as possible, this creates major problems. It is well known that by definition an identity is automatically created during an interaction of at least two rounds: any protocol that requires more than one round of communication from a participant cannot guarantee the participant's anonymity.

In addition to privacy and anonymity issues, the interactivity of a protocol ruins the user experience to the point that it cannot be reasonably secure. For example, nowadays many private keys are kept in specific devices generally called *cold wallets* that are never in direct contact with the Internet and consequently must be kept in protected places since physical possession of the device gives the ability to sign transactions. Generally these protected places are difficult to reach, either because they are hidden or because they require the participation of multiple users to access them (think of a threshold signature mechanism in which several partners are needed to sign a transaction). An interactive protocol requires repeated use of these devices, which therefore cannot be securely stored for the duration of the protocol: being a physical product delivery protocol, the protocol can last for days or even months in borderline cases. This is not acceptable if you want the user to keep his security model of funds intact.

We will see in Chapter 13 how Pay-to-Transport [BS20b] solves this problem as well.

6.5.4 A multi token environment

One of the first problems perceived by the users regarding using multiple blockchains for payment purposes was being able to interoperate between them, i.e. exchanging tokens from one blockchain for tokens in the another. I already mentioned that early methods to interoperate were based on centralized methods (see also [But16]), such as online exchanges. In Section 7.6.2 I explained that this comes at a cost: the problem with having some funds held in a few addresses controlled by one entity (the *custodian*) is that you are subject to the power of that entity. In an ecosystem based on pseudo-anonymity, this is dangerous since custodians are not always trustworthy and/or legally traceable, which creates a very strong incentive to perform malicious human actions on the funds.

Multiple markets imply different exchange rates between the same token pairs. To take advantage of this situation, new services, called *aggregators*, provide services on exchange protocols and networks. The goal is to give users the ability to exchange their tokens at the most favorable exchange rate [iT21].

To date, these exchanges are done manually. This is not a problem when the user wants to exchange funds for reasons of financial speculation nature, but it is a problem when a buyer B wants to pay a merchant M who does not accept the tokens owned by B. In this case, B has to manually go to an aggregator or a DEX to exchange the owned tokens for the tokens accepted by M. This, besides being a waste of time and ruining the user experience on the merchant's application, is also an economic problem since it is necessary to spend an additional sum for the exchange fees.

[4] In general privacy is different from anonymity, but in this case having anonymity delivers also a high degree of privacy.

Interestingly, there is no direct solution in academic literature that addresses this practical problem. While ad-hoc solutions do exists (.e.g point of sales for specific cryptocurrencies, such as Bitcoin, Monero or Zcash), the blockchain space lacks a general solution. We will see in Chapter 14 how the Universal Token Swapper (UTS, [BSS21]) I created solves this problem.

Chapter 7

Blockchain interoperability & Other Problems

7.1 The Fair Exchange Issues

Assume a person currently owning bitcoins but, needing more privacy for his operations, he has to exchange those bitcoins for another currency, such as the one powering the ZCash blockchain[1]. The only reliable way to do that is using centralized online cryptocurrency exchanges. But using them is a threat to the idea of decentralization and censorship resistance ideals highly felt by the blockchain community. Since centralized exchanges are already present on the market and there are no cryptographic problems to solve, I do not treat them in this thesis as a potential interoperability method. Here I will only present methods for interoperability that "decentralized"[2]. Therefore, from now on, every time I say interoperability I implicitly mean *decentralized interoperability*.

The term interoperability is relatively a new one. Before blockchains, the capability of data exchange between two systems was generally called *fair exchange* in academic literature. Therefore all results comes using this terminology. In the following I present them. These are important to understand the underlying situation and consequently the constraints.

7.1.1 Fair Exchange

An exchange is fair if no participant can cheat. More formally, as explained in [Aso98]:

Definition 7.1.1 *An exchange is fair if at the end of the exchange either each player receives the item it expects or neither player receives any additional information about the other player's item.*

In the following, assume two parties P and Q have items i_P and i_Q respectively and they want to exchange them. Assume also that for each item $i_j, j = P, Q$ there is a description $d_j, j = P, Q$. This description represents any knowledge deriving from having i_j.

Note that a "simultaneous exchange" can not be done on the internet, assuming it is a asynchronous or semi-synchronous medium (i.e. assuming that there is a $\delta > 0$ between the sending and receiving

[1] Here I do not regard ZCash as the "most private" blockchain and the example should not be taken as an endorsement. I use this blockchain as an example since the value proposition of the ZCash project is guaranteed privacy if used correctly. Yet there are concerns on how effectively privacy is maintained in these blockchains. Other competitors such as Monero are potential alternatives, but with similar caveats. The privacy problem is a complicated one and no single solution is currently able to tackle and mitigate all the threats. I decided to use ZCash as the example only because I explicitly mentioned the project in the previous section.

[2] The word "centralized" has become a little bit of a *buzzword* nowadays. In the course of this thesis when I talk about a centralized/decentralized method I will explain what I actually mean by that. However multiple academic papers tackle the problem of centralization in the blockchain space. A good way to talk about this topic in a meaningful way is to see the blockchain as a stack of layers and talk about centralization/decentralization at the specific layer. So for example in this case when I say that a method is centralized I mean a centralization at the application level which has nothing to do with centralization arguments on the consensus layer: being able to exchange tokens between blockchains with one, many or no intermediaries is a completely different topic from decentralization that emerges from the behavior of miners and validators at the consensus level. I will elaborate on this concept in Section 7.6.2 and more details can be found in [SBFG20, GSM+20].

of a message and that δ is potentially infinite in the case of an asynchronous network). Therefore it is not possible to assume that P and Q exchange i_P and i_Q *at the same time*. To give some context to the possible ways P or Q can cheat, I give a brief list:

- *Non-sending*: after receiving i_P, Q aborts the protocol. A protocol that allows for this possibility can not be considered fair;
- *Fake-sending*: after agreeing to exchange i_P and i_Q, P sends $i'_P \neq i_P$ to Q while still receiving i_Q from Q. A protocol that allows for this possibility can not be considered fair since in practice Q has not received anything;
- *Information leaking*: after receiving i_P, Q aborts the protocol. By the nature of the protocol, i_P is returned to P, even if Q had it for a brief period. A protocol that allows for this possibility can not be considered fair, since Q has knowledge of d_P and therefore Q gained something while P has not.

Definition 7.1.2 *A protocol that solves the fair exchange problem has four properties:*

1. Effectiveness*: the protocol is effective if when participants P and Q behave correctly, at the end of the exchange party P has i_Q and Q has i_P.*
2. Strong Fairness*: when the protocol has completed, either P has i_Q and Q has i_P or neither party has any additional information about the other's item (e.g. P doesn't have i_Q nor d_Q).*
3. Timeliness*: the protocol is timeliness if it completes in a finite amount of time.*
4. Non-repudiability*: at the end of an effective exchange, P is able to prove that the item i_Q originated from Q and that Q has received the item i_P from P. The vice versa is true for Q.*

Some notes are needed. I don't take into account other properties such as privacy and anonymity right now. Also, note that Non-repudiability is not strictly required for an exchange to be considered fair: its inclusion is mainly for possible subsequent disputes. The strong fairness property is sometimes substituted with the weak fairness property which says that the exchange is completed successfully even if a party misbehave and a judge intervenes in order to make things right. Finally, I treat the fair-exchange problem for only two parties. This is general enough for our purposes: even if I will explain methods that involves more than two parties, they will be grouped into two "teams". So I will treat the multi-party case as a two entities case and exploit the two-party fair exchange theory for it.

Proposed solutions to solving the fair exchange problem fall into three categories: use of a third party, trusted and online; gradual exchange of secrets (depending on the context they may be also called *items*) privileges; optimistic exchanges: I review these solutions (Section 7.1.2). Unfortunately, no such protocol can guarantee a fair exchange and we will see it in the impossibility result is Section 7.1.3.

7.1.2 Proposed solutions

In the following I give a brief explanation of the proposed solution. A summary of all the information can be found in Table 7.1.

Third party protocols

Protocols based on a trusted third party are the easiest to design and build. In fact, every problem is delegated to the third party that, being trusted by definition, is able to solve everything. For example, in the case of the non-sending way of cheating, the third party simply returns i_P to P after a time-out. To prevent the case of fake-sending, the third party checks that the received data conforms to the description or to the agreement. This way of proceeding also solves the case of information leaking.

These strong advantages are matched by strong disadvantages. The third party must be trusted and this means that, at a practical level, few organizations can guarantee or demonstrate this type of trust, which in any case can never be total [Sza01]. This causes a consolidation toward a few available choices, and this narrow group has incentives toward service degradation. Examples may be corruption or high and unjustified fees in exchange for service, censorship of information, denial of service as a *de facto* punishment, etc. In addition to that, the third party must be online for the whole duration of

Method	Brief description	Uses
Trusted Third Party	an external party handle all the exchange, it is assumed (trusted) that this party handle exchanges honestly and perfectly	online custodial exchanges
Gradual releases	the parties release secrets gradually; if a party misbehave the protocol aborts	*N/A*
Optimistic exchanges	the parties assume each other as honest; if a participant misbehave a judge intervenes	Rollup; Escrow

Table 7.1: the table summarize the different proposed *workarounds* to the fair exchange problem that still do not solve it.

the protocol: it is inherent in the definition of trusted third party the requirement of liveness of the offered service.

In the blockchain field the trusted third party is generally represented by centralized exchanges that are online. This method of interoperability gives the possibility to exchange funds, but not to exchange other types of programs such as smart contracts.

Gradual exchange protocols

Protocols of this type are based on the fact that the exchange takes place using multiple rounds of communication [Dam93]. For each round of communication a small part of the entire secret is exchanged by the participants. Each piece of the secret is verified by the receiver who makes sure that it is consistent with the entire secret he intends to receive. In the event that one participant (e.g. P) aborts the protocol or does one of those types of cheating I listed earlier, the other participant (e.g. Q) has the option of aborting the protocol knowing that the one who cheated (P) has no more information than he (Q) does, or that the difference in information is trivial anyway.

This kind of protocol is mainly used to sign contracts and to certify communications [Dam93]. As far as I know, this type of protocol is not used on the large scale for exchanges between blockchains. This is because the necessary communication is very expensive in terms of message exchange: on the one hand it cannot be done on the blockchain itself since every information exchange, i.e. a transaction, involves the payment of money, and on the other hand it cannot be done outside the blockchain since in this case it is not possible to take advantage of all the guarantees of integrity and decentralization that the blockchain offers instead.

Optimistic protocols

Optimistic protocols [ASW98] can be considered a middle ground between protocols with and without a trusted party. In fact an optimistic protocol is optimized for the case in which both participants behave honestly. In this case, there is no need for a trusted third party to intervene and therefore the protocol can be called decentralized, or peer to peer. On the other hand, if one of the participants turns out to be dishonest, the other participant has the option of appealing to a third party, called a judge, who intervenes to resolve the issue. The assumption of these protocols is that the very fact that there is the possibility of appeal discourages both parties from being dishonest: the judge acts as an incentive for honesty even if he or she does not participate directly in the protocol. In particular an optimistic protocol uses the weak fairness property instead of the strong one.

Such protocols are used explicitly for interoperability between different layers of blockchain. The most famous case is probably the optimistic Rollup [Adl19] between Ethereum compatible blockchains and the Layer 2 protocols. A rollup is a collection of N transactions that move the state of a blockchain from a state S_1 to a state S_N[3]. In an optimistic rollup a committee of validators are incentivized to

[3]there are essentially two kinds of Rollup. The zero knowledge based rollups (also called ZK-rollups) and the optimistic rollups. In a ZK-Rollup, a (succinct) zero knowledge proof is used to prove that every state change is valid without exposing the transactions themselves. The proof is built and proposed by a validator.

produce fraud proofs in case of misbehavior of the validators: these validators are a further decentralization of the concept of "judge".

Optimistic protocols are also used in cross-blockchain exchanges. The classical case is a 2-of-3 escrow service where an arbitrator or a judge is one of the three parties involved in the exchange and that can sign a transaction starting from the escrow. If both parties are behaving honestly, then they can sign a transaction from the escrow service to the intended recipient, while if one of those parties is misbehaving then the judge can intervene and sign the transaction from the escrow to the honest party. Since the judge only intervenes if one of the party is misbehaving, then the protocol can be considered optimistic.

7.1.3 Impossibility results

Since there are so many methods of obtaining an exchange, each with its own assumptions, merits, and flaws, it is normal to ask whether it is theoretically possible to obtain a fair exchange between the parties, where by "fair" I mean an exchange that respects the three properties of Definitions 7.1.1 and 7.1.2.

The question is answered in the negative, with an impossibility result due to Pagnia [PG99]. to introduce it, let's define a concept I will use throughout this thesis:

Definition 7.1.3 *A* rogue node *(or* misbehaving node*) is a node that is not honest. In other words, a rogue node is a node that does not follow the protocol.*

The impossibility result then can be stated as:

Theorem 7.1.1 *It is not possible to have a fair exchange that is tolerant of rogue nodes without a trusted third party in an asynchronous system.*

In the following I give a brief description of how Pagnia proved this results.

The first thing is to recognize that there are two different models for the interaction of parties with a system. A system is considered *closed* if it does not interact with its environment. Therefore the users are considered to be part of the system and they interact with it. Conversely, a system is *open* if its behavior is influenced by the surrounding environment (which has to be specified). In this case, users are considered to be outside of the open system. Since in distributed systems and blockchains in particular no external data can influence the behavior of the system[4], the natural thing to do is to consider the blockchain as a closed system.

In order to get this impossibility result, Pagnia starts to formalize the problem previously described by Asokan [Aso98]. The process takes three steps: the first step is to put in logical propositions the definitions of effectiveness and strong fairness. The second step is to prove that the strong fairness problem is at least as difficult as the consensus problem. The last step is to use the impossibility results on consensus problems to prove that strong fairness problems are impossible as well. In the following I briefly summarize the necessary steps, for details see the report of Pagnia [PG99].

Step 1: definitions of effectiveness and strong fairness Let's assume two participants A and B in a closed system, i.e. a system that they can not perturbate in any way[5]. The first thing Pagnia does is to formalize the properties of Effectiveness and (Strong) Fairness [6].

Effectiveness is the property for which either both parties receive the other's item when they both behave non-malevolently (i.e. in a honest way) and the items are the ones which are anticipated (meaning the item description is the one agreed-upon before the protocol instantiation), or if the items

[4]It can be argued that *oracles* do let blockchain to be influenced by events in the real world, but since we do not use oracles in our treatment, we do not treat this case.

[5]As explained in [Lam89], modeling the system as a closed one means the parties are part of it, while in a open one the parties are outside of it, i.e. part of the environment, and have a representative that feeds them data. Pagnia in [PG99] assumes the distributed system is a closed one, i.e. participants are part of the system itself.

[6]The property of timeliness is not formalized in the Pagnia report because in order to formalize questions based on timeliness it is necessary to use the operators of Pnueli [Pnu77] and of Cahndy and Misra [CM88] which would weigh down the explanation without improving the understanding of the concept. Since the property of timeliness on a practical level is equivalent to the property of liveness, the reader may assume the latter when discussing timeliness.

Description	at P	at Q
exchangeable item store (input)	i_P	i_Q
description of wanted item store (input)	d_Q	d_P
partner store (input)	Q	P
exchanged item store (output)	e_P	e_Q
indication of malevolence	m_P	m_Q

Table 7.2: The notation used in the Pagnia report

are not the ones expected then both partners will abandon the exchange by meeting an aborted token. With reference to the Table 7.2 the property can be expressed as follows:

$$\begin{aligned}(d_P = \text{desc}(i_P) \wedge d_Q = \text{desc}(i_Q) \wedge \text{stable}(m_P = m_Q = \text{ false }) \Rightarrow e_P = i_Q \wedge e_Q = i_P) \wedge \\ (d_P \neq \text{desc}(i_P) \vee d_Q \neq \text{desc}(i_Q) \Rightarrow e_P = \text{ "aborted" } \wedge e_Q = \text{ "aborted" })\end{aligned} \tag{7.1}$$

The (strong) fairness property rules out any unwanted behaviors where one party gains something at the expense of the other party. Formally:

$$\neg((e_P = i_Q \wedge e_Q = \text{ "aborted" }) \vee (e_Q = i_P \wedge e_P = \text{ "aborted" })) \tag{7.2}$$

Step 2: prove the relation between the consensus and fair exchange problems The strong fairness problem can be compared to a consensus problem. In order to do that the author creates a consensus algorithm based on an exchange of an item. By doing so, the author shows it is possible to create a consensus algorithm for leader election (see Section 2.2) from a fair exchange, proving that the consensus problem is at least as difficult as the fair exchange problem[7].

To see why the consensus algorithm and the exchange are related, note that in the leader election subroutine of a consensus algorithm every node that is claiming leadership for the epoch has to provide a message to signal their availability. If I assume a consensus algorithm between only two parties P and Q, then I can model the leader election as P and Q that exchange messages, or more generally an item. Therefore leader elections are a special case of fair exchange.

This is proved by the following algorithm:

```
function consensus(π ∈ {0,1}) returns δ ∈ {0,1}
  local variable t ∈ {0,1,"aborted"}
begin
  t := strong_fair_exchange(π, desc(1)); {* 1 *}
  if t ≠ "aborted" then return 1 end
  t := strong_fair_exchange(π, desc(0)); {* 2 *}
  if t ≠ "aborted" then return 0 end
  if π = 1 then
    t := strong_fair_exchange(1, desc(0)); {* 3 *}
    if t ≠ "aborted" then return 0;
      else return π end
  else {* π = 0 *}
    t := strong_fair_exchange(0, desc(1)); {* 4 *}
    if t ≠ "aborted" then return 0;
      else return π end
  end
end
```

Figure 7.1: Pseudo-code of the consensus algorithm proposed by Pagnia in Figure 2 of [PG99]

Formally:

[7]The author conjures that it is not possible to build a fair-exchange algorithm from a consensus one, and therefore the fair exchange algorithm is strictly more difficult that the consensus problem. Since that has not been proved yet, I assume the problems are at least at the same difficulty.

Theorem 7.1.2 *Let S be a system which solves the strong fairness problem in a given system model. Then there exists a system C which solves the consensus problem in that same model.*

Step 3: prove the impossibility of the strong fairness problem A well known impossibility results due to Fisher in 1985 on distributed consensus says that [FLP85]:

Theorem 7.1.3 *There is no asynchronous, deterministic, 1-crash-tolerant consensus protocol.*

where n-crash tolerant means that it works even with n or less rogue nodes (Definition 7.1.3.
It is easy to see that a fair-exchange based consensus algorithm is deterministic since no randomness is used in the consensus algorithm of Figure 5.1. And we already know we are in a asynchronous network because of how it is generally modeled communication network and the fact that Asokan in [Aso98] assumed no timing bounds on message delays.

Therefore, by using the (contraposition of) Theorem 7.1.2 and Theorem 7.1.3 I can show that the strong fairness problem is impossible. More formally:

Theorem 7.1.4 *There is no asynchronous strong fair exchange protocol tolerant against misbehaving nodes*

As in any impossibility result, weakening any assumption may invalidate the theorem. We already saw a way to weaken the strong fairness assumption into the weak fairness property and the consequent optimistic-exchange protocols. In the course of the chapter I will use another way to weakening the assumptions, namely the asynchronicity setting. I will introduce time constraint, time locks and hash locks to create methods of exchange which respects the strong fairness property[8].

In the following subsections I present how interoperability works for two blockchains specifically: Bitcoin 7.2.1 and Ethereum 7.2.2. Projects not mentioned in this section fall into one of those two categories.

Interoperability for the Bitcoin blockchain means interoperability with other blockchains since every cryptocurrency and every token is always created outside the Bitcoin blockchain[9].

In the following I present the details.

7.2 Blockchain interoperability

7.2.1 Interoperability in Bitcoin

One of the first ideas that circulated when multiple blockchains started to spawn was the concept of sidechains, formalized by authors in the Blockstream company [BCD+14]. The goal of their two-way pegged[10] sidechains was to facilitate the balancing of work between multiple chains, and not a general interoperability method: ideally the coins would be equivalent and without difference in value. In practice, this first attempt was a way to improve the scalability problem of a blockchain, but by doing so the authors also created a way to exchange funds between blockchains. The authors acknowledge that and they explicitly argument in favor of the creation of multiple blockchains in order to create a competitive environment so that the project with the best set of features would prevail on the others. This model was bitcoin centric, since no other blockchain solution was production-ready at the time,

[8] In order to do that I will implicitly assume that parties are in an open system. This requirement is a very weak one since assuming parties operating on the blockchains cannot meaningfully participate in the correct functioning of the whole system is equivalent to say that neither party has more than half of the computational power (in PoW settings) or resources (in PoS settings) to hijack the consensus mechanism.

[9] This isn't entirely true. Some projects like colored-coins tried to create tokens inside the Bitcoin blockchain. Yet those kinds of projects are actively discouraged by the developers of the blockchain since those projects put a heavy load on the blockchains. However, small projects like Bisq do have colored-coins in order to finance their working. In any case the financial weight of those niche projects is negligible with respect to the financial weight (sometimes referred to as *Total Value Locked*) of tokens created in other blockchains such as Ethereum, Tezos and Algorand, and therefore I do not treat them in thesis.

[10] A currency peg between two currencies is a predetermined and fixed exchange rate between them. In the case of the bitcoin sidechain the peg was 1-1, meaning a bitcoin was worth the same as a sidechain-bitcoin. The system is two-way meaning that it is possible to exchange coins from bitcoin to the bitcoin sidechain and back. This is different from previous ideas that circulated in forum posts at the times that were limited to a one-way, meaning the only ability to take out bitcoins from the mainchain (i.e. the original bitcoin blockchain) to a sidechain without the possibility to come back.

therefore it arguments in favor of *bitcoin-like* blockchains competing with each others. The work on that paper evolved in the work of Strong Federations a couple of years after [DPW+17]. That work would create the Liquid sidechain, a private blockchain managed by the Blockstream company. To transfer the funds between the bitcoin blockchain and the Liquid sidechain, a committee of notaries is used. The committee is responsible to receive the coins on the bitcoin blockchain and to create an equal amount of liquid-bitcoin coins on the Liquid sidechain. The process works similarly in reverse. Such a protocol improves on the centralized exchange problem by expanding the number of entities in control of the funds from one to many (specifically the Liquid sidechain uses a committee of eleven parties). Yet the interoperability problem is not completely solved, since those parties can still cooperate (or, assuming maliciousness, "collude") and censor the funds. Therefore the Liquid sidechain has *less* security guarantees than the bitcoin blockchain.

The set-of-notaries method proposed for the Liquid sidechain is in its essence fairly common to exchange tokens between blockchains. Yet it is not the only one. Other methods are atomic swaps and decentralized exchanges.

Atomic swaps are peer to peer (P2P) methods to exchange tokens. To give an intuition of the method, assume $coin_a$ is the native currency[11] or token of blockchain BC_a and $coin_b$ is the native currency or token of blockchain BC_b. In an atomic swap Party A has some $coin_a$ and it wants to exchange them with another Party B for some $coin_b$. To do that A gives $coin_a$ to B on BC_a and receives $coin_b$ in the other blockchain BC_b. Party B does the opposite. The exchange is atomic in the sense that either the exchange is successful or the protocol terminates with Party A owning the initial amount of $coin_a$ and B owning the initial amount of $coin_b$. This method has the advantage that does not need any platform *per se* and can be used for trustless over-the-counter (OTC) trades. One problem with this method is that it is intrinsically illiquid[12]. This and other problems are explained in Section 5.3.

Decentralized exchanges for bitcoin are currently not a reality. The main reason is that the decentralized exchanges are not a multi-chain oriented method nowadays. Decentralized exchanges are currently used in Ethereum-like blockchains[13] and in general in smart contract enabled blockchains. We see that in the next section.

7.2.2 Interoperability in Ethereum

As mentioned, in the Ethereum ecosystem the interoperability needs are a superset of the ones in Bitcoin. Supporting smart contracts, users on the Ethereum network have the possibility to create multiple tokens inside the blockchain. Examples of this behavior are the ERC-20 standard [VB15] and the ERC-721 standard [ESES18]. The first standard represents so called *fungible tokens*, or more simply tokens: these tokes are interchangeable between each other. An example of fungible product is cash, where each banknote is equivalent to any other banknote representing the same value. The latter ERC defines *non-fungible tokens*, i.e. tokens that are different from each other. Non-fungible things are easier to reason about, since the majority of objects are not fungible. For example two houses may be in the same building, but they are very different things and people legally owning one can not go to live in the other.

These standards give the possibility to the creator to effectively create new fungible and non-fungible (respectively) tokens. The fungible tokens can represent coupon points, voting rights or real currencies such as the case for stablecoins[14]. On the other hand the non-fungible tokens (NFTs) currently represent art pieces or are used for access control mechanisms.

Consequently, there are two kinds of interoperability methods in the Ethereum ecosystem. One are the cross-chain methods similar to those previously explained in Section 7.2.1. The other is the exchange of token *within* the Ethereum blockchain, which is the one I explain in this section.

The narrative around the exchange of tokens in the Ethereum blockchain originally was downstream from the use of decentralized applications (dApps). The idea was that, in order to use different dApps

[11] See Footnote 15.

[12] Illiquid refers to the state of a stock, bond, or other assets that cannot easily and readily be sold or exchanged for cash without a substantial loss in value. For more details see e.g. https://www.investopedia.com/terms/i/illiquid.asp

[13] Ethereum-like blockchain are EVM-compatible blockchain, i.e. blockchain that run a virtual machine compatible with the Ethereum Virtual Machine.

[14] A so-called "stablecoin" is a token pegged to a fiat currency such as the US dollar or the Euro, i.e. its price is fixed to the unit of the related fiat currency. In that sense, the coin is "stable" with respect to the value of the fiat currency it is pegged to

or decentralized autonomous organizations (DAOs), people had to have the token that represented the application they wanted to use. Consequently people that had ethers[15] and wanted to have the utility token of a dApp or the voting rights of a DAO needed a way to exchange these tokens. As mentioned earlier, before the year 2019 practically the only way to do those exchanges was to go on a centralized exchange. The reason why decentralized exchanges were not mainstream at the time is explained below.

Exchanging tokens within the Ethereum blockchain has less assumptions than in the bitcoin case, since there is no actual change between chains. In fact exchanging tokens within a blockchain essentially means making the right calls to the smart contracts, using the Ethereum blockchain as settlement layer. Consequently the creator of an exchange protocol for tokens in Ethereum does not have to check the underlying security assumptions of the blockchain, such as consistency of both distributed databases after the exchange or liveness: the technical needs are intrinsically present in the Ethereum blockchain already (to better understand the consensus related difficulties, see Chapter 2). For this reason decentralized exchanges capable of exchanging tokens created inside the Ethereum network where born relatively soon. The problem of these decentralized exchanges (DEXs) was *liquidity* related, not technical.

to solve this problem a new kind of liquidity provider was created, called autonomous market makers (AMMs). An autonomous market maker uses mathematical formulas to price a token with respect to any other token. The mathematical formula is called *bonding curve*. The creation of autonomous market makers (AMMs) created the needed liquidity and the majority of token exchanges (in volume) moved from the centralized online exchanges to decentralized exchanges powered by autonomous market makers. In order to work properly each decentralized exchange needed liquidity pool, i.e. a pool of funds to provide liquidity. While in centralized exchanges these pools are owned by the company behind the exchange, in a decentralized exchange liquidity pool has funds from volunteers incentivized by the possibility of gaining other tokens from getting a percentage of the fees paid by the users of the decentralized exchange. Since everything is governed by smart contracts these exchanges has the backend that is completely decentralized. Of course the interface is generally on a server so this layer of the application is currently still centralized (see Section 7.6.1). Yet, being the majority of the software under a open source license, the users have either the possibility to create a new interface and deploy it on another server or distributed-storage systems, or the possibility to interact directly with the smart contract without using the provided interface.

Note that even if AMMs solved the liquidity problem of exchange, there is still a "spending" problem since having multiple tokens is a financial liability for non-speculators (e.g. merchants or workers). Therefore their goal is to either accept one or a handful of tokens or swap any token instantly. The first way create friction in commerce, which can tamper revenue. The second way is more similar to real life customs (e.g. Point of Sales automatically swap foreign currencies for local currencies when paying in foreign countries). See Chapter 14 to see my proposal on how to create a payment processor that can mitigate this spending problem in the latter way.

Beside being a way to have liquidity on a decentralized exchange, nowadays liquidity funds in the pool also create borrowing and lending services as a way to make the capital in the pools more productive. I do not enter into the details of how those services are provided since they are financial in nature, but I mentioned it because it's important to understand that the creation of decentralized financial services (DeFi) has exasperated an underlying problem that was already present: the scalability problem (see Section 7.6.1). In fact decentralized finance has been considered the first "killer application" of the blockchain ecosystem and, by going mainstream, multiple people and financial institutions started to use the Ethereum blockchain for these purposes. As a consequence, gas[16] cost increased and even the new improvement protocols such as the EIP-1559 that specifically tackled the gas-fee imprevedibility-problem didn't solve the underlying problem completely (even if it made it more manageable, see [LLN+22]). Currently the only way to improve scalability of many blockchains

[15] The *ether* is the native currency of Ethereum. This is different from the bitcoin case where the native currency is called bitcoin (with a small b) and the protocol is called Bitcoin (with capital B). Generally when I talk of token I mean a made-up smart contract-created currency that uses a blockchain as settlement layer secured by its native currency. Sometimes instead of native currency I write *coin* meaning the same thing.

[16] Gas is the unit of account for pricing operations in the Ethereum blockchain. Since the scripting language of the smart contracts is (in principle) Turing complete, miners need a way to put a hard cap on the amount of operations needed to execute the whole set of instructions in a transaction. Making the sender pay for every instruction instead of a forfeit amount ensures that the whole transaction won't be an infinite loop. In fact a miner will stop process the transaction if the amount paid by the sender is not sufficient to complete it.

is the creation of a layer on top the original blockchain on which balance the workload. This idea is similar to the ideas of 2014 that we saw before [BCD+14, DPW+17], but it's assumptions are different. I explain them in the following section where I also explain why scalability and interoperability are related problems.

7.3 Interoperability via Tokenization

There are two main ways to operate an exchange between two blockchains. The first one is by using the paradigm deposit-mint-withdraw where, as the name says, the user deposit some coins on a specified address on blockchain A and then, thanks to cryptographic methods, a corresponding token is created on blockchain B. The user can exchange the token in blockchain B, or withdraw it. To perform this last operation, the user *burn*, i.e. destroys, the token on blockchain B and redeems it on blockchain A. Generally this operation is performed by a service which is generally custodial. Consequently the service is a trusted third party that users trust not to steal their coins or censor transactions. Services are generally registered companies which means that they are forced to do checks like AML/KYC to follow regulations. In the following I give a brief presentation of methods to exchange coins between bitcoin and Ethereum. I made this choice since this couple of blockchain is the most popular and most tested. For a comparison ad a more detailed explanation see [Cal22].

7.3.1 wBTC

Wrapped Bitcoin [NIP19] is a token on the Ethereum network. It was born in 2018 from a partnership between Kyber network, BitGo, and Republic Protocol. The goal of this token is to reflect the value of bitcoins in a 1:1 pegged way.

In this partnership, BitGo is responsible to manage the custody of the received bitcoins from the users.

In the following I introduce details about both the protocol and the actors involved for it to properly function.

Custodian In this partnership BitGo is responsible to manage the custody of the received bitcoins from the users. In this sense the protocol Wrapped Bitcoin is a custodial service. Since it is custodial, BitGo has to provide the necessary transparency to perform auditing procedures on the collaterized assets.

Merchant The merchant is the one who receives the bitcoins from the users and then, thanks to the protocol, creates a token on the Ethereum network. The merchant is an intermediary between the user and the custodian. It is also involved in the reverse process, i.e. in destroying the token in the Ethereum blockchain and send the bitcoin to the user on the Bitcoin blockchain.

User The user is the one who sends or withdraw the bitcoins to the merchant.

The Protocol At first, the user starts the request to the merchant and if successful the user sends bitcoin to the merchant. The merchant then sends those bitcoins to the custodian. The merchant then creates the token on the Ethereum blockchain.

If the user wants to withdraw its own bitcoins, the user asks the merchant to destroy the token on the Ethereum blockchain. Finally the merchant transfers the bitcoins from the custodial to the user on the Bitcoin blockchain.

Governance Currently the management of the wBTC contract is operated through a *m-of-n* multi-signature (see Section 4.3) whose key owners are the member of the so-called "WBTC DAO"[17]. Op-

[17] DAO stands for Decentralized Autonomous Organizations. These organizations are managed by smart contracts and therefore all operation of such an organization have to be explicitly coded. The big advantage to operate an organization through smart contracts is that every action is inherently clear and no interpretation is needed. The disadvantage is related to voting or bribing problems. In fact in a pseudo-anonymous environment there cannot be any reliable proof of humanity. Consequently therefore concepts such as "one person one vote" are meaningless. The topic is fascinating and highly complex, but it is not the topic of this thesis so I don't analyze the concept of the DAOs further.

erators handle the contract and the addition or removal of custodians and merchants through the DAO.

Analysis As can be seen from descriptions of the protocol and the governance, the protocol is easy to use from the perspective of the user and financially secure in the sense that one wrapped bitcoin will always be redeemable with one bitcoin and vice versa. In this sense, the user will not lose financial power (in bitcoin terms) by using this service.

On the other hand the merchant and the custodians are two third parties which have to be trusted. In fact the merchant could in theory refuse to send money to the custodian. This is mitigated by the fact that the merchant is registered in the DAO and therefore risks legal persecution if it behaves dishonestly. The other third-party, the custodian, is effectively managed from a multisignature. While better than a single point of failure, this method is not without problems. The problem of centralization, even if the central authority is shared between multiple parties, lies in the fact that those multiple parties can still collude and steal the funds of the users or censor any redeeming.

Finally, the last problem of this protocol is that users lose their anonymity when they use the service and the custodians need to perform KYC procedures.

7.3.2 renBTC

As we saw, the wBTC protocol has some disadvantages. Another protocol that aims to tokenize the bitcoin coin in Ethereum is the renBTC protocol [PW20]. With this protocol, a user doesn't need to go through a KYC process and it has the assurance of a full open source project. Furthermore the renBTC project doesn't need a central custodian to handle the funds.

DarkNodes The main improvement of renBTC with respect to wBTC is that the former decentralizes the merchant and multisignature cold storage by using a new virtual machine (the renVM). This virtual machine is maintained by a set of nodes called DarkNodes. The DarkNodes are responsible to store the bitcoins and create the renBTC tokens. The DarkNodes are also responsible to burn the renBTC token and make a user redeem its bitcoin. Every DarkNodes requires a bond of $100,000$ REN tokens to run. The bond can be slashed if the DarkNode behaves maliciously or if it is responsible for the loss of assets. In the latter case, the slashed bond can be used to restore the lost assets.

Shards DarkNodes are periodically shuffled into shards randomly. Each shard uses a RZL MPC algorithm to generate a secret ECDSA private key (note that the private key can not be known by any DarkNode by construction) used during the deposit and withdraw.

The Protocol The renVM has two different protocols. One for deposit (called 'lock-and-mint' operation) and one for redeeming coins (called 'burn-and-release' operation). We assume there is an application that has integrated the renVM inside. In the deposit process, a user use the application to generate a bitcoin address. This address is a Bitcoin script that is only spendable by one of RenVM's secret ECDSA private keys generated in the shard. After sending bitcoins to the address, the user receives a "minting signature", generated by the shard using the same secret ECDSA private key. The minting signature is used to mint the renBTC in the Ethereum blockchain.

In the redeeming process, a user uses the application to burn the renBTC. The user specifies the Bitcoin address for returning the bitcoins. The current RenVM shard (almost certainly a different one) produces a signature that transfers the bitcoins to the user specified address.

Analysis As mentioned before the protocol doesn't need any loss of privacy for the user. That is because the user has to use a community of nodes that (supposedly) are not under the control of a single entity. So by current regulations the DarkNodes do not have to do any check on users that can still be anonymous. On the other hand, the protocol uses a new multiparty computation algorithm that isn't currently highly tested or peer reviewed[18]. This of course makes the protocol riskier than other protocols that do not require this level of cryptography. Yet the protocol is as decentralized as the number of DarkNodes assuming that the different DarkNodes are no controlled by the same instance or entity.

[18] I checked in famous repository of paper such as DBLP, Scopus, and IEEE.

7.3.3 sBTC

While in the previous two protocols every tokenized bitcoin was backed by a bitcoin, this protocol created by Synthetix has a different nature. Indeed the "synthetix" bitcoin is a financial instrument tracking the performance of bitcoins. Using glossary from the finance world, sBTC is a *derivative instrument* [Tea22]. By owning sBTC a user does not hold the underlying asset (bitcoins in this case, but the protocol has many other "Synths" for other assets such as Ethereum or Gold.

The Protocol sBTC tokens are issued ('minted') by those users who hold the Synthetix native cryptocurrency, SNX. In practice, the user deposits ('stakes') SNX in a decentralized application which acts as an escrow for the sBTC. The SNX therefore acts as collateral for sBTC: the proportion of SNX:sBTC is generally 4:1 for L1 coins and 5:1 for L2 coins and tokens[19]: that means a user has to stake as many SNX as the value of four bitcoins to mint one sBTC. The sBTC is then traded on the Ethereum blockchain or Ethereum-like L2 protocols.

Note that the proportion is a function of the SNX value. Therefore if the financial value of bitcoins decreases with respect to the SNX value, then part of the locked SNXs will be returned to the staker. The value of all synthetic assets in the Synthetix system is determined by different oracles' sources. This way the system gets an aggregate value for each asset. Currently, the price feeds are supplied by Chainlink.

Analysis The method can be considered decentralized since the system uses a decentralized application to mint and burn Synthetix tokens. The decentralized application is deployed on permissionless blockchains like Ethereum so there is no problem of inherent centralization of operators. Also since the price feeds are supplied by the most decentralized network of oracles managed by Chainlink, the risk of a price manipulation is fairly low.

The risks involved are related to the smart contracts themselves. The smart contracts are written in Solidity and are fully open source, but there can be always a risk of bugs. Other risks in such a system are related to general financial issues in the architecture of derivatives which are not part of this thesis.

7.4 Stateless VS Stateful SPVs

The previous methods for tokenization do rely on some sort of centralization structure either in the form of committee or traditional central party). To solve this issue, In Chapter 12 I will present a protocol that achieves a tokenization exchange without central parties. To achieve that goal, I will use *Stateless SPVs* (Simple Payment Verifications) In order to understand what they are, I provide review on stateful and stateless SPV related to Bitcoin.

7.4.1 Stateful SPV

The first light client has been envisioned by Satoshi Nakamoto in the Bitcoin Whitepaper itself [Nak08]:

> A user only needs to keep a copy of the block headers of the longest proof-of-work chain, which he can get by querying network nodes until he's convinced he has the longest chain, and obtain the Merkle branch linking the transaction to the block it's timestamped in.

The method has originally been called Simple Payment Verification (SPV), but it is commonly referred to "light clients". In practice, a light client let user discern whether a consensus has been reached about a transaction or a set of transaction without making the validation or participating actively into the consensus mechanism. Light-clients are generally off-chain software which connect to nodes.

More complex and performant methods have been created throughout the years. For example, a solution based on Non Interactive Proofs of Proof of Work (NiPoPoW) [KLS16] extracts a "deeper" chain which links together blocks whose nonce would be valid for higher difficulties. More formally, recall PoW based blockchains (and Bitcoin in particular) have a difficulty parameter d that is representable as the minimum number of left-most zeroes a block-hash needs to have in order to be considered valid

[19] These proportions are not fixed, they are decided in a DAO, see footnote 17

by the nodes of the chain. Nevertheless, a nonce can create a hash representing a number of zeros $d' > d$. The solution proposed by the authors of [KLS16] leverage the higher difficulty of those blocks to link them together. Although interesting and very performant in principle, the method requires a modification of the Bitcoin protocol and can not be instantiated now. Another method is Flyclient [BKLZ20] which leverages probabilistic sampling and Merkle Mountain Range (MMR) [TD18] commitment of all previous blocks. Flyclient needs some modification to the Bitcoin protocol too.

By definition, any light client assumes that the chain with the most PoW solutions is the one that follows the rules of the network and will eventually be accepted by the majority of miners. Consequently, a light client has to store at least a specific subset of the headers of the blockchain in order to be able to evaluate the presence of transactions: in this sense, those kind of light clients are considered *stateful*.

This has been a brief introduction. For more detail and a complete systematization of knowledge on the matter see e.g. [CBC22]

7.4.2 Stateless SPV

The second kind of light client is the stateless one. It does not store any headers of the blockchain. These light clients perform checks on the headers provided by the users.

Stateless SPV have been presented by Jason Prestwich as a way to solve the problem of relay-maintenance in the Ethereum blockchain[20]. Stateless SPVs are currently proposed for alternative versions of relays but there is no formal explanation of how them work[21]. Goal of this section is to fill this gap.

For our purposes we can model the typical Bitcoin block as a set of five fields: magic number, block size, block header (or simply "header" from now on), transaction counter and transaction list. As in the light-clients case, we are not interested in the transaction list or the other constants, but only in the block header, since this is what SPVs (both stateful and stateless) are interested in.

The fields of the block header are presented in Table 7.3 and I refer to the header of i-th block $\mathcal{B}_i$ as $\mathcal{H}_i$ and to its fields via dot-specification, e.g. the nonce field of the i-th block header is $\mathcal{H}_i.nonce$. The leader election phase of the Proof of Work uses the header to compute a hash. If this hash is less than a target L, then the block can be included in the blockchain. Note that the lower the target L the lower the probability of finding the right hash. Consequently the lower L, the more *work* (i.e. nonce update and rehash) needed by the miners.

Let h_i be the valid hash of the header $\mathcal{H}_i$ and note that h_i is included in $\mathcal{H}_{i+1}$ as $\mathcal{H}_{i+1}.hashPrevBlock$ and that its numerical representation of h_i has to be lower than L.

Assume a user U broadcasts a transaction tx on the Bitcoin blockchain. Slightly abusing language, I identify the transaction with its transaction hash. Assume tx is included in block $\mathcal{B}_N$ among other $k-1$ transactions. Then the vector vec_N containing the transaction and the Merkle tree's branches contains the data needed to verify that transaction tx is included in block $\mathcal{B}_N$ by comparing the Merkle tree hash with $\mathcal{H}_N.hashMerkleRoot$. For example, with reference to figure 7.2, the vector vec_N would be:

$$vec_N = [tx, H_0, H_{23}, H_{4567}] \tag{7.3}$$

Assume a smart contract SC is capable of evaluation hash functions and make integer comparisons. Therefore SC can reliably check if tx has been confirmed in the Bitcoin blockchain via the evaluation of a proof π such that:

$$\pi = [vec_N, \mathcal{H}_N, \mathcal{H}_{N+1}, \ldots, \mathcal{H}_{N+n_{blocks}}] \tag{7.4}$$

where n_{blocks} is the number of blocks needed to be sufficiently secure that π has not been forged. See Section 12.2.2 for a more detailed security proof.

[20] See for example [Eth21] which is currently the most popular relay on the Ethereum network. Despite this, development is stopped at 2017 (last commit) and as of May 19th 2022 the last transaction is from 1325 days ago. The main reason seems to be the fact that maintain a relay is costly and there is no rational incentive to do so.

[21] We checked in famous repository of paper such as DBLP, Scopus, Google Scholar and IEEE. Yet, it is possible to see the original talk by the author at https://youtu.be/njGSFAOz7F8. An informal explanation of how Stateless SPV work and their security is proposed in https://ethresear.ch/t/stateless-spv-proofs-and-economic-security/5451

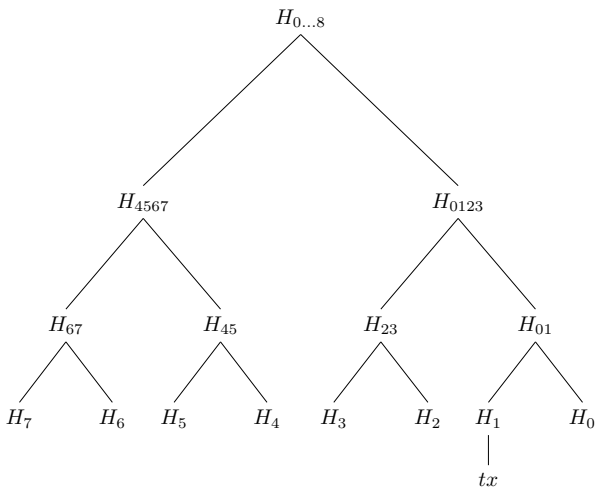


Figure 7.2: Particular Merkle tree structure of a block with transaction tx. In this case the Merkle leafs needed for the proof π are (H_0, H_{23}, H_{4567})

Field	Description
Version	The version of the block.
$hashPrevBlock$	The hash of the previous block in the chain.
$hashMerkleRoot$	The hash of the Merkle root of the transactions.
Time	The time the block was created.
nBits	The target of the block (compressed form).
nonce	The nonce of the block.

Table 7.3: Block header fields

7.5 Interoperability via Peer to Peer Exchanges

Peer to peer (P2P) exchanges are currently the most decentralized way to operate an exchange between two blockchain's and the most ancient way to do that. The inventor of P2P exchanges between blockchains is considered to be Tier Nolan. In his post on the BitcoinTalk forum he describes a way to operate an exchange between the Bitcoin blockchain and another bitcoin-like blockchain, specifically Litecoin [Nol13]. His method uses locks based on hash functions and time to solve the fair-exchange problem that emerges in cases like this one, hence the name Hash Time-Lock Contract (HTLC). For an introduction to the fair exchange problem see Section 7.1.1 and see Section 5.2 to understand how atomic swaps work.

As the name suggests, peer-to-peer exchanges, also called atomic swap, do not need any trusted third-party. More formally:

Definition 7.5.1 (Atomic Swap) *Assume parties A and B want to exchange a certain number of tokens T_a and T_b running on blockchains BC_a and BC_b respectively. Also assume the parties measure their wealth in terms of token T_a and that $\alpha = \frac{T_a}{T_b}$ is the rate of exchange of the tokens. Then a* cross-chain atomic swap *is an exchange of n tokens T_a for $n\alpha$ tokens T_b such that*

- *If the protocol is* successful *then A owns $n\alpha$ tokens T_b and B owns n tokens T_a*
- *If the protocol is* non-successful *then A owns n tokens T_a and B owns $n\alpha$ tokens T_b, i.e. the state of the participants' wealth is unchanged*

Atomic swaps are also known as *fair exchange* protocols in the context of distributed systems, see Section 7.1.1. On the one hand the lack of a third party is good since users do not need to trust anybody, but on the other this comes at the cost of liquidity. In fact since it is difficult to advertise one's intention to operate an exchange and therefore the peer-to-peer exchanges are not the most technically-efficient way to exchange tokens.

Lack of liquidity is due to the fact that atomic swaps are currently difficult to perform practically. The difficulty arises from the fact that to operate this exchange the users have to rely on time locks (see Sections 5.2 and 11.1): practically users have to wait for locks to expire to be able to spend transactions. One of the drawback of waiting in a highly volatile environment is that currency rates will vary a lot. On average users value the risk of loosing financially higher than the risk of being censored or lose funds stolen by the services described in the previous section.

We can be sure of that since we can see what actually happens in the real world, since Bitcoin and Ethereum are permissionless blockchains: users do prefer to use centralized exchange instead of atomic swaps as currently implemented. While we generally do not have raw data on the volume of exchanges in centralized exchanges (we only have publicly accessible data), we can see on chain that there are relatively few atomic swaps.

The fact that it is possible to say that is a hint of the fact that that currently the method is not very private. In fact because of its construction atomic swap based on hash and time locks leaves a trace on both blockchains, specifically the hash used to lock a transaction which is equal to both the transaction on the first blockchain and the transaction to the second blockchain. This hash operates as an index and therefore it is possible to link the two transactions. We will see in the course of this thesis that this is a major flaw from the point of view of the users (Section 5.3).

Of course there are other ways to exchange in a P2P way. The majority of these methods though are not secure in the cryptographic sense. For example any over-the-counter (OTC) transaction is inherently peer to peer, but it is based on trust instead of security. In the course of this thesis I will not deal with those kind of exchanges since they are not based on cryptographic methods but from only human interaction.

7.6 Other problems

In this section I briefly introduce the other problems I mentioned before, namely scalability and centralization.

These problems must be taken into account when outlining new protocols: the worst thing you can do is to create an interoperability protocol that increases the overall centralization of the ecosystem. Conversely, these problems can be mitigated to some extent with an interoperable protocol. as those below will show.

7.6.1 Scalability problem

Due to their consensus algorithm both PoW (2.2.1) and PoS (2.2.2) based blockchains currently rely on the fact that all nodes with writing capability (called “miners” in PoW and “validators” in PoS) and all nodes with reading capabilities validate all the operations on the chain. Consequently, each new node only increases the burden on miners and validators and consequently the whole chain. In technical terms, both PoW and PoS blockchains do not scale in productivity when nodes’ number increases, meaning that block production rate (i.e. throughput) stays constant regardless to the number of nodes. The scalability problem has ripple effects that aggravate other problems. In fact, proposals on both Ethereum and Bitcoin focus on the creation of channels and sidechains (or “parallel blockchains”). The general idea is to offload part of the computations of the parent chain (Bitcoin or Ethereum in this case) to the channel on the sidechain. In order to accomplish that, the parent and the sidechain need to communicate: thus the blockchain-layer interoperability problem.

But sidechains are not the only way to solve the scalability problem and increase the throughput of a blockchain. Another way is to require powerful nodes and let block-size increase. That venue has a ripple effect towards centralization risk, of which I talk in the next section (7.6.2).

7.6.2 Centralization problem

The tradeoff between scalability and (de)centralization is so famous it deserved its own name: the *Scalability trilemma*.

The acclaimed inventor of this model is Vitalik Buterin. The trilemma states that given a blockchain project project, it can achieve only two of the three vertexes of the triangle. So either a decentralized system is scalable or it is secure; it can not be both. Similarly, a scalable project is secure (and

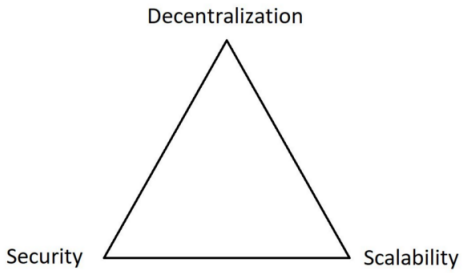


Figure 7.3: The Scalability trilemma

therefore centralized) or it is decentralized. Finally, a secure project can be decentralized or scalable, but not both. Since the not prescinding property of a system is its security, we see why the scalability trilemma is actually a scalability-decentralization tradeoff.

Centralization arises for different reasons. I use the work by [GSM+20] and [SBFG20] to explain the centralization problem with respect to the blockchain use cases. The first step to tackle this problem is splitting the blockchain environment into layers, as it can be seen in Figure 7.4.

In [SBFG20] the author see a blockchain as a stack of six layers. The layers are the following (from the higher to the lower/deep level):

- *Application layer*: the layer that contains the application logic. In practice this layer refers to the wallets used to interact with the blockchain, the exchanges (both centralized and decentralized) and the development tools and processes used for the reference clients[22].

- *Contract layer*: this layer process transactions in the network. Developer can express the logic of payments and other transactions in smart contracts. Note that for the sake of this thesis I will consider smart contracts as whatever set of logic needed to create access-control to funds or data-sharing mechanisms. In this sense, transaction in Bitcoin are handled by (verification based) smart contracts written in the non-Turing complete scripting language Script and Ethereum smart contract are programs written in a (nearly) Turing complete language;

- *Incentive layer*: this layer determines the incentive scheme for the blockchain. Among other things, this layer is responsible for the assurance of security of the blockchain, since without incentive no possible assumption on the blockchain liveness or security is possible (although incentives are necessary but not sufficient, see [ES18, GKL15]);

- *Consensus layer*: this is the layer that implements the consensus algorithm. The consensus layer assures that the network reaches a consensus on which new data that has to be stored after assuring that the data is valid;

- *Network layer*: the layer that let users communicate. This is the layer that is responsible for the message passing between nodes, i.e. the layer that makes the consensus algorithm possible. For further details see Section 2.1;

- *Data layer*: this part concerns all the data structures and algorithms that are used to store the data and the cryptographic primitives need to secure the integrity and the availability data. It is assumed that the network participants adhere to the data layer specifications to participate in the network. this is also the base of the application layer.

On the other hand in [GSM+20] the authors use a stack of three layer, namely the *Storage layer*, the *Network layer* and the *Governance layer*. The storage layer of [GSM+20] is the data layer of [SBFG20], while the network layers refer to the same layer in both papers. The governance layer is the layer that implements both the consensus layer and the incentive layer of [SBFG20].

[22]Since it is difficult for a specification to be universally executed by all developers (see RFC 4181 [Hea05] and RFC 2119 [Bra97] in this regard), it is necessary to have a client that is considered to be "right" for the operation of the blockchain.

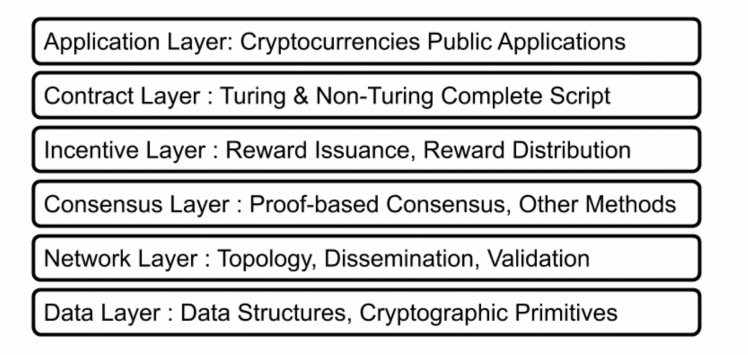


Figure 7.4: The layers of the blockchain, Fig.2 of [SBFG20]

In particular, when I present the interoperability methods of the blockchain, I will be careful not to introduce centralization in the application layer. Specifically since the goal is to diminish the need for centralized exchanges and their custodial based systems, I need to check that I do not introduce centralization in other parts of the application layer.

Dividing the blockchain into layers is very useful to create measures of centralization. This gives the possibility to measure quantitatively the degree of centralization of a blockchain. Since in this thesis I are dealing with interoperability and not centralization I will not go into detail about all the measures. The important thing to keep in mind when dealing with the topic of "centralization" is that it can be measured.

Chapter 8

Common Attacks

A multichain paradigm is of course more complex and therefore less secure (in principle) than a single-chain system. In particular, many attacks applicable on one blockchain can have consequences into the other. Goal of this chapter is to highlight the possible problems happening when creating protocols for blockchains' interaction.

8.1 Fork based

The most critical attacks that can happen are those that have forks as consequences. Forks (see Section 2.1.3) are critical for parties executing an atomic swap protocol since they may lead to inconsistencies regarding the value of the portfolios of the participants. We give an example.

Assume a swap between two clients A and B, and for simplicity we call BC_A and BC_B the two blockchains where A and B initially have funds. In case of a successful swap, A and B have the funds in the blockchains BC_B and BC_A, respectively. We now assume a reorganization in BC_B, such that the swap transaction between B and A is removed from the transactions included in the blockchain. At this point A no longer has funds in BC_A, and at this point it receives no funds in BC_B. It follows that A has lost its funds, getting nothing in return: such a situation cannot be accepted but it is (currently) part of the risks of an interoperable system among multiple chains. Because the problem is inherent in the system and non-solvable, it is important to make atomic swaps only between blockchains in which the probability of block reorganization is very low, i.e., which have a very high *difficulty*, as seen in Section 2.2.

We analyze three attacks, namely the 51% Attack (Section 8.1.1), the selfish mining strategy (Section 8.1.3) and the Eclipse attack (Section 8.1.4) since these attacks can lead to a fork in a blockchain.

8.1.1 51% Attack

While this is the most popular and cited attack, it is also the least technically definable. In fact, a 51% attack is the name given to the potential issue that present itself when a group of miners or validators colludes pooling the resources needed to update the blockchain and reach consensus (be it computational power in the case of PoW or the amount of native coins in stake in the case of PoS). I said "potential" since assuming that the colluding group is composed of benevolent actors, nothing nefarious happens.

It is not possible to analyze this attack directly, although it is undeniable that this situation gives rise to attacks with important consequences. In the following we analyze these attacks.

8.1.2 Double Spending

Unlike the previous attack, this one is well-defined. A double spending attack involves sendings the same funds to two different recipients. The attacker's hope is that both recipients will be temporarily satisfied and act to benefit the attacker. When the blockchain reaches a consensus, only one of the two transactions will pass. At this point, one of the two recipients finds himself without funds, although it has now acted honestly.

q	$z=0$	$z=1$	$z=2$	$z=3$	$z=4$	$z=5$	$z=6$
5	100	10.12	1.26	0.16	0.02	0.00	0.00
15	100	30.97	11.50	4.42	1.73	0.68	0.27
25	100	52.23	31.54	19.61	12.35	7.84	4.99
35	100	73.06	58.88	48.45	40.25	33.64	28.22
45	100	91.98	87.77	84.44	81.56	78.98	76.61

Table 8.1: Table representing the probability (in percentage) for a successful double spend, where q is the share (in percentage) owned by the colluding miners of the total computation power, and z is the number of block confirmations. The values of this table have been computed adapting the code in the Bitcoin whitepaper [Nak08]. See Listing 8.1.

An example may clarify the situation. Assume a malevolent participant M and an honest seller S. Assume M has two addresses $addr_{M1}$ with funds to spend inside it, and $addr_{M2}$ initially empty. Similarly, S has an address $addr_S$ where it receives payments for the items it sells. A double spending by M follows the following steps:

1. M buys an item from S and sends funds from $addr_{M1}$ to $addr_S$
2. Soon after, M sends *the same funds* from $addr_{M1}$ to $addr_{M2}$ (in a UTXO model, that means spending the same coins; in a account model that means reusing the same nonce)
3. M hopes that S sends the object and that the transaction remaining in the blockchain is the one from $addr_{M1}$ to $addr_{M2}$

It is possible for the Step 3 to occur, especially if the objects that S sells are digital and their sending is immediate.

Historically, to mitigate this problem, *confirmations* are expected, e.g. it is good practice in Bitcoin to accept payment after 6 blocks[1].

Note that the specific number 6 is a best practice, assuming the arrival of blocks follows a Poisson distribution with $\lambda = 10$ minutes (an assumption proven right by empirical analysis for blocks in the same group between two difficulty adjustments, see as example [BKKT20]). It's is also interesting to note that the percentage of malicious nodes in the so-called "51% attack" affects this best practice, which unfortunately is a fixed number and not dynamically adjusted for evident coordination reasons.

```c
#include <math.h>
#include <stdio.h>

// function taken from the Bitcoin Whitepaper
double AttackerSuccessProbability(double q, int z)
{
    double p = 1.0 - q;
    double lambda = z * (q / p);
    double sum = 1.0;
    int i, k;
    for (k = 0; k <= z; k++)
    {
        double poisson = exp(-lambda);
        for (i = 1; i <= k; i++)
            poisson *= lambda / i;
        sum -= poisson * (1 - pow(q / p, z - k));
    }
    return sum;
}

void printProbability(double q, int z)
{
    printf("Probability of success for q = %f and z = %d is %.2f\n", q, z,
        AttackerSuccessProbability(q, z)*100);
}

```

[1] At https://howmanyconfs.com/ it is possible to see a table in which for each blockchain (other than Bitcoin) the number of blocks needed to achieve security equivalent to Bitcoin's 6 confirmations is presented

```
26 int main()
27 {
28     for (float q = 0.05; q <= 0.5; q += 0.1)
29         for (int z = 0; z <= 6; z++)
30             printProbability(q, z);
31
32     return 0;
33 }
```

Listing 8.1: Code to obtain results in Table 8.1.

8.1.3 Selfish Mining

As it is widely known, Bitcoin and other proof-of-work blockchain projects rely on the concept of "pools" to diminish the volatility of the block rewards. The bigger the pool, the higher the chance to win. On the other hand, since reward are distributed to pool-members proportionally to the amount of hashrate (proxy for computational power) supplied, the smaller the contribution of each pool-member, the smaller the reward. Note that given a sufficiently long time periods (e.g. months), the expected value of the reward for each member stays the same either in the case it joins a pool or if it "mines solo". What happens is that by joining the pool, the miner gets a share of the reward even if it did not find the nonce (see Section 2.2.1): rewards are more predictable and therefore less volatile (i.e. lower variance), even if the mean reward is the same.

While this reasoning works if *all* miners are honest, the authors in [ES18] describe a strategy that can be used by a pool to obtain more rewards than the pool's fair share, that is, more than its ratio of the total mining power. We present how the Selfish Mining strategy works, so to be able to explain why it has possible consequences in an atomic swap.

Assume a mining pool finds a new block. In this case the poll has an advantage with respect to the rest of the network. According to the protocol, the pool should publish the newly founded block to get a reward. A selfish mining pool, instead, keeps this block private to the pool. Two things can happen at this point. If another (honest) miner finds another block at the same height, then the selfish mining pool looses its advantage. On the other hand if the selfish pool mines a second block, it extends its lead on the honest miners.

If the honest miners succeed in finding a block on the public branch and the selfish mining pool's lead is nullified, then the dishonest pool publishes its private branch (of length 1) creating a block race between with the block of the honest miner. In this case, the selfish miners unanimously adopt and try to extend the previously private branch, while the honest miners will choose to mine on either branch, depending on the propagation of the block.

Now three things can happen in this first case:

1.1 The selfish pool mines a new block building on their previous block: the selfish pool get twice the reward

1.2 A honest miner finds a new block building over the selfish miner pool's previous block: the selfish mining pool still gets the reward for the previous block

1.3 A honest miner finds a new block building over the honest miner pool's previous block: the selfish mining pool does not get any reward

The second scenario is easier to analyze, since the selfish miner pool gets twice the reward, but with less risk since by keeping the first block private, honest miners are denied the chance to compete.

The authors prove that the Bitcoin protocol is not incentive-compatible, meaning that this attacks leads on the long run to a revenue which is higher than the fair share of the dishonest miners (i.e. a revenue proportional to the total computation power). The consequences highlighted by the authors of the paper deal with the possible centralization of consensus due to the fact that more miners are rationally incentivized to join the dishonest pool. On the other hand, the most damaging consequence as far as interoperability is concerned is the creation of two parallel chains (forks) which, even if of a few blocks, risk creating damage to the participants A and B mentioned above, since it is possible that e.g. the transaction of A to B is not recorded after the reorg, and that B therefore runs out of its share of funds

8.1.4 Eclipse attack

The *eclipse attack* allows an adversary to leverage a Sybil attack (i.e. pretending to be different nodes while controlling all of them) to monopolize all connections to and from a victim blockchain node [HKZG15]. The adversary exploits the victim for attacks on the blockchain consensus system, including N-confirmation double spending and selfish mining (Section 8.1.3).

The authors cleverly engineered the attack by forming unsolicited incoming connections to the victim from a set of attacker-controlled nodes sending bogus network information. Specifically, the authors in [HKZG15] consider two attack types. The first is an attack at the infrastructure level: the adversary is considered to be an entity with ISP, company, or nation-state powers. Specifically the adversary holds several contiguous IP address blocks and the goal is to subvert a blockchain network by attacking its peer-to-peer communications. The second type of attack is a botnet attack, launched by bots with addresses in diverse IP address ranges.

The consequences of an eclipse attack are plenty, ranging from *ad hoc* block races, i.e. a race between newly-minted blocks at the same height, selfish mining and N-confirmations double spend.

This last consequence is the most critical in an atomic swap. Assume an atomic swap between parties A and B, with A and adversary able to eclipse B. Transactions need confirmation in an atomic swap as in any exchange (see Part II to see how this is accomplished). Then A broadcasts its transaction to B, who incorporate it into its (obsolete) view of the blockchain. B sends its part of the transaction (typically the funds it "owes" to A) assuming its own node is tracking the "right" blockchain. When A receives B's payment, it stops eclipsing B's node. The eclipsed participant's blockchain is orphaned, and the attacker A obtains the funds without paying.

The attack is possible and disruptive, but nodes can mitigate the probability of its success by taking some precautions. For example, the authors of the paper on the attack recommend to disable incoming connections, while allowing 'specific' outgoing connections to well-connected peers or known miners (i.e., use whitelists). Some of those counter measures have been incorporated into major blockchain software, but it's possible that some less maintained project does not have yet by default, so, as usual, it is better for users to run highly reviewed code which has regular security updates.

8.2 Privacy threat

Even if touted as private and anonymous, cryptocurrencies are neither one nor the other (including so called privacy coins, see Address Clustering in Table 8.2). The privacy problem is the one that is the most important in the blockchain use cases. In the following I give a brief overview of the issue. For a survey on the major privacy threats and mitigations of the blockchains, see [FHZ+19]. I also include in Part II how the protocol I present do mitigate some of these threats.

It is well know that Bitcoin addresses can be clustered [MPJ+13], and the same holds for Ethereum addresses [Vic20]. Since the two blockchains cover both the UTXO-based and Account-based models, we can say that at least clustering can be done on the vast majority of blockchain. See Table 8.2 for a list of the privacy threats and their explanation. One way to mitigate the privacy threat is the use of a mixer [FHZ+19]: As part of our on-going effort to create peer-to-peer and privacy preserving atomic swaps, I present a decentralized mixer called DMix [BS20a] in Chapter 9.

Threat	**Explanation**
Address Clustering	A way to group addresses as owned by the same identity. Address clustering is based on heuristics. Even if clustering can not by itself uncover the identity of the entity, it is a dangerous first step in that direction. See e.g [MPJ+13, MN22] for Bitcoin and e.g [Vic20] for Ethereum. In the past it has been possible to use address clustering even in ring-signature based Monero [MSH+18]: even if the bug that let the traceability possible has been fixed, the highly-novel and untested cryptographic constructs implemented in Monero make it subjected to possible new attack vectors. Finally, note that nearly all Zcash transactions are not shielded (See Table 1 of [KYMM18]), and therefore they suffer of the same problems Bitcoin transaction suffer.
Network Analysis	Since the blockchain is based on a P2P network architecture, nodes will leak their IP address when broadcasting transactions. In [KKM14] the authors developed heuristics for identifying ownership relationships between Bitcoin addresses and IP addresses.
Transaction Fingerprinting	In [AKR+13] the authors prove that the user-profiles of nearly 40% can be recovered even when users adopt the popular privacy measures recommended. They achieve it by measuring many properties of the network, such as Random time-interval (RTI), hour of day (HOD), time of hour (TOH), time of day (TOD), coin flow (CF) and input/output balance (IOB)

Table 8.2: Possible privacy threats

Part II

Blockchain to Blockchain Interoperability Methods

Chapter 9

Decentralized Mixers

I introduced the privacy problems in Section 8.2. In particular, we saw how address clustering can threat the privacy of the users of the Bitcoin blockchain. One of the reasons why it is possible to cluster the addresses is the possibility of following the funds from one transaction to the other. In other words, the addresses are *linkable*. To mitigate that, in this chapter I present DMix [BS20a], a decentralized protocol that acts as a decentralized mixer for the Bitcoin blockchain. I called it DMix. DMix provides unlinkability and privacy leveraging threshold signatures. The description of the implementation is done assuming the Schnorr signature scheme (see Schnorr Threshold signature construction in Section 4.4.3), but it is possible to make it also with the threshold-version of the ECDSA. In fact you can find a demo of how this protocol works which uses a threshold ECDSA protocol at my GitHub page[1].

In particular, throughout the description of the protocol I assume the parties will use the MuSig2 [NRS] threshold scheme. DMix is a stepping stone for the MP-HTLC protocol [BS22b] I present in Chapter 10.

This chapter is so composed. In Section 9.1 I give a brief overview of the protocol used to mitigate privacy threats and address clustering in particular. In Section 9.3 I explain how Partially Signed Bitcoin Transactions work: I use them to create the transaction in Phase 3. The whole protocol is described in Section 9.4. In Section 9.5 I give a walk-through of how DMix protocol work using three fictitious parties Alice, Bob and Carol. Finally in Section 9.6 I analyze the protocol.

9.1 History

There are two current methods which address the address clustering issue on Bitcoin: mixers and CoinJoin (see Table 8.2). In the following I proceed to analyze the proposals to address the problem.

9.1.1 CoinShuffle

In CoinShuffle [RMK14], the authors propose a method to mix coins in an anonymous and decentralized manner. The authors assume that every participant holds the same amount of coins at some Bitcoin address and that this address will be one of the input addresses in the mixing transaction. Furthermore every message from this participant is signed with the private key associated with the public key/address.

The protocol is split into three phases. In the first one, participants create their new addresses, then they shuffle and broadcast them to write the transaction and in the last phase they verify the transactions. Particular care must be taken for the second phase: the shuffling.

If parties A,B and C want to perform a CoinShuffle, they have to decide an order of shuffling. Without loss of generality I can assume the order is lexicographical, and therefore it will be A,B and C. To perform the shuffle A acts as a leader and she will encrypt its new address with C's public key first and then with B's one, using the non commutativity of encryption. When B receives the double encrypted message from A, he can decrypt it and see the encrypted (with C's public key) message. B

[1] https://github.com/disnocen/dmix2

will encrypt his new address with C's key, then B shuffles his message and the A's one and finally he sends both messages to C.

C can decrypt both messages and see the addresses. He can not say which new address belongs to A or B. He creates a transaction with those new addresses and broadcast it to the blockchain. This method presents a possible point of failure: if C does not broadcast the transaction, the whole process will stop.

9.1.2 TumbleBit

Differently from CoinShuffle, TumbleBit is a centralized mixer [HAB+17]. The authors propose a method to interactively exchange bitcoins between two parties in a way which promote unlinkability by creating two channels; one between the first user A and the Tumble T and the other between the other user B and the Tumble.

In the Payment phase, A pays T which then pays B. This can be scaled to hundreds of users, therefore providing anonymity thanks to a big anonymity set and the many possible choices of A and B.

The system provides unlinkability, balance [2] and security against DDoS and Sybil attacks. On the other hand, the system accomplish those goals being a trusted third party which could abuse its position (e.g. he can delay payments or refuse to establish channels with some user which won't be able to swap funds). Furthermore, to be able to provide a multitude of users with the required liquidity, the system must hold a great amount of bitcoins. This is an incentive for the creation of centralized payment hubs, similar to centralized exchanges.

9.1.3 CoinJoin

CoinJoin, originally proposed in [Max13], is a particular transaction which aggregates inputs of different people to jointly pay one or more parties. The goal is to build transactions in a way that tries to invalidate naive taint tracking. It does not need any change to the Bitcoin protocol and it is relatively easy to perform, but it needs interactive coordination between parties. In particular, CoinJoins try to invalidate the heuristic that can be defined as follows [MPJ+13]:

> If two (or more) addresses are inputs to the same transaction, they are controlled by the same user.

On the one hand, CoinJoins invalidate the efficacy of the heuristic [MO15], on the other hand users risk to come into contact with *dirty* coins, such as bitcoins suspected to come from a theft or illegal black market: the innocent person could be considered to be the author of such theft if the heuristic is applied and considered as valid. At this point he must prove that he made a CoinJoin, losing his privacy [TMEJ19]. Another analysis by Maurer et al. [MNF17] suggests that normal CoinJoins do not provide unlinkability. To obtain unlinkability it is necessary to consider other mechanism, such as mixers.

9.2 Conditional Transactions

As mentioned in Section 3.1 every Bitcoin transaction has one of the standard output scripts where there are encoded the conditions for redeeming it. *Non-standard* output scripts, are generally ignored by the majority of nodes for security reasons[3].

Thanks to BIP 13 [And11a], any output script can be converted to P2SH as seen in Section 3.1.2. This allows users to use non-standard output scripts by publishing it in the form of a P2SH transaction. In particular, the output script we use in this protocol uses an opcode described in BIP 65 which defines the `CHECKLOCKTIMEVERIFY` opcode. This opcode allows a transaction output to be made unspendable until some point in the future. See [Tod14] for details and examples.

In DMix, we use a particular form of escrow, described in [Tod14]. Assume that there are two participants in an instance of DMix, *Alice* and *Bob*, that want to send money to the aggregate address

[2] Balance means that there is no possibility of inflation or destruction of money even if parties colludes

[3] It is possible to execute DDoS attacks by placing output scripts that are too complicated to verify [BMS19].

created with the procedure described in the previous subsection. Let `HpkA`=pk_A be the hash of the public key of *Alice* and `HpkAB`=pk_{AB} be the hash of the aggregate public key. Then the script[4]:

```
IF <nBlock> CHECKLOCKTIMEVERIFY DROP <HpkA> CHECKSIGVERIFY 1
ELSE 1 ENDIF <HpkAB> 1 CHECKMUSIG
```

gives *Alice* and *Bob* a chance to spend the money signing the transaction with their common private key before block `nBlock`. If the two participants do not reach an agreement before that block, then *Alice* can still spend her money operating only with her private key. Therefore *Alice* is protected in case *Bob* is malicious, and the exchange is atomic. We call this output script *Script*1.

To publish a P2SH Bitcoin transaction based on *Script*1, *Alice* create the hash of *Script*1, `Hs1`, and then publish the following outputScript[5]:

```
HASH160 <Hs1> EQUAL
```

We call this transaction $tx1$. Redeeming transaction $tx1$ requires a transaction $tx2$ with four inputs: the output script of the hash in transaction $tx1$ (*Script*1 in our example), the relative inputScript (the inputScript which would evaluate to `true` once concatenated with *Script*1 in our example), the set of output addresses of $tx2$ and the respective output amounts.

The inputScript for $tx2$ in the case of an outputScript analogous to *Script*1 is of the form

$$(\mathbf{sig}(H(tx2), sk_A), pk_A)$$

assuming Alice alone signs $tx2$ and therefore redeems transaction $tx1$, or $(\mathbf{sig}(H(tx2), sk_{AB}), pk_{AB})$ in case both *Alice* and *Bob* sign this transaction. Function $\mathbf{sig}(H(m), sk)$ produce a signature of message m using the key sk.

9.3 Partially Signed Bitcoin Transactions

Some times multiple parties need to cooperate to produce a transaction. Examples include multisignature setups, transferring funds form or to cold or hardware wallets, and the CoinJoin transactions mentioned before. In these cases, one party A may need to exchange an unsigned or partially-signed transaction with another party B so that A and B together can send these funds to a third party C. Originally this process was wallet-implementation dependent, making it hard for people who use different wallet softwares to exchange these partially signed transactions. This problem has been solved with BIP174 [Cho17] and BIP370 [Cho21]. Those BIPs create an interchange format for Bitcoin transactions called Partially Signed Bitcoin Transaction (PSBT).

To give an example of how PSBT work, I describe the process in the case mentioned before, i.e. where a party A and a party B need to send a payment to C using inputs from both A and B. The construction goes through the following steps[6]:

1. Without loss of generality, A proposes a particular transaction to be created by building a PSBT ptx that contains certain inputs and outputs; no additional metadata is needed. A sends ptx to B

2. B adds information about the UTXOs being spent by the transaction to ptx. In particular B adds its inputs

3. Signers A and B inspect the PSBT transaction ptx and its metadata to decide whether they agree with the transaction.

4. If they agree, they produce a partial signature for the inputs for which they have relevant keys. Without loss of generality A sends the current state of ptx to B

5. B runs an extractor protocol to produce a valid Bitcoin transaction from ptx if all inputs are finalized.

[4] We used the non existent opcode `CHECKMUSIG` to simulate the opcode of MuSig because there is not any accepted specific at the time of this writing.

[5] See BIP16 at https://github.com/bitcoin/bips/blob/master/bip-0016.mediawiki.

[6] See https://github.com/bitcoin/bitcoin/blob/master/doc/psbt.md#psbt-in-general for details

In practice PSBTs let Bitcoin users aggregate inputs and outputs in a single transaction. This helps them reduce fees by requiring them to broadcast only one transaction to the chain instead of many. I take advantage of this fact during Phase 3 to make only one signature.

PSBT are already working on both Bitcoin, Bitcoin Cash and its subsequent forks[7]. A version of PSBT called Partially Created Zcash Transactions (PCZT) are currently in the process of being finalized[8].

9.4 Three phases mixer

In DMix, I assume n distinct participants $\{P_1, \ldots, P_n\}$ need to mix their coins in the Bitcoin blockchain. The protocol requires a distinct transaction from each participant toward an aggregated address, DM, using the MuSig2 key generation algorithm created by the participant themselves; I call this kind of transactions `inDMix` transactions. Then, to redeem the coins, the participants need to create an aggregate signature of exactly one transaction that starts from DM and goes to multiple new addresses belonging to the participants; I call this transaction `outDMix`. Figure 9.1b illustrate both kind of transactions. No intermediary is needed to perform all of these steps.

Our protocol is divided into three steps which can be referred to as:

1. Information and Public Keys exchange
2. `inDmix` Transactions
3. `outDmix` Transaction

In the following subsections I present the working of a every phase in detail. I put a concrete example in Section 9.5.

9.4.1 Phase 1: Information and Public Keys exchange

I assume the participants use a secure channel to communicate between each other. To setup a DMix exchange, participants need to at least agree on five inputs in order:

1. **Total Fees**: In the third phase of the protocol, participants send a transaction from the aggregated DMix address DM, so they need to agree on the total fee `totalFee` which they intend to give to the miners beforehand. That way they can independently create the `outDMix` transaction. `totalFee` has to be divided between the participants, e.g. equally among them. Given participants $P_1, P_2, \ldots, P_n$ which have to pay $\texttt{fee}_1, \texttt{fee}_2, \ldots, \texttt{fee}_N$ fees respectively, then $\sum_{i=1}^{N} \texttt{fee}_i = \texttt{totalFee}$.

2. **Timeout**: Users decide the block number (either relative or absolute) for the `nLocktime` field of the `inDMix` transaction. This field is used by the `OP_CHECKLOCKTIMEVERIFY` opcode and it acts as a timer: after that period, if the protocol aborted for some reason, users can redeem their coins (see Section 5.2.2). Thanks to this, the DMix protocol ensures the atomicity.

3. `amountOut`: Users have to establish a common value for each output amounts of the `outDMix` transaction. I call this parameter `amountOut`. Having a common output amount is crucial to preserve the unlinkability requirement in the decentralized mixer: different transaction output amounts could lead to the linking of input and output amounts and lose all the privacy property of a privacy preserving method due to the subset sum problem[9]. This problem undermines the privacy of some methods such as CoinJoin [MNF17], and to avoid this problem we require user to agree on a common output and decide their input based on this parameter.

4. $\texttt{amountIn}_i$: Each participant has to declare the amount he wants to send to DM. Differently from CoinShuffle, users can put different amounts of coins in DMix. Declaring the input amount in this phase lets every other participant to independently build the `outDMix` transaction making DMix a leaderless mixer protocol and consequently preventing a central point of failure. The

[7] https://docs.bitcoincashnode.org/doc/release-notes/release-notes-0.20.6/
[8] https://github.com/zcash/zcash/issues/2542
[9] See for example the Wikipedia article at https://en.wikipedia.org/wiki/Subset_sum

only constraint is that $\texttt{amountIn}_i - \texttt{fee}_i$ must be an integer multiple of `amountOut` for each participant[10].

5. **DMix Address Creation**: Participants in DMix create the aggregate address from the aggregated public key generated by the MuSig signature scheme. In particular, each participant P_i sends in the channel his own public key pk_i . After he collects the other public keys, he creates the aggregate public key PK (see Section 4.4.3). Each participant derives the same public key pk and from that key the participant can create the DMix address DM independently[11].

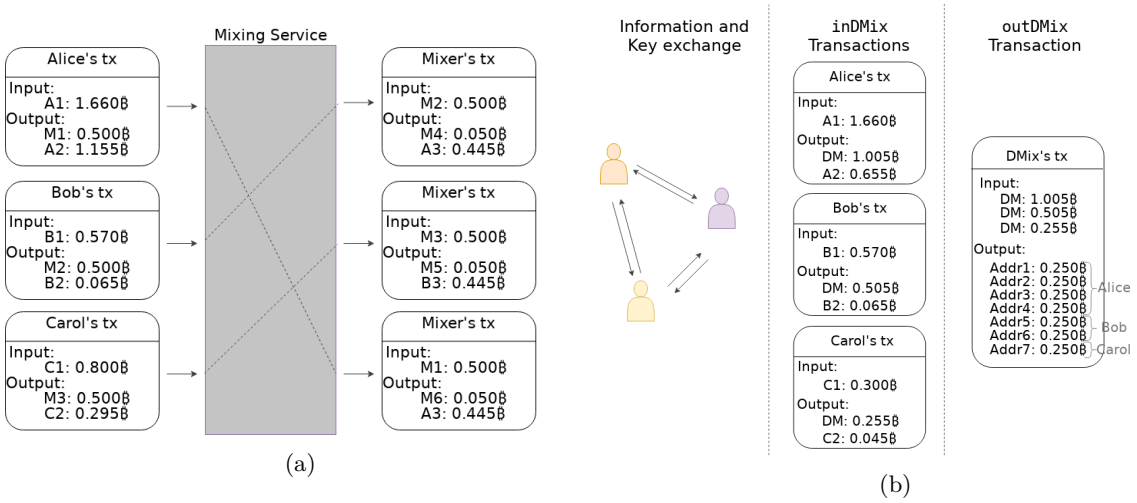


Figure 9.1: A graphical comparison between a centralized mixing service and DMix. (a) A centralized mixer. The path of the mixing is known to the operators of the mixer. (b) DMix example. I marked Alice's, Bob's and Carol's addresses in gray because this information is known only to those who participated in the DMix protocol

9.4.2 Phase 2: `inDmix` Transactions

After discussing the terms of the transaction in the previous step, each participant sends a transaction from his address to the aggregate address DM created in the previous phase. Exactly one output of this transaction must be equal to the `amountIn` previously declared.

These `inDMix` transactions have a construction analogous to transaction $TX1$, as explained in Section 5.2.2. For each participant P_i I call $\texttt{inDMix}_i$ the P2SH transaction he sends to DM. The relative script encoded in the hash of the P2SH transaction sent by P_i will be addressed as $script_i$.

This phase ends when all participants $\{P_1, \ldots, P_n\}$ have sent their `inDMix` transactions and all of them have been included in the blockchain.

9.4.3 Phase 3: `outDmix` Transaction

The goal of all participants in this phase is to redeem their coins. To accomplish this goal they need to spend the coins previously transferred to the DM address. In particular, the participants need to jointly create and sign a single transaction which spends all the inputs from the `inDMix` transactions, moving the coins to many outputs belonging to the participants: this is the `outDMix` transaction. The `outDMix` transaction is analogous to $TX2$ of Section 5.2.2: participants have to create one such transaction and an aggregate signature of it to redeem their coins.

Of the four parameters needed to create this kind of transaction, participants already know the output amounts: these amounts are equal to `amountOut`, decided during the first phase. Furthermore,

[10]Theoretically, a drawback of this method is the creation of many outputs risking the creation of dust coins, i.e. coins whose value is less than transaction fees and therefore they are not spendable. In practice, this is not a real problem because people agree on the fees before this step and decide `amountIn` taking into account future fees.

[11]The address DM is the `base58` of the hash of the public key pk [Nak08]

each participants already knows the aggregated public key: they only need to compute the aggregate signature to complete the input script of the `outDMix` transaction. For these reason these nine steps are needed in this phase. Each participant P_i:

1. sends his newly created addresses $\{Addr_1^i, \ldots, Addr_{m_i}^i\}$, where $m_i = \frac{\texttt{amountIn}_i - \texttt{fee}_i}{\texttt{amountOut}}$
2. sends $script_i$ used in the $\texttt{inDMix}_i$ transaction
3. checks the hashes for each $script_i$ received
4. sorts $\{Addr_1^i, \ldots, Addr_{m_i}^i\}, \forall i = 1 \ldots n$
5. assigns `amountOut` to each output amount and put $script_i$ in the right field
6. create the aggregate signature of the `outDMix` transaction using the MuSig2

These steps give to each participant the ability to independently create and sign the same `outDMix` transaction that will be published to the blockchain.

According to the protocol, every participant tries to publish this transaction. Miners will include only one of these and will treat the others as double-spend attempt, therefore discarding them. The advantage is that this way each participant is not trusting the others anymore than he trusts himself. Therefore, thanks to the properties of the blockchain, they can redeem their money in a leaderless manner.

9.5 Concrete Example

In this example *Alice*, *Bob* and *Carol* are the participants that need to mix their coins and decide to use DMix.

Information Exchange Using a shared secure channel, *Alice*, *Bob* and *Carol* decide the fees they intend to pay to the miners for the `outDMix` transaction, and the timeout. In this case they decide that the fees will be a total of 0.015฿ to be payed equally among them. After this step, they decide for the `amountOut` which in this case is 0.25฿.They see that the bitcoin blockchain has created block number 1000 few minutes ago, and they decide to put the timeout in about three hours. They decide to put an absolute block number: the timeout is decided to be block number 1018. Based on these parameters, Alice decides her `amountIn` = 1.005฿, Bob's `amountIn` = 0.505฿ and Carol's `amountIn` = 0.255฿. Finally Alice, Bob and Carol share their public key with the others and create the address DM following the key generation algorithm of the MuSig2 protocol.

A visual representation of this phase is given in the leftmost column of Figure 9.1b.

`inDmix` Transactions If PK is the aggregated public key and DM is the aggregated address derived from it, I write `HPK`$= H(PK)$ the hash of PK. I call $A1$ the input address of *Alice* in the $\texttt{inDMix}_{Alice}$ transaction and $A2$ her change address. With `HpkA1`$= H(pk_{A1})$ the hash of *Alice*'s public key pk_{A1} relative to the address $A1$.

Alice's script for the P2SH transaction, denoted $script_{Alice}$, is:

```
IF <1018> CHECKLOCKTIMEVERIFY DROP DUP HASH160 HpkA1 EQUALVERIFY ELSE DUP HASH160
↪  HPK EQUALVERIFY ENDIF CHECKSIG
```

and its hash is denoted `HsA`. The $\texttt{inDMix}_{Alice}$ transaction of Alice has the following parameters:

- input Address: $A1$ output Address: DM, $A2$
- input Amount: 1.66฿
  output Amount: 1.005฿, 0.655฿
- inputScript[12]:`true`, outputScript:

  ```
  HASH160 <HsA> EQUAL
  ```

Bob and Carol send analogous transactions, see the middle column of Figure 9.1b.

[12] We put a single `true`, because I assume the user can spend a previously received output

`outDmix` Transactions *Alice*, *Bob* and *Carol* communicate their newly created output addresses and their input scripts to redeem their coins. For easiness of explanations, let's say Alice's new addresses are $\{Addr1, Addr2, Addr3, Addr4\}$, Bob's addresses $\{Addr5, Addr6\}$ and Carol's address is $\{Addr7\}$. Assume these address are already in lexicographical order. *Alice* sends $script_{Alice}$, *Bob* sends $script_{Bob}$ and *Carol* sends $script_{Carol}$; *Alice* checks that $H(script_{Bob})$ =`HsB` from $\texttt{inDMix}_{Bob}$ and that $H(script_{Carol})$ =`HsC` from $\texttt{inDMix}_{Carol}$. Analogous things do both *Bob* and *Carol*.

The input addresses of their `outDMix` transaction are the output addresses of their respective $\texttt{inDMix}_i$, so the redeeming transaction will be:

- input: $DM : 1.005$Ƀ, $DM : 0.505$Ƀ, $DM : 0.255$Ƀ
- output: $Addr1 : 0.25$Ƀ, $Addr2 : 0.25$Ƀ, $Addr3 : 0.25$Ƀ, $Addr4 : 0.25$Ƀ, $Addr5 : 0.25$Ƀ, $Addr6 : 0.25$Ƀ, $Addr7 : 0.25$Ƀ

Alice, *Bob* and *Carol* obtain the same transaction independently and then they sign it, obtaining sig_{Alice}, sig_{Bob} and sig_{Carol}. Finally they collect the other signatures and independently aggregate them and publish their `outDMix` transaction. In the rightmost column of Figure 9.1b I represented this transaction.

9.6 Analysis

9.6.1 Properties satisfaction

I explain here how our protocol satisfies the properties previously listed. Obviously participants do not require any third party to cooperate (e.g. mail communication do not intrinsically need any third party) and there are no mixing fees; the only fees required are those belonging to the miners. Moreover, the protocol has a small overhead with respect to a normal transaction: it only requires one transaction to the DMix address and a transaction from it. For the same reason, the protocol is Efficient if the participants are honest. Assuming Schnorr signatures are deployed, the protocol is compatible with Bitcoin.

Atomicity derives from the the `inDMix` transaction built as in Section 9.2. Similarly, Verifiability derives from the signature choices. In fact, every participant can check that the others are following the protocol: the protocol makes (honest) participants fill the required field of the `outDMix` transaction in a unique way; therefore by applying the verification algorithm of the Schnorr scheme to the collected signatures in the seventh step of the third phase (Section 9.4.3), the participants can be sure that they are creating a valid aggregate signature, i.e. a signature of the right transaction.

The last property is Unlinkability: thanks to the equivalence of all output amounts in the `outDMix` transaction and the fact that the communication between parties is private and/or encrypted, there is no possibility to link input and output in the transaction. Furthermore, this solve the issues of other privacy preserving methods such as CoinJoins (see Section 9.4.1)

9.6.2 Managing attacks

The method lets people exchange funds in an atomic and privacy preserving manner thanks to a private secure channel, aggregate signatures and the `CHECKLOCKTIMEVERIFY` opcode without third parties or any leader election. While I used techniques that prevent passive attacks, it is still possible to perform an internal active attack. In fact, a malicious participant could track all the addresses, linking them to the owners during the third phase and then broadcast the transcription of the communication.

To mitigate this attack, users can open private a channel of communication with other users. In this private channel, parties would privately exchange the addresses. In the common chat room those participants would communicate addresses not belonging to themselves, while being sure that others will communicate their own. I explain here the rationale from a probabilistic point of view.

Let I be the event {There is exactly one malicious user in the group} and set the probability of that event to p, i.e. $P(I) = p$. If there are N participants $\{P_1, \ldots, P_N\}$, this means that I is the union of N different events:

$$I = \{P_1 \text{ is malicious}\} \cup \ldots \cup \{P_N \text{ is malicious}\}$$

Because these are disjoint events, I assumed there is exactly one malicious actor, and given that I have no other information about the participants, then $P(\{P_i \text{ is malicious }\}) = p/N, \forall i$. This is the probability for a member of the DMix group to create a private channel of communication with the malicious actor[13]. This has a good impact on the overall privacy preserving properties.

In fact, assume that the probability $P(I) = 90\%$ and that there are $N = 10$ participants. Therefore the probability of exchanging the address with the malicious actor (and therefore the probability to still be tracked after the use of a private channel and DMix) is 9%. If a participant deems this probability to be too high, he can open multiple private channels with other users and relay other people addresses. For the sake of an example, let's say that P_1 opens three private channels. In this case then, the probability of communicating his own address to the malicious person is

$$\frac{p}{N} \cdot \frac{p}{N-1} \cdot \frac{p}{N-2}$$

which means, following the numeric example above, that P_1 has less than 0.15% of probability of communicating his own address to the malicious person. Furthermore, to be extra cautious, parties can relay addresses using the encryption method used by CoinShuffle [RMK14] which would further increase the uncertainty.

9.7 Conclusions

In this chapter we have seen how the DMix protocol that I created works to mitigate problems arising from transaction linkage. As of now, the protocol is under development. There is a proof of concept as mentioned in the introduction, but it is not user friendly and many steps have to be done manually. We are implementing a version more suitable for the less technical users, while maintaining the properties listed in this chapter. The major challenge so fare is the implementation of a distributed message exchange platform in a threatening and dishonest environment

[13] Actually, if this party is sure not to be malicious himself, the probability is slightly higher and it is equal to $k/(N-1)$ for obvious reasons.

Chapter 10

Hash-based fund-exchanges between multiple blockchains

As we saw in the previous chapter, my work on DMix [BS20a] aims to unlink transactions (and therefore addresses) in the Bitcoin blockchain essentially obfuscating the history of coins. The goal of this chapter is to extend the work done on DMix to make it a full-fledged interoperability protocol. Essentially my goal is to run two instances of DMix on two blockchains BC_A and BC_B such that the receiving addresses on the DMix instance on BC_A are the sending addresses on BC_B. To do that I extend the ideas of HTLCs into a MP-HTLC [BS22b], where MP stands for "multi-party", which also gives the name to the protocol which I present here. By coupling HTLCs with DMix I obfuscate the history on blockchains BC_A and BC_B and mitigate two of the three HTLC problems presented in Section 5.3.2, namely Privacy and Slowness. I do that by increasing the number of participants in a single instance of the protocol and I show how that improves the situation over those problems. I also describe both a use case for interoperability between Bitcoin and an EVM-compatible blockchain and a use case for interoperability between two EVM-compatible blockchains. Finally, a proof of concept implementation with the modifications required in the standard Bitcoin and Ethereum libraries is presented in the related GitHub repository[1].

The section is so composed. In Section 10.2 I give a brief overview of the concept of Multi Party computation (MPC). I need a MPC protocol to create the preimage of the hash for the HTLC in a multi-party way which is also leaderless In Section 10.3 I describe the design of MP-HTLC. Sections 10.4 and 10.5 present two case studies, namely the interoperability between Bitcoin and Ethereum and the interoperability between two EVM-compatible blockchains: in those section the differences of implementation are showed and provided in detail. Finally, Section 10.6 presents an analysis of the protocol under different perspectives: performance, cost and attack-handling.

10.1 Blockchain, Participants and Network Model

We assume the blockchains constitute an asynchronous system lacking of a global clock across chains. For this reason we rely heavily on the result of Asokan [ASW98] on the impossibility of fair exchange without a third parties: its consequence is that there can't be a cross chain communication protocol tolerant against misbehaving nodes with out a trusted third party (see Corollary 1 in [ZAZ+21]). Our goal is to provide a multi-party locking mechanism to remove the asynchronicity assumption. Note that we rely on chain-dependent clocks on the chain. In particular, time is based on block generation. This hinders a possibility of synchronization across chains. We analyze possible attacks based on this in Section 10.6.2.

We model blockchain transactions as a tuple

$$tx = (fromAddr, toAddr, amt, \{conds\})$$

where $fromAddr$ is the sending address, $toAddr$ is the receiving address, amt is amount of token sent and $conds$ are the conditions needed to redeem tx. In UTXO-based blockchain, $conds$ is never empty

[1] See the repository here: https://github.com/disnocen/mp-htlc

because the redeemer has to present at least a signature as a form of zero knowledge proof of knowledge of private key. On the other hand, we assume $conds = \emptyset$ if tx is in an account-based blockchain. For a difference between the two models see Chapter 3.

The MP-HTLC protocol has multiple participants. We call $P_1, \ldots, P_N$ the participants that have funds on BC_1 and $Q_1, \ldots, Q_N$ the participants that have funds on BC_2. We model the set $\{P_1, \ldots, P_N\}$ as a set of participants containing at least one rational participant, P_r and the set $\{Q_1, \ldots, Q_N\}$ as a set of participants containing at least one rational participant, Q_r.

We assume $P_1, \ldots, P_N$ are "on the same side", meaning that no P_i is actively dishonest towards $\{P_1, \ldots, P_N\} \setminus P_i$, while allowing for passive dishonesty, i.e. willing to read messages which are not sent to it (the honest-but-curious model) and the ability for P_i to go offline and not replying to messages sent by $\{P_1, \ldots, P_N\} \setminus P_i$ towards P_i. The same goes for $Q_1, \ldots, Q_N$.

Note that this assumption means that there can be no meaningful collusion between $P_1, \ldots, P_N$ and $Q_1, \ldots, Q_N$: if P_i colludes with one of $Q_1, \ldots, Q_N$, then if P_i acts on the result of the collusion then P_i would be actively dishonest towards $P_1, \ldots, P_N$.

On the other hand, we allow $P_1, \ldots, P_N$ to be actively dishonest towards $Q_1, \ldots, Q_N$, and viceversa, as long as this does not result in a situation which is equivalent to collusion. In particular, we do not allow e.g. P_i to bribe or otherwise manipulate the behavior of Q_j since then Q_j would act actively dishonest manner towards $Q_1, \ldots, Q_N$. Note that this last requirement intuitively "normal" since it is the natural extension of the HTLC between two parties A and B: in that case it is assumed that A can not control B, since if that were not the case, A could steal from B without having to start a HTLC.

Finally, since we present a HTLC-based protocol, we require that the domain of the hash function has the same size in both chains. Possible attacks that can be performed if this assumption is not met are presented in [MR97]. In practice all examples will use the SHA256 hashing algorithm, so the domain of the hash function will obviously be the same.

10.2 Multi Party computation

Similar to HTLC, MP-HTLC also has a secret creation phase. This secret cannot be created by a single participant, who would become a kind of leader in the protocol, decreasing its decentralization and security. For this reason the creation of the secret must be a joint computation performed by all parties involved. That means that we are in a distributed computing scenario and secure multi-party computation (MPC) is used to to be sure that the joint computations are secure: the goal of MPC is for a group of participants to learn the correct output of some previously agreed function applied to their private inputs without revealing anything else. To practically create the secret, the participants will use MPC to sum their private inputs.

In MPC security is defined using the *real-ideal paradigm*. In the ideal world parties interact with a third party which is completely trusted, cannot be attacked and would never betray any of the parties. Of course I can't use the same model for the real world, because no entity can be absolutely trusted. In the real world the third part is replaced by a protocol. Intuitively, this protocol is considered secure if any consequences caused by an adversary in the protocol in the real world could also be achieved in the ideal world.

There are two different adversary models commonly used for MPC: *semi-honest* and *malicious* adversary. A semi-honest adversary is one who corrupts parties but follows the protocol as specified. This is the case where parties can collude by pooling their views (i.e. private inputs) together. This kind of adversary is considered passive, because they cannot act on the knowledge gained. Another name for semi-honest is *honest but curious* adversary.

The other kind of adversary is the *malicious* (also called active) one. This kind of adversary may cause corrupted parties to deviate arbitrarily from the prescribed protocol. A malicious adversary has all powers of a semi-honest one, but it can also act on the knowledge it gains manipulating the outputs of the protocol. For more details see for example [EKR18].

To implement this part of the protocol, I used the framework proposed by [Kel20]. The framework proposes several methods to perform MPC protocols between two or more participants in different types of adversaries. Given the modularity of MP-HTLC, the choice of the specific MPC protocol does not affect the other routines. For this reason the proof of concept proposed in that paper uses the Tale protocol to do the sum.

10.3 Design

The goal is to make a multiparty-swap protocol between two blockchains minimizing the number of transactions needed while ensuring the same level of security. I do that by generalizing the HTLC to multiple participants (multiparty-HTLC) with the same secret. In the following I will be blockchain agnostic. I call blockchain BC_1 the first blockchain and blockchain BC_2 the second blockchain. The N participants in blockchain BC_1 will be $P_1, \ldots, P_N$ and the N participants in blockchain BC_2 will be $Q_1, \ldots, Q_N$. In the course of the explanation, $P_1, \ldots, P_N$ will also be called the initiators and $Q_1, \ldots, Q_N$ the finalizers. The protocol is easily extendable to the case where a P_i serves both Q_i and Q_j or where a P_i is served by Q_i and Q_j for some i, j, so that there is a different number of participants in the two blockchains, say N in BC_1 and N' in BC_2. In fact, I can represent a participant as dual, instead of a single participant with multiple funds, so that I end up having the same number of participants in both blockchains.

The only assumption on both blockchains is that they are able to understand the same hash function and have a basic scripting language. An example blockchain is Bitcoin, but also Ethereum or Tezos are supported. On the other hand blockchains like Monero or Komodo cannot be used as they do not have a scripting language.

In the course of explaining the design in the protocol, I will refer to the two types of blockchain: account-based blockchains and UTXO-based blockchains. The protocol is divided into three phases: Precommit, Commit and Redeem. Table 10.1 gives an overview of the whole protocol and the different steps needed in the UTXO and Account based models.

<table>
<tr><th></th><th>UTXO Based</th><th>Account Based</th></tr>
<tr><td rowspan="3">Precommitting Phase</td><td>Secret Creation (MPC)</td><td>Secret Creation (MPC)</td></tr>
<tr><td rowspan="2">Aggregate PubKey Creation</td><td>Aggregate PubKey Creation</td></tr>
<tr><td>Smart Contract deployment</td></tr>
<tr><td>Committing Phase</td><td>P2SH-tx sending to AggPubKey</td><td>Smart Contract funding</td></tr>
<tr><td rowspan="2">Redeeming Phase</td><td rowspan="2">P2SH-tx sending to receivers</td><td>Smart Contract activation</td></tr>
<tr><td>Smart Contract sends transactions to receivers</td></tr>
</table>

Table 10.1: Phases of the MP-HTLC protocol in the UTXO and Account based models. The Account based model needs more steps than the UTXO one.

10.3.1 Precommitting Phase

In the first part of the protocol, there are two objectives:

- *Creation of the secret*: this part is similar to the creation of the secret in HTLC and is done only by one group of participants.
- *Creation of the aggregate public key*: this is done by everyone and it is necessary for all participants to have the same possibility to manipulate the money.

In the following I will explain in detail these two subroutines. This phase is carried out in parallel by the two groups.

Secret creation

Participants $P_1, \ldots, P_N$ undertake a process of creating a secret. To do so, participants use a multiparty computation protocol with the goal of obtaining an output from hidden inputs. Formally, $P_1, \ldots, P_N$ from private inputs $(x_1, \ldots, x_n)$ create an output $y = f(x_1, \ldots, x_n)$. Examples for a function f are $sum(x_1, \ldots, x_n)$, $max(x_1, \ldots, x_n)$ or $min(x_1 \ldots, x_n)$. To be more concrete, let's assume that f is the sum of the private inputs.

After creating this secret number, all participants can hash it, similar to what happens in a HTLC. The secret is therefore y and the hash is $h := H(y)$, where H is a hash function that can be computed on both blockchain BC_1 and blockchain BC_2

Aggregated Key

Participants of both blockchains create an aggregate key using threshold signatures. Threshold signatures are supported by the major signature schemes that are used (or planning to be used) in each blockchain. Examples of digital signature schemes that support threshold signatures are ECDSA, Schnorr, EdDSA, and BLS signature. This is necessary because participants need a shared key to manage the transfer of coins: if the key weren't shared among all people, then there would be a leader. And this is not acceptable since I want a peer-to-peer protocol.

Remember that the aggregate key is inherently public, as explained in Section 5.2, and that I don't know its private key and that the public key is different from the address on the blockchain. In fact, the address on the blockchain is a derivation of the public key. In the case of Bitcoin for example the address is a derivative of the script used within the transaction.

In this part we see the difference between a blockchain with an account-based model and one with a UTXO-based model. A blockchain with UTXO-based model uses particular constructs within the transaction, while a blockchain with account-based model deploys a Smart Contract.

Whatever mechanism is used in this phase, it must have the ability to be inactive for this phase. In fact, if it were active the other participants could take the tokens without giving their own in return. Inactivity can be achieved with time-locks or activation switches in the smart contract.

This mechanism must have two possibilities of redemption of funds. The first is to be used in case all participants are honest. In practice this means that the money goes from the group of participants $P_1, \ldots, P_N$ to $Q_1, \ldots, Q_N$ or vice versa. The second way of redemption is instead used in the event that the other participants are dishonest: assuming that the mechanism is initiated by $P_1, \ldots, P_N$, the second way gives them the possibility to take back their tokens in case $Q_1, \ldots, Q_N$ behave dishonestly.

10.3.2 Commit Phase

At this phase, the parties commit to sending money. Participants in both sets send transactions, whether this is a particular transaction or a transaction to the smart contract, in order to prove that they have the tokens and are willing to continue the protocol. In order not to confuse the transactions that occur in the different phases, I will call these transactions the $inTx$ transactions, while the other transactions in the Redeeming Phase will be called the $outTx$ transactions.

These transactions will be understood in detail in Section 10.4 where I present the case study between Bitcoin and Ethereum. This is the end of the Commit Phase.

10.3.3 Redeem Phase

The last step is performed sequentially. Each step is done first by one group of participants and then by the other group of participants, when the latter receives confirmation that the intermediate step has been done. The goal of the participants in this step is to redeem all the tokens.

The protocol initiators $P_1, \ldots, P_N$ perform the first $outTxes$ to redeem the transactions or smart contract of the finalizers $Q_1, \ldots, Q_N$.

In the case of a blockchain with UTXO model, this type of transaction also has paths to be redeemed: the first if the finalizers are honest and the second in case they are dishonest. These transactions can be signed with the aggregate key created in the first step, contain h and can be redeemed by the finalizers using the y secret. In the case of a blockchain capable of supporting Smart Contracts, the logic is already all built into it and since the Smart Contract is already deployed in the first phase, this transaction simply consists of activating it.

After finalizers see the transactions made by initializers, they can proceed with their transactions and redeem the funds on the other blockchain, having seen the secret y in the blockchain.

10.4 Ethereum-Bitcoin Case Study

For the purpose of this proposal, I create a proof of concept of a MP-HTLC between the Bitcoin and the Ethereum blockchains. I made this choice because the former blockchain has a UTXO-based model while the latter has an account-based model.

Throughout this section, the notation changes slightly from the previous section to make it more intuitive according to the case study. Table 10.2 give a schematic view of what happens. For easiness

	Bitcoin	Ethereum
Precommitting Phase	Secret Creation (MPC)	Aggregate PubKey Creation
	Aggregate PubKey Creation	Smart Contract deployment
Committing Phase	P2SH-tx sending to AggPubKey	Smart Contract funding
Redeeming Phase	P2SH-tx sending to receivers	
	Smart Contract activation	
	Smart Contract sends transactions to receivers	

Table 10.2: Steps for the MP-HTLC protocol between the Bitcoin and Ethereum blockchain. The Precommit and Commit phases are done in parallel. The Redeeming phase is sequential.

of explanation I distinguish between "bitcoiners", participants who have bitcoin and want ethers, and "etherers", participants who have ethers and want bitcoin. Formally, I assume there are $2N$ participants such that $B_1, \ldots, B_N$ are bitcoiners and $E_1, \ldots, E_N$ are etherers. I proceed with the description of the three phases of the MP-HTLC protocol.

10.4.1 Precommitting Phase

In the first phase I assume that each B_i has already chosen an E_i and exchanged with it all the necessary information to proceed to a classical atomic swap (e.g. pubkeys, BTC/ETH rate, timeouts, and amount of coins). This part is outside the scope of the Chapter and can be performed via public billboards, order-books or, theoretically, even with Automated Market Makers.

In this phase and the next one, bitcoiners and etherers proceed in parallel without interaction. For ease of explanation, however, I first explain what happens between bitcoiners and then what happens between etherers.

Bitcoiners Bitcoiners have two goals for the precommitting phase: to create a secret and its hash for the MP-HTLC, and to create a shared public key. Bitcoiners $B_1, \ldots, B_N$ use a distributed key generation mechanism $Thresh - DKGen$ to create an aggregate public key $AggPk^{BTC}$.

Then bitcoiners $B_1, \ldots, B_N$ create a secret all together. It is necessary that the secret is created by all bitcoiners, otherwise there would be a leader. For this reason the secret s is created in multiparty computation, as described in Section 10.2. From the secret s, all participants can build the same hash $h = H(s)$ (where $H(\cdot)$ is a hash function) and use it in the second set of transactions in the Redeem Phase.

After that, each B_i is responsible to send h to E_i. Although it is not necessary for everyone to do this from a technical point of view, this overhead is necessary from a social point of view: this way each etherer E_i does not have to trust other etherers $E_1, \ldots, E_{i-1}, E_{i+1}, \ldots E_N$.

Etherers Etherers have two goals for the precommitting phase: to create a shared public key, and to create and (after receiving h) deploy a smart contract SC. The smart contract contains the necessary logic for either receiving and sending funds based on the presentation of a preimage of the hash h or returning of funds to the owners. This smart contract is the analogue of the P2SH transactions made by bitcoiners. Etherers first use the same distributed key generation mechanism used by bitcoiners to create an aggregate public key $AggPk^{ETH}$.

This step is also necessary in an account-based blockchain like Ethereum. This is because many of the operations can only be done by the person who creates the smart contract. Formally the smart contract must be deployed on the blockchain using the address associated with the public key $AggPk^{ETH}$.

Deploying a smart contract means publishing a transaction to a particular address. For this reason it is possible to use threshold signatures of $E_1, \ldots, E_N$ to sign and publish it.

The main functions of the smart contract [2] are:

- `activationSwitch`: this function makes the smart contract active; before SC is activated, sending a preimage of the hash (however correct) does not allow the release of the coins contained in the smart contract

[2] See the smart contract at https://github.com/disnocen/mp-htlc/blob/master/Interop.sol

- `isSha256Preimage`: this function takes care of sending funds to bitcoiners upon presentation of the right h hash preimage if SC has been previously activated via the `activationSwitch` function

As of now the smart contract is deployed but unfunded and not activated. Both the funds and the activation arrive in the Commit Phase. This concludes the precommitting phase

10.4.2 Commit Phase

Bitcoiners In this phase, the goal of Bitcoiners is to fund $AggPk^{BTC}$ with a transaction. Using PSBT (see Section 9.3) $B_1, \ldots, B_N$ are able to send only one transaction with multiple inputs and only one output. I call this transaction ptx

Using both the aggregate public key $AggPk^{BTC}$ and the hash h, each bitcoiner B_i creates a P2SH transaction (see Section 3.1.2) I call ${}_{in}tx_i^{BTC}$. The locking script of this transaction has two redeeming paths. The first path allows redemption before a timeout ${}_{in}\Delta$ by signing the redeeming transaction via a threshold signature. That's because, for this redeeming path, the redeeming transaction has to be valid when verified by $AggPk^{BTC}$. The second redeeming path allows B_i to redeem ${}_{in}tx_i^{BTC}$ as the timeout ${}_{in}\Delta$ passes using a signature verifiable by some public key Pk_i^{BTC} owned (together with the related private key) by B_i. In the Bitcoin Script language, this output script looks similar to the following one where I put `InDelta`$={}_{in}\Delta$, `AggPkBtc`$=AggPk^{BTC}$ and `BiPkBtc`$=Pk_i^{BTC}$:

```
IF
    <now + InDelta> CHECKLOCKTIMEVERIFY DROP
     <BiPkBtc> CHECKSIGVERIFY
ELSE
    <AggPkBtc> CHECKSIGVERIFY
ENDIF
```

These transactions are P2SH scripts that produce an address $SHAddr_i^{BTC}$. Note that $SHAddr_i^{BTC}$ is different for script. The address format doesn't take amounts, private keys or transaction hashes into consideration. However, given that public key Pk_i^{BTC} is different for each B_i in the second redeeming path, the redeeming script is different. Therefore there are multiple outputs in the PSBT ptx.

Then all the $B_1, \ldots, B_N$ complete together the PSBT-sequence and broadcast the transaction. they get transaction hash ${}_{in}txHash^{BTC}$ and they inform $E_1, \ldots, E_N$ about this hash.

Etherers Having created the smart contract SC in the precommitting phase with all the necessary logic, this phase is easier for etherers. $E_1, \ldots, E_N$ just send their funds (plus fees) to the smart contract. Then they wait for bitcoiners $B_1, \ldots, B_N$ in the redeem phase. Now SC is funded, but still inactive

10.4.3 Redeem Phase

This phase is sequential. If each bitcoiner B_i informs E_i about ${}_{in}txi^{BTC}$ and if the etherers complete their part of the precommitting and committing phases (visible on the blockchain, so they can keep track), then all participants start the last phase of the protocol. The goal of this phase is the actual transfer of funds from $SHAddr_i^{BTC}, \quad i = 1, \ldots, N$ to etherers $E_1, \ldots, E_N$ on the Bitcoin blockchain and from the smart contract SC to bitcoiners $B_1, \ldots, B_N$.

At the formal level, $B_1, \ldots, B_N$ must make a single transaction ${}_{out}tx^{BTC}$ that has as input the transaction outputs identified by the ${}_{in}txHash^{BTC}$ and as output Bitcoin addresses redeemable by $E_1, \ldots, E_N$. $B_1, \ldots, B_N$ must then produce one threshold signature to publish this transaction.

The output addresses in this transaction are derived from P2SH transactions as in the Commit Phase. In fact each output has two redeeming paths as well (one before and one after a timeout I call ${}_{out}\Delta$, which is generally different from ${}_{in}\Delta$) to defend bitcoiners from possible non-activation by etherers. So this transaction is not a standard one and needs to be encapsulated in a P2SH transaction. For each i-th output, the first redeeming path expires at timeout ${}_{out}\Delta$ and needs the preimage of hash h, as well as the signature verifiable from the public key provided by E_i. The second redeeming path becomes active after the timeout ${}_{out}\Delta$ and needs only the signature verifiable by the public key of B_i. I stress that ${}_{out}tx^{BTC}$ is a single transaction with multiple output addresses.

For each output address, its locking script is (where `OutDelta`=$_{out}\Delta$, `h`=h, `EiPkBtc` is the Bitcoin public key provided by E_i and `BiPkBtc`=Pk_i^{BTC}):

```
IF
    <now + OutDelta> CHECKLOCKTIMEVERIFY DROP
     <BiPkBtc> CHECKSIGVERIFY
ELSE
    SHA256 h EQUAL <EiPkBtc> CHECKSIGVERIFY
ENDIF
```

The etherers know when the transaction they are interested in has been published by the bitcoiners because knowing $_{out}\Delta$ they can compose the P2SH script and therefore they can monitor the P2SH address. When etherers see the address funded, they activate the smart contract using the `activationSwitch` function. After the activation they wait for one of $B_1, \ldots, B_N$ to send the preimage into the smart contract. This triggers the sending of coins to the addresses of bitcoiners $B_1, \ldots, B_N$ on Ethereum.

When this happens, each etherer E_i can learn about the preimage by parsing the blockchain. With this preimage, each E_i can redeem the i-th output of $_{out}tx^{BTC}$.

The protocol is then concluded.

10.4.4 Implementation

I propose an implementation of this use case. The link can be found here[3].

To implement the threshold signature I used the ZenGo project[4]. I modified the demo so that it would sign particular digests of transactions from the input. In particular I used the distributed key generation and the distribute signing protocol from [GG18]. This is one possible choice of many and I could use other threshold protocols. Then I proceeded to create scripts for both Bitcoin and Ethereum. To explain what I did I split this explanation into two parts

Bitcoin I used the bitcoin-libs Python 3 library to create the transactions. I modified the library to make the transaction signed by the ZenGo library.

Namely I did five steps:

- I create the aggregated key using the distributed key generation algorithm of the rust library and derived a P2PKH address; I call this address $AggAddrBTC$
- I sent coins to this $AggAddrBTC$
- I created a transaction to send the money to another address, I call this address $NewAddr$, and I get the digest.
- I used the rust library's API to sign in a distributed way the digest
- I sent the transaction[5]

Ethereum In the case of Ethereum I made these steps:

- I create the aggregated key using the distributed key generation algorithm of the rust library and derived an Ethereum address; I call this address $AggAddrETH$
- I created a smart contract able to manage the funds, as explained in the previous subsection; the owner of the smart contract is the $AggAddrETH$
- I modified the `ethereumjs/tx` creating a new function I called `signTh`: this function calls the rust library to sign the transactions so that it is verifiable using the aggregated public key behind $AggAddrETH$

[3] https://github.com/disnocen/mp-htlc

[4] https://github.com/ZenGo-X/multi-party-ecdsa

[5] You can see the transaction at this URL: https://live.blockcypher.com/btc-testnet/tx/40b0dcfdfc6461f79aabf88660f50adc03a32c0a3c40b5d4af5ddf161243909b/

10.5 EVM-compatible Blockchains Case Study

Here I present another use case, namely a Ethereum-Polygon token exchange. After reading the section, it will be easy to see how this way to exchange tokens between blockchains can be used for any couple of EVM-compatible blockchain, effectively creating an alternative to centralized bridges such as the Binance smart chain bridge or the Polygon one. This use case is different from the previous one in two ways. The first difference concerns the model of the blockchains involved. In fact, both the Ethereum and Polygon blockchains are account-based: in this case MP-HTLC is smart contract based on both sides. The second difference is in terms of purpose. As stated by the Polygon whitepaper, the project aims to be an interoperable sidechain of Ethereum to achieve scalability. Polygon has faster block-creation time and cheaper fees. Currently many blockchain DApps are creating Polygon version of their services, especially DeFi services like AAVE [Aav22], SushiSwap [Int22] or Uniswap [AZS+21]. Note that the idea of using new EVM-compatible blockchains to solve Ethereum's scalability problems is not new and is also used by blockchain-based video games such as Axie Infinity [Axi21] which created its own EVM-compatible blockchain for game management.

Of course Polygon already provides a *bridge* to exchange tokens between the Polygon network and Ethereum, but this bridge is centralized and therefore is subject to possible censorship and theft. Here I propose a decentralized alternative using MP-HTLC.

To understand why our alternative can substitute the whole bridge, the reader should recall two things. The first one is that every token on a EVM-compatible blockchain are the result of the deploy of a smart contract which then manages the balances of the users. The second thing to remember it that given two EVM-compatible smart contracts SC_A and SC_B, it is possible to call a function in SC_B from SC_A. I use the notation $SC_B.functionName(\cdot)$ to symbolize the call from SC_A. Therefore the MP-HTLC protocol can be used to exchange tokens based on smart contracts that have a version deployed on the Ethereum blockchain and one deployed in the Polygon blockchain. Some examples of these tokens are the USDT token or the WrappedBTC token[1].

I model the group of participants in this way. I assume there are N participants that own some amount of $TokenA$ on the Ethereum blockchain and want the same amount of $TokenA$ on the Polygon blockchain. I denote these participant as $E_1^A, \dots, E_N^A$ and I call them etherers, as in Section 10.4. Similarly I assume there are N participants that own some amount of $TokenA$ on the Polygon blockchain and want the same amount of $TokenA$ on the Ethereum blockchain. I denote these participant as $P_1^A, \dots, P_N^A$ and I call them polygoners. So there are $2N$ participants in total

I proceed with the description of the three phases of the MP-HTLC protocol in the case of EVM-compatibility blockchains.

10.5.1 Precommitting Phase

In the first phase I assume that each E_i^A has already chosen an P_i^A and exchanged with it all the necessary information to proceed to a classical atomic swap (e.g. pubkeys, timeouts, and amount of coins). Note that there is no rate of exchange for $TokenA$ because we are assuming this token is pegged. As in the previous use case, how the participants exchange this data is outside the scope of the Chapter and can be performed via secure chats.

In this phase and the next one, etherers and polygoners proceed in parallel without interaction. For ease of explanation, however, I first explain what happens between etherers then what happens between polygoners, so in practice without loss of generality I assume etherers will start the process.

Etherers Etherers have three goals for the precommitting phase: to create a secret and its hash for the multi-HTLC, to create a shared public key and to use this public key $AggPk^{ETH}$ as the identity (address) for deploying the smart contract SC_{ETH} that containing the necessary logic for managing the funds.

In practice etherers $E_1^A, \dots, E_N^A$ use the distributed key generation mechanism $Thresh-DKGen$ to create an aggregate public key $AggPk^{ETH}$ and MPC to create a secret all together, as in the previous use case. From the secret s, all participants can build the same hash $h = H(s)$ and use it in the second set of transactions in the Redeem Phase.

[1] Wrapped tokens are pegged to the original blockchain but transferable on the Ethereum or Polygon blockchain.

After that, each E_i^A is responsible to send h to P_i^A. The smart contract is the same as the one in the previous use case. The only difference is the function to send money. As explained before, there is a function call from SC_{ETH} to the smart contract of $TokenA$ on Ethereum. Formally the function is:

$$TokenA_{ETH}.send(_to,_amount)$$

which is called if the `checkHash` function returns `true`

Polygoners Polygoners have two goals for the precommitting phase: to create a shared public key, and to create and (after receiving h) deploy the smart contract SC_{POL}. Polygoners first use the same distributed key generation mechanism used by etherers to create an aggregate public key $AggPk^{POL}$.
l

The smart contract is the same as the one in the Ethereum case. The function call from SC_{POL} to the smart contract of $TokenA$ on Polygon is:

$$TokenA_{POL}.send(to, amount)$$

which is called if the `checkHash` function on SC_{POL} returns `true`

As of now both smart contracts are deployed but unfunded and not activated. Both the funds and the activation arrive in the Commit Phase. This concludes the precommitting phase

10.5.2 Commit Phase

In this phase participants on both chains send funds to the send the funds to their respective smart contracts. Now SC_{ETH} and SC_{POL} are funded, but still inactive

10.5.3 Redeem Phase

Similarly as the Redeem Phase of Section 10.4, participants activate their relative smart contract. After doing that etherers redeem the funds in SC_{POL} by sending the preimage of h. Polygoners are therefore able to get the funds from SC_{ETH}.

10.6 Analysis

In the following I present a cost analysis where I prove that MP-HTLC can be superior to HTLC in terms of number of transactions needed and an attack analysis. Finally I present a note on the security of the protocol.

10.6.1 Cost Analysis

In this section I analyze the costs of our MP-HTLC. I compare it with the standard HTLC. I prove that as long as there are at least two participants on a UTXO-based blockchain, then it is more cost effective to use MP-HTLC instead of instantiating different HTLCs.

In the standard HTLC as defined in [Nol13] (and explained in Section 5.2) there is a best case scenario and a worst case scenario. Still, because the parties exchange signatures and do not broadcast transaction, then the number of transaction is the same regardless of the best/worst scenario. In fact, according to the HTLC protocol there are two transactions published on chain to start the exchange (one per party, and therefore one per blockchain), and two other transactions published on chain if the exchange is successful. Therefore the parties will publish a total of four transactions per instantiation of the protocol

For the MP-HTLC protocol there are two different implementation of the swap. One for the UTXO model and one for the account based model. I count the number of transactions for each phase in both the UTXO and account model assuming there are n participants on each blockchain. In Table 10.3 I put a summary of the number of transaction for each phase and each model, while in Figure 10.1 I put how the progression is.

	Number of Transactions						
	HTLC			MP-HTLC			
	1st phase	2nd phase	TOT	Precommit	Commit	Redeem	TOT
Acct. based	n	n	$2n$	1	n	$n+1$	$2n+2$
UTXO based	n	n	$2n$	0	1	1	**2**

Table 10.3: The table describes the number of transactions required in the two implementations. The number of participants n is meant for each blockchain. This means that a token exchange needs a total of $2n$ participants between the two blockchains.

Precommit Phase In the UTXO model there is no need to do any transaction at this point, so in this phase there are 0 transactions done by these participants. On the other hand, participants on an account-based blockchain have to deploy a smart contract. This is the first transaction they have to do.

Commit Phase Thanks to PSBT (see Section 9.3), participants in the UTXO model broadcast only one transaction, regardless of the number of inputs. On the other hand, in the account-base model all participants have to fund smart contract on the blockchain. Therefore there is 1 transaction in the UTXO model and there are n transaction the account model model.

Redeem Phase In this last phase, participants on UTXO-based blockchain have a clear advantage. In fact it is really easy to batch transactions in UTXO-based blockchain because transactions can have multiple outputs.

Batching is currently impossible on account-based blockchain, even if there are multiple proposals that aim to address and solve this problem, especially for EVM-compatible blockchain [Zol20, Mat20a, Mat20b, WZL+21]. If it will be possible to batch more transactions into one, then the number of transaction in an account-based blockchain will be the same as the number of transactions needed in a UTXO-based blockchain.

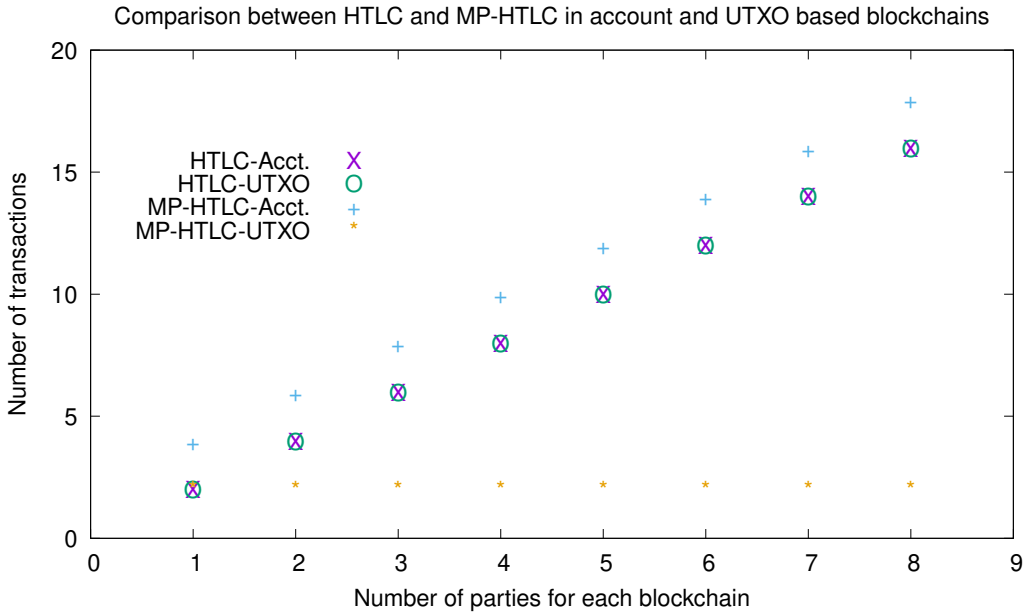


Figure 10.1: Comparison of number of transactions for HTLC and MP-HTLC protocols.

10.6.2 Attack Analysis

Multiple-blockchains modeling is more complicated than treating each part singularly, as there is extra complexity underlying the cross-chain communication. In fact, even if each blockchain has its own security model, these models can be different between blockchains. For example: the Bitcoin blockchain Proof-of-Work-based consensus algorithm is proved to work correctly only if the adversarial computation power is less than 33% [ES18]. On the other hand, some Proof-of-Stake-based consensus algorithms are proved to work correctly only if the total adversarial stake is less than 33%. Even if the two values are the same, the cost of attaining them is potentially different.

I decided to address the two blockchain attacks that have reorganization of blocks (also called *reorgs*) as a consequence: other attack would have no effect on the transactions. Note that there is no 51% attack. This is because it is not a real attack formally, but it is a state of the blockchain nodes that allows the implementation of real attacks, such as censorship or double spending. So in this sense, I do not directly mention the 51% attack, but I do mention its effects.

Double Spending This is the most famous blockchain attack and it is even mentioned in the Satoshi's Bitcoin paper [Nak08]. The goal is to trick two different entities (e.g. two different merchants) into assuming they have been paid, using the same coins (technically, the same transaction output(s) in a UTXO-model blockchain, see Section 3.1). When this attack is successful, only one of those transactions is definitely accepted by the blockchain network, but the attacker receives both products from both vendors.

This is how this attack would play out in the MP-HTLC protocol. Adversary party A claims it wants to exchange v_A coins of blockchain BC_A for an equivalent amount of v_B coins of blockchain BC_B. These coins are currently owned by party B. So A starts participating to the protocol and it sends v_A to the aggregated address $AggAddr_A$ on BC_A, but it also sends v_A to another BC_A-address $OthAddr_A$. If A can trick B into believing it is honestly behaving and B continues to follow the protocol and A's double-spend attempt is successful, then A would get v_B coins of blockchain BC_B and retain v_A coins in the BC_A-address $OthAddr_A$. B would lose its v_B coins of blockchain BC_B without gaining any v_A coins.

The easiest way to deal with this attack is by waiting for the confirmation period, where the "time" is defined as the passing of blocks' generation. In this case, participant have to wait for the confirmation period of their chain before assuming the current phase is finished and starting the next phase of the protocol. Other works, such as Sai et al. [ST19], propose methods to punish the double spend problem across two blockchains.

Similar to the Double spend attack, the Finney and Brute force attacks aim to spend the same coins twice. they are different on the social level. For example, the Brute-Force attack assumes the attacker can control a lot of computing power and it will use this power to perform the double spend. Because the mechanics of the attack are the same, I don't further analyze these variation.

Selfish Mining and Selfish Endorsing Eyal and Sirer propose the selfish mining attack [ES18], where a malicious miners is incentivized to deviate from the honest protocol withholding valid blocks until it mines two consecutive blocks. If this strategy is successful, then the miner gets double the reward because both of its blocks gets accepted forming the longer chain[2]. In the following years Sapirshtein et al. [SSZ16] identify the optimal selfish-mining policy, Nayak et al. [NKMS16], consider network attacks, and Kwon et al. [KKS+17], consider the impact of selfish mining in the context of mining pools. Still, these works treat selfish mining in the context of a Proof-of-Work consensus algorithm. The work of Neuder et al. present the *selfish behavior* attack [NMRP19], that is the selfish mining attack in the context of a Proof-of-Stake blockchain (Tezos in particular). This attack incentivizes rational bakers [3] to ignore the longest-chain rule and to create a separate two-block fork faster than the rest of the network can publish two blocks.

This attack would disrupt the procedure of the protocol because some transactions could be effectively erased from the blockchain. The solution is similar to the solution for the Double spend attack: parties should wait for the confirmation period of both blockchains before advancing to the next phase.

[2]Technically in both the GHOST and Nakamoto consensus protocols, the winning chain is the chain with the most work. In our scenario these two concepts are equivalent, so I model it using the more intuitive one

[3]bakers are the analogous of miners in the Proof-of-Stake chains. Their goal is to produce blocks and their probability to be chosen is related to how much token they hold (*bake* or *stake*)

10.6.3 A Note on Correctness

Goal of this section is to prove that the MP-HTLC protocol is correct in the same sense a HTLC is correct. Intuitively, that is because MP-HTLC follows the same construction of HTLCs, if we abstract the participants on one side as a single entity. Yet, we have to prove that this abstractions is well defined, meaning that we have to prove that (using the general notation of Section 10.3) a cheating participant between $P_1, \dots, P_N$ or $Q_1, \dots, Q_N$ would not endanger the correctness of the protocol. We will prove it by showing that if there is a dishonest participant $\widehat{P_j}$ between $P_1, \dots, P_N$, then the protocol is still correct if $Q_1, \dots, Q_N$ treat $P_1, \dots, P_N$ as all dishonest participants and abort the protocol. The same goes if there is a dishonest participant $\widehat{Q_j}$ between $Q_1, \dots, Q_N$.

We do that in two steps. In the first step we assume participants can only cheat by missing timeouts. Then we let participants cheat by sending arbitrary messages. Note that in Section 10.1 we stated that $P_1, \dots, P_N$ may be actively dishonest only towards $Q_1, \dots, Q_N$ and passively dishonest towards all participants in the MP-HTLC protocol, and that the same goes for $Q_1, \dots, Q_N$.

We need to reason around the concept of "time", an essential concept to talk about timeouts in a meaningful way. Throughout the Chapter we assumed asynchronous systems and a block-based time tracking (see Section 10.1). Assuming a constant rate of block production for each blockchain (e.g. at most 15 blocks produced in Ethereum per one block produced in Bitcoin) is enough to ensure correctness of the HTLC protocol between two parties (single entities) Alice and Bob. The question is under which assumptions this reasoning holds true in the case of multiple parties for each blockchain.

The main source of problems comes from the fact that in normal HTLCs between two parties A and B, one entity A either respects or does not respect the timeout, with no middle cases. In case A does not respect the timeout, then B acts accordingly. On the other hand, in case of multiple participants $P_1, \dots, P_N$ it is possible to have a situation in which *some* participants have been honest and respected the timeout, while others did not. We denote as $\widehat{P_j}$ the dishonest participant in the set $\{P_1, \dots, P_N\}$. In such a case, we have implicitly considered the protocol as aborting, i.e. we assumed that participants $P_1, \dots, P_N$ and $Q_1, \dots, Q_N$ stop the protocol and redeem the coins they already invested. We have to prove that this behavior does not affect the correctness of the protocol.

We start by stating what *correct* means in MP-HTLC and then we prove that the protocol is correct even in the aforementioned case.

Definition 10.6.1 *[Correctness] Let $P_1, \dots, P_N$ be participants in blockchain BC_1, powered by coin c_1, who owns amounts $a_1^1, \dots, a_N^1$ of coin c_1 respectively. Similarly, let $Q_1, \dots, Q_N$ be participants in blockchain BC_2, powered by coin c_2, who owns amounts $a_1^2, \dots, a_N^2$ of coin c_2 respectively. Finally, assume $c_1 = \rho c_2$ with ρ the agreed exchange rate between c_1 and c_2 and $\frac{a_i^1}{a_i^2} = \frac{\rho c_2}{c_1}$, so that the value $a_i^1 c_1$ has the same (monetary) value as $a_i^2 c_2$ for each $i = 1 \dots N$. Then the protocol as described in Section 12.1 is* correct *under the model of Section 10.1 if and only if at the end of the protocol:*

1. *Either any rational participant in $P_1, \dots, P_N$ obtained amounts $a_1^2, \dots, a_N^2$ of coin c_2 and any rational participant in $Q_1, \dots, Q_N$ obtained amounts $a_1^1, \dots, a_N^1$ of coin c_1*
2. *Or any rational participant in $P_1, \dots, P_N$ still own amounts $a_1^1, \dots, a_N^1$ of coin c_1 and any rational participant in $Q_1, \dots, Q_N$ own amounts $a_1^2, \dots, a_N^2$ of coin c_2*

In other words, Definition 10.6.1 states that the MP-HTLC protocol is correct if and only if participants either get the amounts of coins they want (Point 1 of the definition) or they still own the initial amount of funds (Point 2 of the definition), in accordance with the requirements of a general CCC [ZAZ+21] as presented in Section 5.2. We are ready to prove that MP-HTLC is correct in the case of participants missing timeouts.

Proposition 10.6.1 *Assume participants $P_1, \dots, P_N$ need to exchange an amount $a_1^1, \dots, a_N^1$ of coin c_1 on blockchain BC_1 for an amount $a_1^2, \dots, a_N^2$ of coin c_2 on blockchain BC_2, and assume participants $Q_1, \dots, Q_N$ have an opposite need. Assume there is at least one rational participant P_r among $P_1, \dots, P_N$ and at least one rational participant Q_r among $Q_1, \dots, Q_N$.*

Assume $P_1, \dots, P_N$ and $Q_1, \dots, Q_N$ agree on using MP-HTLC and that participant $\widehat{P_j}$ is dishonest and does not follow the protocol by missing the timeout Δ, as defined in Section 10.3.2. Then if all participants $P_1, \dots, P_N$ and $Q_1, \dots, Q_N$ abort the protocol, then the protocol MP-HTLC is correct in the sense of Point 2 of Definition 10.6.1. This is valid also in the case participant $\widehat{Q_j}$ is dishonest and does not follow the protocol by missing the timeout Δ' as defined in Section 10.3.3.

Proof 10.6.1 *Recall that we assumed participants can get the current state of both blockchains BC_1 and BC_2 at any time.*

Assume dishonest participant $\widehat{P_j}$ does not follow the protocol at the Commit Phase (Section 10.3.2) and by the time the timeout Δ passes $\widehat{P_j}$ has not sent its own inTx *transaction. Then all participants in the set $\{P_1, \ldots, P_N\} \setminus \{\widehat{P_j}\}$ can redeem their own* inTx *transaction. Furthermore, participants $Q_1, \ldots, Q_N$ see this behavior by observing the blockchain BC_1. Since there is at least one rational participant in the $\{Q_1, \ldots, Q_N\}$ set it is clear that it is impossible for them to conclude the Redeem Phase, since Q_r will not participate in signing the* outTx *(Section 10.3.3), forcing $Q_1, \ldots, Q_N$ to redeem their* inTx *transaction. Therefore the protocol aborts and $P_1, \ldots, P_N$ own amounts $a_1^1, \ldots, a_N^1$ of coin c_1 respectively and $Q_1, \ldots, Q_N$ own amounts $a_1^2, \ldots, a_N^2$ of coin c_2 respectively, which is the case presented in Point 2 of Definition 10.6.1. Note that this reasoning also applies to the case where dishonest participant $\widehat{Q_j}$ has not sent its own* inTx *transaction on BC_2 by the time the timeout Δ passes in the Commit Phase, since rational participant P_r will not sign* outTx.

On the other hand, assume all participants $P_1, \ldots, P_N$ and $Q_1, \ldots, Q_N$ completed the Pre-commit and Commit Phases successfully, and they are in the Redeem Phase. Then $P_1, \ldots, P_N$ have to sign a transaction together using threshold signatures as in Section 4.4, regardless of whether they are in a UTXO based blockchain or in a Account based blockchain. Since $\widehat{P_j}$ can only cheat by missing deadlines, we assume $\widehat{P_j}$ participates in it. Assume for the same reason that $Q_1, \ldots, Q_N$ also follow the protocol. The only deadline $\widehat{P_j}$ can miss is the redeeming of outTx *in BC_2. But in this case $\widehat{P_j}$ would not be rational and the correctness definition does not apply to him since $\widehat{P_j}$ goes against his own interest. Interestingly, Q_j if rational will be able to redeem its own coins on BC_1 regardless, since Q_j can surely see the preimage of the hash from rational participant P_r: which is the case presented in Point 1 of Definition 10.6.1.*

After proving that if one participant misses a deadline then the protocol is still correct, we now see that this remains true for any kind of dishonest behavior, provided it is compliant with the model in Section 10.1. We do that by showing that any dishonest behavior involving transactions is equivalent to missing a timeout and therefore already covered in Proposition 10.6.1.

Proposition 10.6.2 *Assume a setting similar to Proposition 10.6.1 where $P_1, \ldots, P_N$ and $Q_1, \ldots, Q_N$ are allowed to have dishonest behavior according to the model in Section 10.1. Then the protocol as described in Section 12.1 is* correct*, as in the Definition 10.6.1.*

Proof 10.6.2 *The easiest way to prove that is by going through each of the phases. In the Pre-commit phase there is no transaction involved, so any cheating has no effect on any fund.*

In the Commit phase, the only transactions are the one analyzed in Proposition 10.6.1 and are "on the same side", meaning $P_1, \ldots, P_N$ do transactions that are redeemed by either $P_1, \ldots, P_N$ together with a threshold signature or each participant will redeem its own. All possible kinds of dishonest behavior in this phase have to be equivalent to missing the timeout Δ: dishonest participant $\widehat{P_j}$ can not steal funds from P_i in any way since it is not allowed by the model described in Section 10.1. Therefore the whole Commit Phase for $P_1, \ldots, P_N$ is covered by Proposition 10.6.1. The same works also for $Q_1, \ldots, Q_N$.

Finally, regarding the Redeeming Phase, $P_1, \ldots, P_N$ are required to essentially send funds to $Q_1, \ldots, Q_N$ and viceversa. Assume a dishonest participant $\widehat{P_j}$ which blocks this step: then the redeeming phase can not start, which is basically a missed timeout and parties can redeem their own funds. As in Proposition 10.6.1 then, the protocol remains correct thanks to Point 1 of Definition 10.6.1.

10.6.4 A Note on Security

While the HTLC is vulnerable to bribery attacks (see e.g. [JSZ+19] and [WHF19]), recently Tsabary et al. presented MAD-HTLC [TYME21]. In MAD-HTLC the authors they modify the classic HTLC adding a *redeem path* that benefits the miner in case Bob is dishonest but benign (i.e. rational and willing not to follow the protocol in case of potential reward but prefers to act honestly for the same reward). This contract is called MH-Dep.

To prevent the case where Bob is spiteful (i.e. dishonest even if he would lose the reward) the authors introduce a collateral contract (MD-Col). MD-Col can be redeemed by Bob if he's honest, or by any miner if Bob tries to be dishonest.

Using MAD-HTLC instead of HTLC in a cross-chain communication doesn't provide atomicity anymore: MAD-HTLC uses the mutual assure destruction (MAD) principle where if a party misbehave, then all parties lose everything.

MP-HTLC can obtain synchrony with both MAD-HTLC and HTLC thanks to the modularity of the construction, but for ease of explanation I presented the MP-HTLC protocol using HTLC.

10.7 Conclusions

In this chapter we presented a method to achieve a hash-time-lock contracts atomic swap between multiple participants in the same instance in order to decrease costs and increase user privacy. We have also presented two use cases. The first between Bitcoin and Ethereum, where we also linked the code used to instantiate the protocol and produce a test transaction. The other use case is done between two EVM-compatible blockchain.

To prove that the MP-HTLC protocol satisfies correctness as the HTLCs, we described a suitable model where the participants can be abstracted as one rational entity and proved correctness in that model. To achieve that, we assumed participants on one blockchain are not willing to perform active dishonest acts to each other. The rationale has been that one single participant would act in its own self-interest if rational, and we decided to treat the whole group as rational. Surprisingly, we proved that under these assumptions it is possible to treat the whole group as rational by requiring only *one* rational party per group of participants. We argued that for protocol correctness purposes requiring one rational party for each group allows for the treatment of the whole group of participants $P_1, \ldots, P_N$ and $Q_1, \ldots, Q_N$ as two separate single entities.

Chapter 11

Time Based Methods

In the previous chapter on MP-HTLC [BS22b] (Chapter 10), I presented a way to mitigate the privacy and slowness problems of HTLCs (see Section 5.3.2). These are two out of three HTLC related problems, so the natural question is "how can we solve the third problem?", namely the reliance on the scripting capabilities of the blockchain. The answer is in the affirmative and the proof is explained in this chapter.

Instead of relaying on blockchain scripting capabilities, I devised a protocol to move much of the computation off chain. The construct I used is called Time-Lock puzzle and it is the main primitive I use for BTLE [BMS21].

The history of time-lock puzzles and time-based encryption in general is highly interesting and I give a brief account here. By strictly adhering to the mantra of "cypherpunks write code", 1900s cypherpunks developed software to solve very practical privacy-related problems emerging in the growing "remote world". Some of these problems will never be "solved" since they are inherently cat-and-mouse games where sometimes the cat becomes the mouse and viceversa. Examples are private communication, data encryption at rest and data encryption in motion. The underlying goal of these problems is to block the access of an unwanted agent to the data indefinitely. More concretely the goal of a secure and private communication protocol is to block reading and tampering of messages by external adversaries for tens or hundreds of years. Similar goals are expected for the other problems.

On the other hand, sometimes the goal of a protocol is to prevent the receiver of a message (or of a piece of data in general) from reading it before a predetermined and, relatively short, time. Notorious cypherpunk Timothy C. May called such protocol "cryptographic time-capsule".

While May considered a construction via third parties, others proposed techniques to solve cryptographically the time-capsule problem without third parties. These constructs go by the name of "timed primitives" and they came up in several contexts. In the following I distinguish between a pre-blockchain phase and a post-blockchain phase. The first generation, pre-blockchain, includes time-capsules for key escrowing as in Bellare et al. [BG96], time-based cryptographic secrets as Rivest et al. [RSW96] and contract signing as in Boneh et al. [BN00]. Of those, only the last two protocols are secure against parallel processing, i.e. they use what has been defined as *inherently secure* function [BBBF18].

After the introduction of Bitcoin [Nak08], time based cryptography had a new wave of research. In particular, the post-blockchain study of time-based cryptographic protocols is focused on *verifiable delay functions* (VDF) and *time-lock puzzles* (TLP). VDFs are a subset of TLPs: the difference is that in VDFs other people can verify the solution without the need to solve the puzzle [BBBF18].

The majority of the newer studies have focused on VDFs. Some of them proposed new protocols, such as [MT19], while others extended the TLP in [RSW96] making it a VDF. The works of the latter type are that of Wesolowski [Wes20] and that of Pietrzak [Pie18]. These works are compared in [BBBF18]. Interestingly, the new wave of research on time based cryptography has generally left aside the traditional time-lock puzzles: we are aware of only one paper on this topic, i.e. the time-lock puzzle in [LJKW18].

I propose BTLE (Broadcast Time Lock Exchange), a two steps protocol to achieve peer to peer an privacy preserving atomic swap. BTLE is split into the matching protocol, which matches two parties without an order book, and a swap protocol, which takes care of the actual swap. BTLE uses time-lock puzzles (Sections 11.1.2 and 11.1.3) since it does not need any outside verification of the result. That

is because our puzzles have an implicit verification: if the solution is right, then the user can retrieve the funds, otherwise it can not. However, I present also how to transform both time-lock puzzles into VDF in Section 11.2.

The chapter is so organized. I first present the concepts of Time-Lock Puzzle (Section 11.1) and Verifiable-Delay Function (Section 11.2), and then the design of the two-step BTLE protocol (Section 11.3). Finally I analyze the protocol from a performance perspective (Section 11.4).

11.1 Time-Lock Puzzles

My BTLE protocol is based on the notion of "timed release of cryptography." The goal is to encrypt a message in such a way that it cannot be decrypted for a predetermined amount of time. Originally, the goal was "to send information into the future," as Timothy May explained in [May93].

Another application of essentially time-based cryptography is the creation of time-lock puzzles, i.e., computational problems whose solution can be found only by brute force. In these puzzles, the time taken to find the solution is known a priori.

Below we see in more detail how to obtain these puzzles that are the basis of the BTLE protocol.

11.1.1 On time

Thanks to the sequential operations involved in these time-lock puzzles, it is possible to predict the time to solve them. Said T the time such that A wants to keep B busy, S the number of squaring per unit of time (either done by the RSW-TL method or the BM-TL method), then TS is the number of squaring needed. Clearly S strongly depends on the processor used.

Since they use sequential functions, the systems are not parallelizable. Thus, there is no advantage in having more CPUs, because all computations are necessarily done only on one core of the CPU. These time-lock puzzles can be considered as a CPU-bound puzzle with a timing function and an implicit verification [ACP20].

11.1.2 RSA-TL

The time-lock puzzle proposed in [RSW96] is simple yet effective and exploits repeated squares. Usually, a time-lock puzzle is used to encrypt a secret sk (for instance a key of a symmetric cryptosystem) so that it could be recovered only after a fixed amount of time T.

Let A be the entity that creates the time-lock puzzle, A encrypts sk by means of

$$c \equiv sk + a^{2^t} \pmod{n}$$

where $n = pq$, with p and q prime numbers, $0 < a < n$ a random number and t a positive number computed as before. If p and q are known, one can efficiently compute c, observing that 2^t can be reduced modulo $\varphi(n)$, i.e., $e \equiv 2^t \pmod{\varphi(n)}$, and then one just has to compute $a^e \pmod{n}$ with a great reduction of the repeated squares needed for obtaining c. The entity A sends (n, a, t, c) to the entity B that has to recover sk.

Since p and q are kept secret by A, the entity B has to perform t squarings, since the computation of a^{2^t} is believed not to be parallelizable. In fact, multiplication of "big" primes is the same trapdoor function used by the RSA cryptosystem.

11.1.3 BM-TL

The idea of Rivest et al. for creating time-lock puzzles can be easily adapted using different products for performing the powers. Given a field $\mathbb{F}$, the conic

$$\mathcal{C} = \{(x, y) \in \mathbb{F} \times \mathbb{F} : x^2 + Hxy - Dy^2 = 1\},$$

with D square-free, can be equipped with the product

$$(x_1, y_1) \otimes (x_2, y_2) = (x_1x_2 + y_1y_2D, x_1y_2 + x_2y_1 + y_1y_2H),$$

so that $(\mathcal{C}, \otimes)$ is an abelian group with identity $(1, 0)$ and the inverse of an element (x, y) is $(x+Hy, -y)$. One can easily observe that $\mathcal{C}$ is isomorphic to $\mathbb{A} = \mathbb{F}[X]/(X^2 - Hx + D)$ and then we can parameterize the conic by considering $P = \mathbb{A}^*/\mathbb{F}^* \cong \mathbb{F} \cup \{\alpha\}$, where $\alpha \notin \mathbb{F}$ is the point at the infinity. In this way, the induced product over P is given by

$$\begin{cases} a \odot b = \dfrac{D + ab}{H + a + b}, & \text{if } a + b \neq -H \\ a \odot b = \alpha, & \text{if } a + b = -H \end{cases}.$$

It is interesting to observe that the same parameterization can be obtained by using the line $y = \dfrac{1}{m}(x + 1)$. In [BM16], the authors developed an RSA–like cryptosystem based on these groups in the particular case where $\mathbb{F} = \mathbb{Z}_p$ and $H = 0$. In this case, when D is not a quadratic residue modulo p we have that $\mathcal{C}$ is a cyclic group of order $p + 1$ [BM16] and thus

$$a^{\odot p+1} \equiv 1 \pmod{p}$$

for every $a \in P$, where the powers are evaluated with respect to the product $\odot$. Moreover, we can also consider the conic $\mathcal{C}$ with points whose coordinates belong to $\mathbb{Z}_n$. In this case, we do not have a group structure, however, considering $P = \mathbb{Z}_n \cup \alpha$, with $n = pq$, p and q primes, we have an analogue of the Euler's totient theorem:

$$a^{\odot \Psi(n)} \equiv 1 \pmod{n}, \quad \forall a \in \mathbb{Z}_n^*,$$

where $\Psi(n) = (p + 1)(q + 1)$, see [BM16].

Given the possibilities of the conic described in [BM16], I decided to explore the possibility of using the conic to develop a RSA-like timelock puzzle. The secret sk can be encrypted by

$$c \equiv sk + a^{\odot 2^t} \pmod{n}.$$

Similarly to the RSA-TL, knowing the factorization of n, one can efficiently compute $a^{\odot 2^t}$ evaluating first $e \equiv 2^t \pmod{\Psi(n)}$ and then $a^e \pmod{n}$. Without knowing the factorization of n, one must perform t squarings with respect to the product $\odot$.

Finally, note that while it is certainly possible to build a time-lock puzzle based on the conic, from a cryptographic point of view a thorough analysis is needed, in particular regarding the security of the system. On the other hand, the use of a particular time-lock puzzle is not a prerequisite for the inner working of BTLE. Therefore we will treat the TLP parts as black-box.

11.2 Verifiable Delay Functions

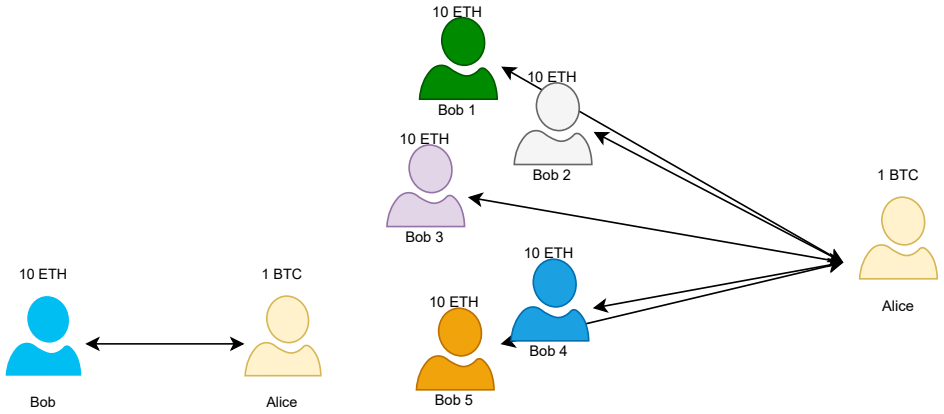
In the following I explain how it is possible to extend the timelock puzzles into a Verifiable Delay Function (VDF). To do that I apply the method described by Wesolowski [Wes20] to the BM-TL conic.

We say that a Verifiable Delay Function (VDF) implements a function $\mathcal{X} \to \mathcal{Y}$ if it is a tuple of three algorithms [BBBF18]:

- **Setup**$(\lambda, t) \to pp$ is a randomized algorithm that takes a security parameter λ and a time bound t, and outputs public parameters pp,
- **Eval**$(pp, x) \to (y, \pi)$ takes an input $x \in \mathcal{X}$ and outputs a $y \in \mathcal{Y}$ and a proof π.
- **Verify**$(pp, x, y, \pi) \to \{$ accept, reject $\}$ outputs accept if y is the correct evaluation of the VDF on input x.

The work in [Wes20] uses the **Setup** and **Eval** phases already described in [RSW96] and and Section 11.1.2. I are going to describe how to extend its **Verify** phase. There are two possible proof, one interactive and one non-interactive. I start by explaining the interactive one. If Primes(λ) is the set of prime numbers before λ, its steps are:

0. The verifier checks that $g, h \in \mathbb{G}$ and outputs reject if not,



(a) A 1-1 exchange (b) A many-1 exchange

Figure 11.1: In a 1-1 exchange there are only to parties involved. In a many-1 exchange, multiple parties want to exchange the same amount of funds. In this case a matching algorithm is used to choose Alice's partner. An example of a many-1 exchange are the online (crypto-)currency markets.

1. The verifier sends to the prover a random prime ℓ sampled uniformly from Primes(λ),
2. The prover computes $q, r \in \mathbb{Z}$ such that $2^t = q\ell + r$ with $0 \leq r < \ell$, and sends $\pi \leftarrow g^q$ to the verifier.
3. The verifier computes $r \leftarrow 2^t \pmod{\ell}$ and outputs accept if $\pi \in \mathbb{G}$ and $h = \pi^\ell g^r$ in G

This conclude the interactive proof. The non interactive proof uses the Rivest-Shamir heuristics in the random oracle model.

Since the creation and solution of $2^t \bmod n$ is the same for both the RSA-TL and BM-TL methods, the work by Wesolowski [Wes20] naturally extends the BM-TLP to a BM-VDF.

11.3 Time-based fund-exchanges between multiple blockchains

As I saw in Section 7.1.1, the underlying problem with decentralized methods is due to the impossibility result of a fair exchange. This result proved by Pagnia [PG99] states that in an asynchronous context, such as blockchain, it is impossible to have a secure and fair exchange without a trusted third party. The reason is that the trusted third party operates a synchronization work between the two parties. I already saw a way to introduce synchronization through HTLC in Section 5.2. In this Chapter we see a way to introduce synchronization thanks to time-lock puzzles or VDFs that do not require a trusted party, leaving the method completely decentralized, or peer-to-peer.

Similar to the work I did on MP-HTLC [BS22b] in Chapter 10, I structure the system in a way that lets multiple parties to operate an exchange. But differently from a "two groups setting", here I present a method that is more similar to a decentralized exchange, i.e. a "many-to-one" exchange, as in Figure 11.1b.

I do that creating a matching algorithm and an atomic swap algorithm based on time-lock puzzles. Both algorithms are part of the Broadcast Time-Lock Exchange Protocol (BTLE) and for simplicity I call BTLE-MA the matching algorithm and BTLE-AS the swap algorithm. Without loss of generality, I can say that the participant that initiates the process is selling its token in exchange of (or *to buy*) the other.

It is not necessary to use both the BTLE-MA and the BTLE-AS algorithms to obtain a token swap. In particular, it is possible to add to the current swap methods based on HTLC a matching algorithm based on BTLE-MA. This facilitates the partner discovery steps in current atomic swap systems. In the following, I assume a decentralized platform where participants want to swap tokens. Each participant owns a client that can track the progress of the reference blockchain. I informally call *view* the client's collection and ordering of states of the blockchain. This view is not necessarily the same for all participants due to network lag or possible attacks, such as an Eclipse attack that has

the effect of giving a false view of the blockchain blocks. Here "false" means that the victim's view is different from the view of the majority of blockchain miners/validators. In the following I will not focus on the consequences of this attack because it has no real consequences: if the victim has a different view of the blockchain, then it is very difficult for her to have transactions accepted by miners. The existence of different views explains why BTLE-MA is necessary: there cannot be a temporal-based ordering of possible buyers and an asynchronous systems imply the absence of a shared clocks.

I distinguish two classes of participants. The first is the class of *initiators*: this kind initiates the exchange by proposing the deal, i.e. the selling of its token. The initiator corresponds to a *market makers* in traditional centralized markets. The latter is the class of *exchangers*: to this class belong the possible buyers interested in an equivalent and opposite deal but that do not want to start it. These participants correspond to *market taker*.

I assume there is a single initiator which I denote by A (Alice), and many possible exchangers. Since the exchangers represent Alice's partner, following the cryptography tradition I will call them all "Bobs" and, supposing they are indexed and in finite number d, I denote them as $\{B_1, B_2, \ldots, B_d\}$ and I assume d secure channels of communication between A and each one of $B_i, i = 1 \ldots d$. I also assume $\{B_1, B_2, \ldots, B_d\}$ compete at the same price level, as in traditional order books. If not, then A narrows the view to the subset of Bobs with the most favorable exchange rate for A.

Note that the set $\{B_1, B_2, \ldots, B_d\}$ is completely determined by the view of A since I assume that A's view of the blockchain is the correct one. Consequently there may be other potential exchangers or some have withdrawn and A's view of potential buyers is not synchronized yet. In the protocol description I will see why neither of these two cases is a problem.

Finally, in the following I treat the time-lock puzzle TLP as a black-box which takes two inputs and then outputs a cryptographic puzzle, thanks to how I modeled the time-lock puzzle primitive in in Section 11.1. The solution of the puzzle is the cleartext itself. This shows that in theory the system can use any kind of time lock puzzle, including those described in Section 11.1. In Section 14.4 I will see which one is better from a practical point of view.

11.3.1 The Parties-Matching Engine

In this round Alice has to choose the exchanger among $\{B_1, B_2, \ldots, B_d\} \leftarrow GetExchangersList()$. Here $GetExchangersList()$ is a routine on the platform that returns the addresses of the exchangers and a way to communicate with them. How to implement it is outside the scope of this proposal.

BTLE-MA is explained in pseudo-code in Algorithm 6. As seen in the figure, A starts by generating a random message $rand_i, i = 1 \ldots d$ for each participant in $\{B_1, B_2, \ldots, B_d\}$. Then A associates this message to the intended receiver, $rand_i \leftrightarrow B_i, i = 1, \ldots, d$. The inputs of each time-lock puzzle are the message $rand_i$ and a time in seconds. Note that in a time-lock puzzle $TB_i = TLP(rand_i, time)$, the random message is different for each B_i, but $time$ is equal for all participants: all potential exchangers must have the same chance of being able to find the solution at the same time. The randomness of the winner is determined by unpredictable factors such as network latency or the puzzle real solving time. In this sense the choice of B_j is truly random. The output is a tuple that represent the cryptographic puzzle (See Section 11.1). A performs this subroutine for all $B_i, i = 1, \ldots, d$ and then it sends all the puzzles. Finally A waits for a solution.

When A receives the first solution (i.e. the cleartext of the random message) $\widehat{rand_j}$ from some B_j, A checks that it is a valid message. Practically, this means that A verifies that the cleartext $\widehat{rand_j}$ is equal to the message $rand_j$ associated with B_j. If that is the case, A accepts the solution and B_j is the winner: from now on A will proceed to communicate only with B_j and discards all other solutions. If the message isn't valid, A waits for another solution.

I stress the fact that the entirety of the BTLE-MA routine runs off-chain. Therefore it isn't costly nor slow. Currently, I am currently evaluating the advantages of such a system with respect to the traditional matching engines.

11.3.2 BTLE-AS

Let Bob be the winner of BTLE-MA and I denote it as B. This means that A will deal only with B, as in a classical token exchange. The goal of the parties in BTLE-AS is exchange their tokens in an atomic way.

Algorithm 6 BTLE-MA routine

$\{B_1, B_2, \ldots, B_d\} \leftarrow GetExchangersList()$
for $i = 1, \ldots, d$ **do**
    $rand_i \leftarrow RandGen()$
    $map(rand_i \leftrightarrow B_i)$
    $TB_i \leftarrow TLP(rand_i, time)$
end for
for $i = 1, \ldots, d$ **do**
    $Send : TB_i \longrightarrow B_i$
end for
$AcceptedSolution$ =`False`
while $AcceptedSolution$ =`False` **do**
    $waitSolution()$
    if $VerifySolution(sol)$ ==`True` **then**
        $AcceptedSolution$ =`True`
        $WinnerSolution = sol$
    end if
end while
$Winner = map(WinnerSolution)$
return $Winner$

In the following I present the notation used to understand the protocol. At the beginning of BTLE-AS, A owns 1 `coin1`, and B owns 1 `coin2`[1]. To enforce the atomicity of the protocol, I consider the ending of the BTLE-AS protocol as "successful" only if one of this two possible endings occurs:

1. A has 1 `coin2` B has 1 `coin1`
2. A has 1 `coin1` B has 1 `coin2`

Case 1. means that both A and B were honest during the execution of the protocol, while Case 2. happens if either A or B has been dishonest or faulty.

Intuitively BTLE-AS aims to exchange private keys between A and B so that they can move their newly acquired fund to other addresses. Of course, simply exchanging private keys would easily lead to possible theft: if A is not honest, A can still use his key to move his funds even if it sent the key to B, effectively stealing B's funds. Therefore I split a private key into two parts and I use the homomorphism of both the sum and external product in the elliptic curve group. Formally, given two secret keys sk_1, sk_2, an elliptic curve generator g and the relative public key pk_1 and pk_2, then the key sum is homomorphic:

$$(sk_1 + sk_2)g = pk_1 + pk_2 = (sk_1 g) + (sk_2 g). \tag{11.1}$$

Beside not sending the time-lock puzzle, A has another way to cheat B. In fact, A could create and send to B the time-lock puzzle of a random number instead of its private key sk_A. To avoid this problem, I introduce a zero-knowledge proof of the fact that the time-lock puzzle is hiding the right secret key, of course without revealing it.

In the following I introduce the swap and the proof protocols.

11.3.3 The Exchange Algorithm

I explain the exchange of tokens `coin1` and `coin2` with reference to Table 11.2, with notation in Table 11.1.

At first, A and B create the key pairs sk_2^A, pk_2^A and sk_1^B, pk_1^B respectively, where sk_i^j, pk_i^j are related to blockchain $BC_i, i = 1, 2$ and user $j = A, B$. These key pairs are respectively the first of the two

[1]The real exchange rates between the two tokens and how they are decided are beyond the scope of this proposal. I assume this kind of information can be exchanged either in the channel during BTLE-MA or the UI implementation of the protocol.

BC_1,BC_2	Blockchain 1 and 2 with tokens `coin1` and `coin2`
G_1,G_2	the base point for the elliptic curve of BC_1 and BC_2
l_1,l_2	the base point order for the elliptic curve of BC_1 and BC_2
PK_1^A	public key for the address on blockchain BC_1 where A has the coins
PK_1^B	public key for the address on blockchain BC_1 where B receives the new coins
PK_2^A	public key for the address on blockchain BC_2 where A receives the new coins
PK_2^B	public key for the address on blockchain BC_2 where B has the coins
$sk_{1,2}^A, pk_{1,2}^A$	shares created by A for blockchain $BC_{1,2}$
$sk_{1,2}^B, pk_{1,2}^B$	shares created by B for blockchain $BC_{1,2}$

Table 11.1: Notation used in the explanation of the token exchange protocol

shares of A and B to redeem the exchanged funds. After this step, A and B exchange the public keys pk_2^A and pk_1^B. If this first part is successful, both A and B create ephemeral keys (sk_1^A, pk_1^A and sk_2^B, pk_2^B respectively) that represent the second of the two shares to redeem the exchanged funds. At this point A and B can create new public keys (called PK_1^B and PK_2^A respectively) to which they can send the funds.

This transaction is a $(\frac{\lambda}{2}, t)$-conditional transaction as explained in Section 9.2 where t is a generic deadline the participants A and B can agree on or can be included in a specification.

Because of the way the keys PK_1^B and PK_2^A and consequently the addresses are built, neither of the two participants can redeem the funds in either blockchains at this point of the exchange. For example, A needs to know sk_2^B in order to redeem coins in the address for PK_2^A.

For this reason in the second part of the exchange A and B exchange the time-lock puzzles TLP_A and TLP_B and their respective proofs, explained in Section 11.3.3 . Once the time-lock puzzles are opened/solved, A gets sk_2^B and B gets sk_1^A. Using λ as time unit, we see that the second time-lock puzzle is sent later with a smaller opening time (1/4 of the time unit). This is to make the two participants A and B open the puzzle at about the same time.

The Proof

In the previous section we saw how the exchange works, and I treated the proof as a black-box. In this section I explain in details how the proof works. Here I explain how to do this assuming that the users will use one of the two time-lock puzzles described in Section 11.1. In particular I assume that for each protocol there exist two functions **ProofGen**() and **ProofVer**() such that:

- **ProofGen**$(sk, rand)$ is a deterministic algorithm that, given a secret key sk and a random message $rand$, creates a proof π
- **ProofVer**(π, pk) is a deterministic algorithm that, given the public key pk with respect to sk and the proof π, outputs 1 if π is valid and 0 otherwise

I start with the explanation of **ProofGen**. In case of the RSW-TL and the BM-TL time-lock puzzles, c is obtained from the formula

$$c = sk + a^{2^t}, \quad c = sk + a^{\odot 2^t}$$

respectively, where sk is the message (the secret key) the user wants to encapsulate in the puzzle, a a random number and t is the number of squarings.

To generate the proof, we recall that a digital signature can be seen as a zero-knowledge proof of knowledge of a private/secret key and that c can be seen as an ephemeral private key. In this case, let **SigGen**(m, s) be the signature routine of a digital signature scheme such that given a message m and a secret key s it outputs a signature σ, and let **SigVer**(m, σ, S) the respective verification algorithm such that given a message m, a public key S and a signature σ outputs `True` if σ is the signature of message m, and `False` otherwise. Finally let P a prover and V a verifier, similar to every zero-knowledge proof protocol. Then the protocol works in the following way and the case of the RSW-TL can be seen in Algorithm 7.

Alice (`coin1`→`coin2`)		Bob (`coin2`→`coin1`)
$sk_2^A \xleftarrow{\$} [1, l_2 - 1], pk_2^A = sk_2^A G_2$		$sk_1^B \xleftarrow{\$} [1, l_1 - 1], pk_1^B = sk_2^B G_1$
$rm_A \leftarrow GenRandomMessage()$		$rm_B \leftarrow GenRandomMessage()$
	$\xrightarrow{pk_2^A, rm_A}$	
	$\xleftarrow{pk_1^B, rm_B}$	
$sk_1^A \xleftarrow{\$} [1, l_1 - 1], pk_1^A = sk_1^A G_1$		$sk_2^B \xleftarrow{\$} [1, l_2 - 1], pk_2^B = sk_1^B G_2$
$PK_1^B = pk_1^A + pk_1^B$		$PK_2^A = pk_2^A + pk_2^B$
$hash_{A\to B} \leftarrow$SendCondTx$(\frac{\lambda}{2}, t, PK_1^A \to PK_1^B)$		$hash_{B\to A} \leftarrow$SendCondTx$(\frac{\lambda}{2}, t, PK_2^B \to PK_2^A)$
		$T_B \leftarrow TLP_B(sk_2^B, \lambda)$
		$\pi_B \leftarrow \textbf{ProofGen}(sk_2^B, rm_B)$
	$\xleftarrow{T_B, hash_{B\to A}, \pi_B}$	
$T_A \leftarrow TLP_A(sk_1^A, \frac{3}{4}\lambda)$		
$\pi_A \leftarrow \textbf{ProofGen}(sk_1^A, rm_B)$		
	$\xrightarrow{T_A, hash_{A\to B}, \pi_A}$	
$\textbf{ProofVer}(rm_B, pk_1^B, \pi_B)$		$\textbf{ProofVer}(rm_A, pk_2^A, \pi_A)$
open T_B and redeem `coin2`		open T_A and redeem `coin1`

Table 11.2: Protocol execution between Alice and Bob for a successful swap

Algorithm 7 ProofGen(sk, rm)

1: $rm \leftarrow GetRandomMessage()$
2: $c = sk + a^{2^t}$
3: $A \leftarrow$ **PubkeyGen**(a^{2^t})
4: $\sigma \leftarrow$ **SigGen**(rm, c)
5: $\pi = (\sigma, A)$
6: $Send(\pi)$

During the exchange, the prover P has to send the public key pk related to sk to the verifier V. V acknowledge the reception of the public key by sending a random message rm. Note that the public key is necessary to route the funds to the address, so P would have sent it anyway.

Assume there is an algorithm $X =$ **PubkeyGen**(x) that given a number x considers x as a ephemeral secret key and outputs the relative ephemeral public key X. When P creates the time-lock puzzle c, P also computes the public key $A =$ **PubkeyGen**(a^{2^t}) RSW-TL or $A =$ **PubkeyGen**$(a^{\odot 2^t})$ in case of BM-TL. Then P computes $\sigma =$ **Sig**(rm, c), a signature of the random message rm using the time-lock puzzle result c acting as ephemeral secret key. The proof is $\pi = (\sigma, A)$. This concludes the **ProofGen** routine.

Upon receiving the time-lock puzzle an the proof π, V starts the **ProofVer** routine as seen in Algorithm 8 . V first checks that the proof works. To do that, V uses the public key $pk + A$ to verify that the message signed in σ is rm (recall that V already has pk as per the swap algorithm) using the **SigVer**$(\sigma, pk + A)$ routine. This works because $c = sk + a^{2^t}$, pk is the public key related to sk, $A =$ **PubkeyGen**(a^{2^t}) by construction and for the property of homomorphism of the sum in elliptic curve cryptography. If the verification checks, then V is sure P's time-lock puzzle is correctly built and then proceeds in solving it.

Algorithm 8 ProofVer(rm, pk, π)

1: $\sigma = \pi[0]; A = \pi[1]$
2: $Pubkey = pk + A$
3: $Bool \leftarrow$ **SigVer**$(rm, \sigma, Pubkey)$
4: **if** $Bool ==$ `True` **then**
5: *Accept* π
6: **else**
7: *Reject* π
8: **end if**

11.4 Implementation details

In this section, I analyze the exchange protocol relative to three points of view. The first is that of security, then I will look at the protocol from a fee point of view and finally I will look at the speed of the protocols. I conclude that the best time-lock puzzle protocol for this purpose is the time-lock puzzle based on the Bellini-Murru protocol (BM-TL).

Both the time-lock puzzles are based on already analyzed methods. In particular, both the RSW and BM method derive their security from the difficult problem of number factorization.

BTLE is competitive with the classic atomic swap in terms of fees. This is because no on-chain operations are required: the exchange of random messages in BTLE-MA, the exchange of private key shares in BTLE-AS and the computation needed for the generation an resolution of the time-lock puzzles are done off-chain. The only transaction per blockchain is the transaction that participants make to send funds to another address. This is an advantage over the number of transactions required in an HTLC.

Another advantage of BTLE over HTLC is that the timelock puzzle system is more flexible. While in an HTLC it is necessary to decide the waiting time between transactions as a function of the timing of the creation of blocks in both blockchains, in BTLE the timing can be shorter since BTLE does not have to do with blocks (there are no transactions sent other than the funding transaction if both

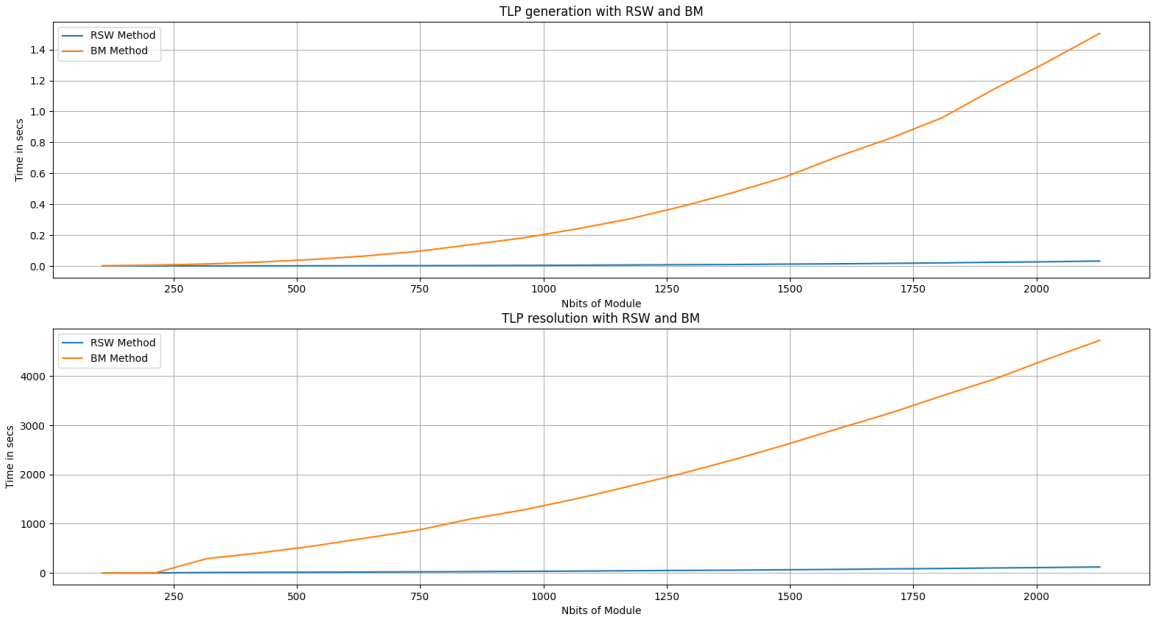


Figure 11.2: Comparison of generation (above) and resolution (below) times of the time-lock puzzles in the RSW-TL and BM-TL cases. As seen in figure, for a scenario like the one presented in this article, on a practical level only BM-TL can be used.

parties are honest), but the only delays are related to the latency of the internet network, which are obviously much shorter

I conclude this section with advice on choosing a time-lock puzzles between RSW-TL and BM-TL based on the tests I conducted. As seen in the comparison in Figure 11.2, using a squaring number $t = 10^7$ the BM-TL takes much longer than the RSW-TL method. In particular it can be seen how with a number N such that its length in bits is about 2000 bits, the resolution time of the time-lock puzzle is $\tilde{1}00$ seconds in the case of RSW-TL , while it exceeds 4000 seconds (just over an hour) in the BM-TL case. Figure 11.3 highlights this result. Furthermore, the growth in terms of seconds in both generation and resolution of the time-lock puzzle is not linear in terms of the number of bits in the modulo N.

This makes it more practical to use BM-TL as fewer total squarings are required. If participants want to wait less than an hour, they can use a N module of less than 2000 bits or decrease t. Test results can be found in the GitHub repository[2]. There, it is possible to find the results of the tests made with both BM-TL and RSW-TL, on a core of different machines (Apple M1, AMD Series 7, Intel Xeon Skylake IBRS),

11.5 Conclusions

In this chapter we presented BTLE, a two-phases protocol for obtaining atomic swaps between two blockchains based on time-lock puzzles instead of HTLC which (differently from HTLC) can be used in both a 1-1 swap and a many-1 swap as found in e.g. online exchanges. In the Chapter we have proved the feasibility of the protocol and we proposed an implementation of BTLE that can be found online. We also tested BTLE using the two types of time-lock puzzles, namely RSW-TL and the one proposed by us which we called BM-TL and published the results.

[2]See files whose name starts with `generation − values − from − test` at https://github.com/disnocen/dex-tlp

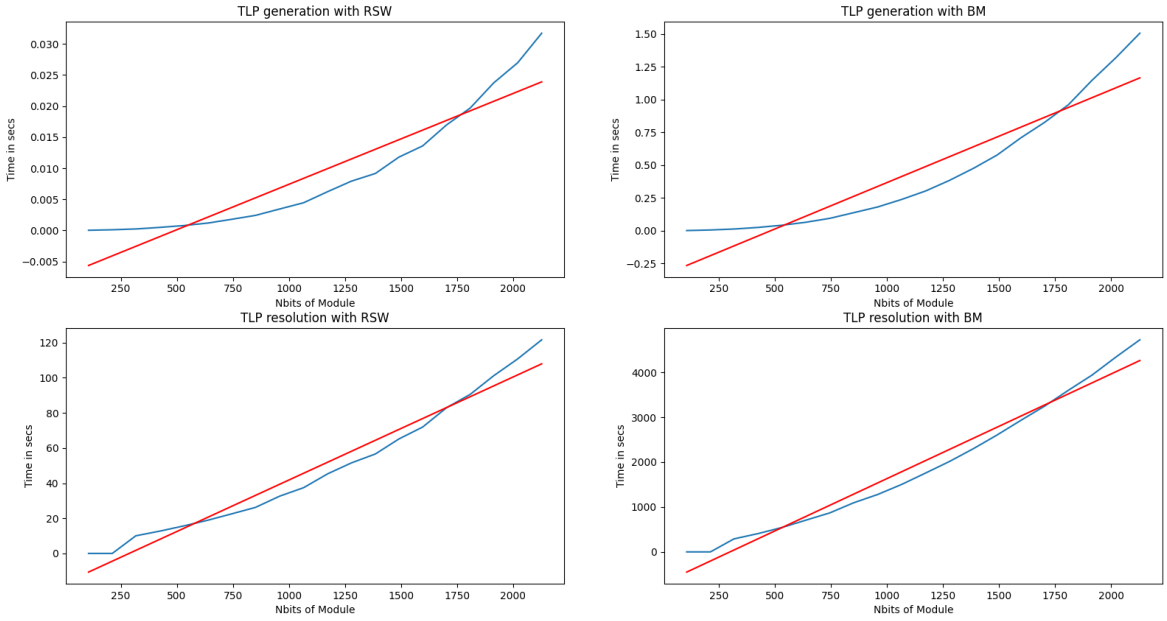


Figure 11.3: Comparison of generation (above) and resolution (below) times of the RSW (left) and BM (right) time-lock puzzles. It is clear that the growth in terms of seconds with respect to the magnitude in bits of the modulo N is not linear. I also note that the times in the BM-TL case are about 30 times greater than the times in RSW-TL.

Chapter 12

Wrapped Token Exchange

It is well known that it is relatively easy to exchange tokens whose smart contracts are on the same blockchain. In contrast, cross-exchanging bitcoins for a Bitcoin wrapped token is still cumbersome since we are talking of a multi-blockchain exchange. As seen in Section 7.3, current methods of exchange are still custodial and perform privacy-threatening controls on the users in order to operate.

In previous chapters I dealt with general purpose peer-to-peer exchanges. In MP-HTLC [BS22b] I extended HTLCs (Section 5.2) to mitigate some implementation problems (see Section 5.3.2) and in BTLE [BMS21] I moved part of the exchange off-chain to deal with the rest of the problems (see Chapters 10 and 11 respectively to see how these protocols work). Theoretically, both protocols could be used to mitigate the wrapped-token exchange problem. Yet, both protocols rely on time-locks (and therefore *time* and *waiting periods*) to provide security. My research question is: is it possible to have a better user experience in some particular cases?

To answer this question, in this chapter I present Bitcoin-Cross-Tokenized-Bitcoin (BxTB [BS22a], pronounced *B-cross-TB*) which deals with cross-chain exchanges of bitcoins for any Bitcoin wrapped tokens on a EVM-compatible blockchain. BxTB lets users achieve a decentralized and private wrapped-token exchange by bypassing the mint-and-burn paradigm of current wrapped tokens and cross-exchanging already minted tokens in a P2P way. Instead of relaying on HTLCs or on the overhead of communication and slowness due to time-locks, here I use Stateless SPVs, i.e. proof-of-inclusion of transactions in the Bitcoin chain validated through a smart contract deployed on another EVM-compatible blockchain.

Using BxTB a user A can exchange its bitcoins with a user B in exchange for a Bitcoin wrapped token. B in this way can come into possession of "real" bitcoins without interacting with the custodian of the service.

Although, in theory, the method I am presenting can be used to exchange any token for bitcoin, in cases other than a wrapped-token the protocol must take into account several different assumptions. For example, users A and B must decide an exchange-rate between token and bitcoin, modifying the protocol and requiring an introductory step. A hint of how this can be achieved is presented in the Future Works (see Section 12.3).

Contribution: My contribution related to BxTB can be summarized in the following list:

- I present a way to exchange wrapped token without using a mint strategy and consequently bypass privacy-threatening controls using Stateless SPVs
- I give a formal explanation of the primitive Stateless SPV: to date the primitive is informally described and there is no formal discussion on the subject
- I give a proof of the security of the primitive, which can be of independent interest.

Organization: We assume the reader has already read Sections 7.3 and Section 7.5 which are fundamental to understand this chapter. In Section 7.4 I give a brief presentation of the stateful SPVs used by light clients and present more details on how *Stateless* SPVs work. In Section 12.1 I present how the BxTB works: I will delineate how the exchange between participants works and how to prove the validity of stateless SPV proofs. In Section 12.2 I analyze the BxTB protocol: in particular, I will prove in Theorem 12.2.2 that it is not economically viable to forge the proof if it is based on enough blocks,

similar to how normally a transaction is considered "confirmed" in a proof of work blockchain. I also explain the advantages between BxTB and HTLC, commonly used in atomic swaps. In Section 12.3 I describe how I plan to continue the development of BxTB. Then I conclude.

12.1 Design

12.1.1 Setting and Notation

We assume the existence of a decentralized-application platform PLAT which acts as a match-making algorithm between parties and provides a secure channel between parties[1]. The goal of PLAT is to match parties who want to exchange bitcoins for a wrapped equivalent on a Turing-complete blockchain, I will call *Tchain*. I call this exemplary wrapped token *wrBTC* and I assume it is build on the model of those explained in Section 7.3.

After the match, PLAT interacts with a smart contract SC deployed on Tchain. The smart contract's pseudocode can be seen in Algorithm 9. Note that there is no explanation of how the matching works, since this is outside the scope of the protocol: I assume the match has been done (for example by using a decentralized method such as the one presented in Section 11.3.1) and I describe how parties interact to complete the protocol.

We also assume two parties. Alice, A, that owns bitcoins and wants to exchange them for wrBTC on Tchain. Bob, B, that owns wrBTC and wants to exchange it for bitcoins. For simplicity, assume the exchange amount is amt, the timeout on the exchange is T and the number of blocks to build the stateless SPV proof π is n_{blocks} (See Equation (12.3) for a practical way to decide on n_{blocks}). I assume that parties agree on those parameters during the match-making phase, non-necessarily without suggestions or defaults provided by the platform PLAT. Note that there is no need to agree on an exchange rate since the currencies are supposedly pegged. I briefly mention a way to exchange non-pegged tokens in Section 12.3.

We denote the addresses of parties in the following way. Since both parties needs to have two addresses (one on Bitcoin and one on Tchain), I denote the addresses of Alice and Bob on Bitcoin as $AddrA_{\text{BTC}}$ and $AddrB_{\text{BTC}}$ respectively and on Tchan as $AddrA_{\text{TCH}}$ and $AddrB_{\text{TCH}}$ respectively. The exchange of meaningful data between A and B, e.g. addresses and amount, can be public or exchanged privately in the channel. From a practical point of view this does not change anything privacy-wise: all information becomes public once the smart contract is used by the parties.

12.1.2 The Swap

We assume that B initialize the smart contract SC by calling the PREPROCESS (line 1) and the INIT functions (line 14). The latter function put set the State equal to Init.

After the initialization, B is required to put amt wrBTC in the smart contract (line 20) to advance the protocol[2]. If B does that before the timeout, the State of SC is set to ExpectProof. Note that A has not provided any bitcoin at this point: even if B fails this step, A does not lose anything.

If B behaves honestly, A broadcasts a transaction from $AddrA_{\text{BTC}}$ to $AddrB_{\text{BTC}}$ on the Bitcoin blockchain. After waiting for the creation of n_{blocks} blocks on the Bitcoin blockchain, A builds the proof π. After that, A calls SC's function RECEIVEPROOF (line 28) from $AddrA_{\text{TCH}}$: one of the inputs of the call is the the proof π. If A provides a correct π before the timeout (see Section 12.1.3 and Algorithm 10 to see how the verification works), the State of SC is set to Redeem (see Figure 12.1 to see how States of SC change) and A can redeem the wrBTC (line 39). On the other hand, if A fails to provide a correct π the state of SC remains ExpectProof: after the timeout, B can redeem the wrBTC (line 44).

The protocol is atomic (see Definition 7.5.1) and I show that in Section 12.2.1.

[1] This can be implemented with end-to-end encrypted communication so that PLAT never knows anything more about the exchange,see e.g. [ACD19]

[2] Note that the PREPROCESS, INIT and RECEIVEWRBTC can in principle be collapsed into only one function: B would certainly save some fees doing that. I decided to keep this functions separated in Algorithm 9 for two reasons. The first one is that keeping the functions separated help the understanding of the protocol. The second reason is that this way the platform PLAT could theoretically initialize SC in place of B, possibly as promotion to make users save on fees.

Algorithm 9 PLAT's smart contract SC

Require: $AddrA_{\text{BTC}}$, $AddrB_{\text{BTC}}$ and $AddrA_{\text{TCH}}$, $AddrB_{\text{TCH}}$ are valid addresses
Require: amount amt is a positive integer
Require: timeout T is a positive integer

function PREPROCESS($AddrA_{\text{BTC}}$, $AddrB_{\text{BTC}}$, $AddrA_{\text{TCH}}$, $AddrB_{\text{TCH}}$, amt, T , L)
    Struct Exchange = {
        AddrABTC = $AddrA_{\text{BTC}}$,
        AddrBBTC = $AddrB_{\text{BTC}}$,
        AddrATCH = $AddrA_{\text{TCH}}$,
        AddrBTCH = $AddrB_{\text{TCH}}$,
        amt = amt,
        T = T
        L_{max} = L ▷ L_{max} is the maximum target accepted as difficulty in headers
    }
    States = { Init, ExpectPayment, ExpectProof, Redeemable, Success, Fail }
    setState(sid,Init)
end function
function INIT(sid, Exchange)
    if State == Init **then**
        map sid to Exchange
        setState(sid,ExpectPayment)
    end if
end function
function RECEIVEWRBTC(sid, Exchange)
    if `msg.sender` == AddrBTCH &
    `msg.value` == amt &
    State == ExpectPayment **then**
        Event(Exchange, sid,received, {AddrBTCH, amt})
        setState(sid,ExpectProof)
    end if
end function
function RECEIVEPROOF(sid, Exchange, π)
    if `msg.sender` == AddrATCH & time < T &
     Verify(π, Exchange)== True & State == Init **then** ▷ Verification is explained Section 12.1.3
        send(AddrATCH, amt)
        Event(Exchange, sid,sent, {AddrATCH, amt})
        setState(sid,Redeemable)
        REDEEMWRBTC(sid, Exchange)
    end if
end function
function REDEEMWRBTC(sid, Exchange)
    if `msg.sender` == AddrATCH &
     time < T & State == Redeemable **then**
        send(AddrATCH, amt)
        Event(Exchange, sid,sent, {AddrATCH, amt})
        setState(sid,Success)
    else if `msg.sender` == AddrBTCH &
time > T & State == ExpectProof **then**
        send(AddrBTCH, amt)
        Event(Exchange, sid,withd, {AddrBTCH, amt})
        setState(sid,Fail)
    end if
end function

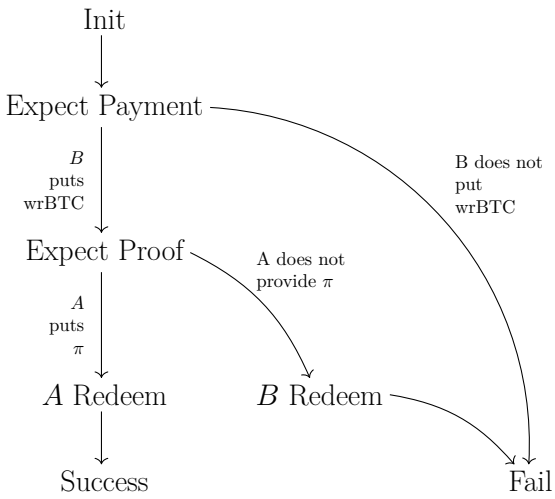


Figure 12.1: State succession of Algorithm 9

12.1.3 Proof Validation

We are left to explain how the stateless SPV proof π is validated. The verification exploits the fact that it is highly expensive to produce a formally valid proof in the dishonest case: see Theorem 12.2.2 in Section 12.2.2.

The first thing to check is if the transaction tx is included in the Merkle tree of block $\mathcal{B}_N$. It is easy to build the vector vec_N (see Equation (7.3)) for A which has access to the Bitcoin blockchain. The Merkle root hash is included in the header $\mathcal{H}_N$, see Table 7.3, and to perform this check the function only needs the correct hash function.

If tx is included in the Merkle tree, then the vec_N is valid and it is possible to proceed and check if the subsequent headers are built one upon the other with a sufficient difficulty. With reference to Algorithm 10, this is performed from line 7 through 15.

12.2 Analysis

12.2.1 Atomicity

In this section I prove that the protocol is atomic. Formally:

Definition 12.2.1 *the protocol designed as in Algorithm 9 and Algorithm 10 is* correct *if two parties A and B such that:*

- *A has x bitcoin and wants x wrBTC, while B has x wrBTC and wants x bitcoin,*
- *A and B execute all the steps honestly*

obtain x wrBTC and x bitcoins respectively by following the protocol.

I can now prove:

Theorem 12.2.1 *Let A and B two parties who want to exchange bitcoins for a bitcoin wrapped token* wrBTC*. Then the protocol designed as in Algorithm 9 and Algorithm 10 is correct, as defined in Definition 12.2.1, and atomic in the sense of Definition 7.5.1*

Proof 12.2.1 *Correctness is obvious once looking at the algorithms and at Figure 12.1.*

To prove that the protocol is atomic, I need to show that no party would incur in financial loss in case of abort or incorrect execution. I do that with reference to Figure 12.1.

Of course if no initialization is done (Init), then neither A nor B incur in financial loss since there is no deployment of capital. After that, the smart contract expects a payment from B (ExpectPayment). Assuming incorrect execution, at this step, B does not put any wrBTC in the smart contract. Similar to the case of the initialization, A does not incur in financial loss since A has not deployed capital yet.

Algorithm 10 Verification procedure of SC

Require: $\pi = [vec_N, \mathcal{H}_N, \mathcal{H}_{N+1}, \ldots, \mathcal{H}_{N+n_{blocks}}]$ ▷ Equation (7.4)
Require: $\pi.vec_N = [tx, H_i, H_{jk}, \ldots]$ ▷ Equation (7.3)
Require: Exchange as defined in Algorithm 9
Require: $H(\cdot)$ hashing function of Bitcoin
1: **function** VERIFY(π,Exchange)
2: vec = $\pi.vec_N$
3: VerifyMerkleTree(vec,$\mathcal{H}_N.hashMerkleRoot$)
4: $\mathcal{H}_{prev} = \pi.\mathcal{H}_N$
5: L_{max} = Exchange.L_{max}
6: assert $\mathcal{H}_{prev}.nBits \leq \mathrm{L}_{max}$ ▷ Check that the target is not too large
7: **for** $i = 1, \ldots, n_{blocks}$ **do**
8: $\mathcal{H}_{cur} = \pi.\mathcal{H}_{N+i}$
9: assert $\mathcal{H}_{cur}.nBits \leq \mathrm{L}_{max}$
10: **if** $H(\mathcal{H}_{prev}) == \mathcal{H}_{cur}.hashPrevBlock$ **then**
11: $\mathcal{H}_{prev} = \mathcal{H}_{cur}$
12: **else**
13: **return** False
14: **end if**
15: **end for**
16: **return** True
17: **end function**

On the other hand, if B does put the right amount of wrBTC in SC, then A is expected to make a payment from AddrA_{BTC} to AddrB_{BTC} on the Bitcoin blockchain, then build and send the proof π to SC. In this case, if A does not make a payment before the timeout or if A provides an incorrect proof (we show in Theorem 12.2.2 the sufficient condition for B to be sure A will not be able to create an invalid proof which is accepted), then B does not incur in financial loss since B will be able to redeem the wrBTC from SC by calling the REDEEMWRBTC *function.*

Finally, if A and B follow the protocol correctly, then A can redeem the wrBTC on Tchain by putting π in the RECEIVEPROOF *function, since the* REDEEMWRBTC *function is called from* RECEIVEPROOF.

12.2.2 Security

We want to prove that it is economically secure to validate a transaction with no stored block header but only with the n_{blocks} headers provided by the user (which of course can potentially be forged by it). By *economically secure* I mean that the cost of forging a proof is higher than the amount of the transaction the user needs to prove the inclusion of. Formally:

Definition 12.2.2 *A proof π defined as Equation (7.4) of transaction tx with amount amt is* (n_{blocks}, p^*)-economically secure *if the expected cost of having a probability of p^* of producing π is higher than amt.*

To prove it, I will leverage the *nBits* field of the block header (see Table 7.3). Also assume that the difficulty/target do not change between block N and $N + n_{blocks}$. This is not a big constrain since the difficulty changes once every 2016 blocks in Bitcoin and n_{blocks} is generally less than 10 (note that a Bitcoin transaction is generally considered confirmed after 6 blocks, but see Table 8.1 for a detailed analysis of probability of double-spends). Furthermore, it will be easy to see that *if* the difficulty changes between block N and $N + n_{blocks}$, then taking $\max(\mathcal{H}_N.nBits, \mathcal{H}_{N+n_{blocks}}.nBits)$ as difficulty solves the problem.

We briefly formalize the leader election phase of the Nakamoto consensus protocol which is based on the HashCash system [Bac02]: Given a target L, the *work* of a miner is to find the nonce *nonce* for a block $\mathcal{B}$ such that[3] $H(\mathcal{B}) < L$, where H is the hash function used by the system, SHA256 in the Bitcoin case. In this case we say that *nonce* is a valid nonce for block $\mathcal{B}$. Valid outputs for the hash function used in Bitcoin range from 0 to $2^{256} - 1$, but L is much less than that. So the number

[3]The nonce is part of the block header and therefore part of the block, for this reason it does not appear explicitly, see Table 7.3.

of available hashes for block $\mathcal{B}$ is L and the probability of finding one is $p = L/2^{256}$. "Extracting" hashes is therefore a Bernoulli process of probability (and mean) p. To give an idea with real values, on May 25, 2022, the difficulty is $d = 31251101365711$ [4]. The target L can be computed from the difficulty by doing $L = 2^{224}/d$ (the exact computation to arrive at this relation between L and d can be seen in [BKKT20]). Therefore $p \approx 7 \times 10^{-24}$. The probability of 1 success in n Bernoulli trials is $p^* = np(1-p)^{n-1}$. For example, with the values presented above, we need $n = 10^{26}$ trials to be $\approx 60\%$ sure of finding a valid nonce. Consequently, assuming C to be the cost of computing a *terahash* (i.e. 10^{12} trials), being 60% sure of producing one single header (i.e finding a nonce) has an expected cost $EC_1^{0.6} = 10^{-12} \cdot n \cdot C$.

We can now prove that:

Theorem 12.2.2 *Given a target $\mathcal{H}.nBits = L$, let $p = L/2^{256}$ and p^* be the probability of success in finding a suitable hash for one header $\mathcal{H}$ after n_{p^*} trials. Assuming C to be the cost of computing a terahash, then the expected cost $EC^{p^*}_{n_{blocks}}$ in being p^* sure of finding n_{blocks} nonces for n_{blocks} headers is:*

$$EC^{p^*}_{n_{blocks}} = n_{blocks} \cdot 10^{-12} \cdot n_{p^*} \cdot C \tag{12.1}$$

Then a sufficient condition to for a protocol between rational participants to achieve (n_{blocks}, p^*)-economic security for a proof π of transaction tx with amount amt is

$$EC^{p^*}_{n_{blocks}} \geq amt \tag{12.2}$$

Proof 12.2.2 *It is easy to compute $EC^{p^*}_{n_{blocks}}$ of Equation (12.1) noting that all trials are independent events. To see the sufficiency of condition in Equation (12.2), first note that generally the cost of producing hashes is shared by all the miners in the honest case and therefore comes at a cost of ≈ 0 for the honest user (it only has to pay the fees of transaction tx)*

On the other hand, if the user is dishonest and needs to forge π, then it incurs a potential loss: it has to bear the expected cost $EC^{p^*}_{n_{blocks}}$ without being completely sure of finding a solution within a certain timeout, since there is still a probability of $1-p^*$ of not *producing the proof even after incurring in the $EC^{p^*}_{n_{blocks}}$ cost*[5]*. Therefore, the user has no interest in forging a proof π in the case of Equation (12.2) since it is expected to lose more than it has to gain.*

Since Equation (12.2) is not operational, Equation (12.3) gives a practical way for participant B, or Plat, to enforce economic security of the protocol for any amount *amt* of every transaction:

$$n_{blocks} \geq \frac{amt}{10^{-12} \cdot n_{p^*} \cdot C} \tag{12.3}$$

12.2.3 Privacy and Communication

BxTB is as private as the HTLC described in Section 5.2. In HTLCs in particular the hash created by the initiator A acts a indexing of the transactions in Bitcoin and in the other chain used for the swap. Using this index it is possible to learn A and B's addresses in both blockchains and the amount exchanged. These are the same information an external observer and the platform Plat can gather by looking at the smart contract SC.

On the other hand, BxTB achieves the same results the HTLC achieves but with less transactions, even in the worst case. In fact, BxTB requires two transactions in the worst case instead of four as in the HTLC case: One transaction from B to SC to initialize the smart contract and deposit the tokens; One transaction from B to SC to withdraw the funds after the timeout. The worst case is considered to be the case where A does not follow the protocol, since if B does not follow the protocol then no transaction is being done by definition (B does not initialize the smart contract and/or does not deposit the tokens in SC).

[4] See https://bitinfocharts.com/comparison/bitcoin-difficulty.html

[5] It is possible that $EC^{p^*}_{n_{blocks}}$ and *amt* do not share the same currency to be evaluated. In this case assume *amt* is evaluated using the currency $EC^{p^*}_{n_{blocks}}$ is evaluated.

12.3 Future Works and Perspectives

As we mentioned before, BxTB can in principle work for any couple of tokens, from a Proof-of-Work blockchain to a wrapped-token on any Turing complete blockchain. In fact, assuming the hash function(s) used for the leader election is implemented in the smart-contract language of Tchain, the basic checks are trivial to make and shared by all Proof-of-Work blockchains. Some changes to the protocol are needed: for example, if the tokens are not pegged, then the parties need to agree on the rate of the exchange as in a general atomic swap (see Section 7.5). Technically this rate can be decided while posting the order on PLAT, but some considerations on privacy and communication-complexities should be studied.

BxTB can also be extended by forcing A to put some collateral. In the current design of BxTB, A can potentially abort the protocol after the initialization or intentionally put an invalid proof. While that would not hurt B financially (assuming n_{blocks} has been chosen in accordance to Equation (12.3)), it is still a problem since B's capital is blocked for a certain amount of time. This is a common problem in atomic swaps and it is present in current implementations of HTLC too (see Section 5.3). The collateral would act as an incentive for A to continue the protocol, since it can be slashed in cases where A puts an invalid proof or simply does not follow through the protocol.

12.4 Conclusions

We presented BxTB, a new protocol to exchange bitcoin wrapped tokens without incurring in privacy threatening controls. I achieved exchanges leveraging stateless SPVs, practically describing how to create a light client in a smart contract. I proved the economic security of the protocol and I achieve atomicity using less transactions between parties even in the worst case when one party is dishonest.

Part III

Blockchain Applications to Commerce

Chapter 13

Anonymous Blockchain-enabled Physical Products Deliveries

We dealt with the blockchain integration in commerce and its interoperability requirements in Chapter 6. We saw that the majority of product-delivery systems are custodial, interactive (especially in case of dishonest parties), not privacy oriented and rely on extensive scripting capabilities. This last requirement precludes the development of these systems on many blockchain projects, such as Bitcoin, Zcash and Monero.

Furthermore, the basic assumption of the protocols is that delivery is done by humans, while in the future the majority of deliveries will be done by non-humans devices, e.g. drones[1]. In those cases, the delivery itself has to as much non-interactive as possible, while still maintaining a tackability and enforcing non-deniability of receipt. Such a system would also respect the security model of the participant, as explained in Section 6.5.3

To solve those problems, I present Pay-to-Transport (P2T) [BS20b]. P2T is a protocol that lets customers buy an item remotely in an atomic, privacy preserving and trustless manner. P2T needs only basic features of a blockchain scripting language and does not need any tracking systems, arbitrator or deposit to preserve its security properties. For this reason P2T can be implemented on any permissionless blockchain, regardless of its scripting language, without additional trust. Merchants' and transporters' addresses are public, but in P2T the parties never pay those addresses directly. Therefore P2T maintains the privacy of customers, merchant and transporters.

13.1 Delivery tracking via Atomic Payments

The P2T protocol involves three parties: a merchant M, a customer C and a transporter T. Public keys $\mathbf{pk}_M$ and $\mathbf{pk}_T$ and blockchain addresses P_M and P_T of M and T respectively are public and known in advance to all the parties (e.g. those information are in the contact internet pages of the parties). Note that a protocol for the shipment of a product from M to C is different from a protocol for returning that product *after* the acceptance of the delivered package: in this proposal I focus only on the shipment protocol. Therefore the shipped package can be accepted or refused by C on the spot only. Of course, it is possible to adapt this protocol in case of a product return, treating it as a shipment from C to M, but this is not discussed here.

Broadly speaking, the P2T protocol works as follows. The transporter T goes to merchant M, pays M the cost of the item (ad interim payment) and takes charge of the package. The transporter then brings it to his own pick up point (e.g. T's company headquarters). Finally, customer C goes to T's pick up point, pays T the cost of the item plus transportation costs and takes his package. C does not have to reveal his own physical address nor his identity to perform these actions.

Since a system of "simple" transactions (e.g. P2PKH in Bitcoin or transactions between Externally Owned Accounts in Ethereum) would not give sufficient guarantees to any of the parties involved, I based every passage of coins and product on time constraints and spend ability conditions (see

[1] See for example this video on how packages are currently in some parts of the world: https://youtu.be/qcsszdkjlXg?t=18

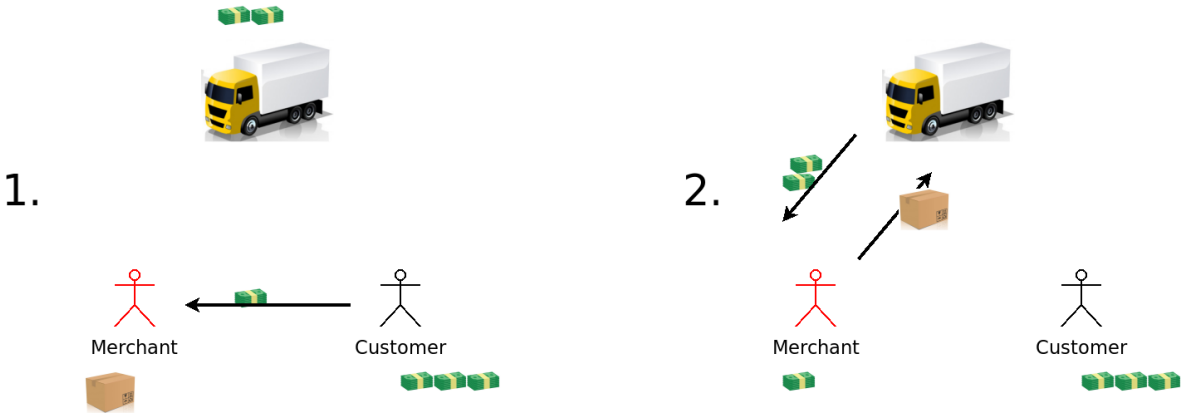


Figure 13.1: Phases one and two of the Basic Protocol with One Transporter. **1.** C pays M the transportation costs, **2.** T physically goes to M, pays M and receive the package

Algorithm 11 Basic protocol

1: C decides on I and M
2: C and M engage T, C accepts c_T
3: T generates r, T computes $R := H(r)$
4: T sends R to C
5: C pays c_T to M
6: CT1: C sends conditional transaction $(C, T, c_I, t_1 + \tilde{\delta}_T^{TM}, (R, 1))$
7: CT2: T sends conditional transaction $(T, M, c_I - c_T, t_1 + \delta_T^{TM}, \texttt{null})$
8: M gives package to T and *at the same time* CT3: T and M send conditional transaction $(M, T, c_I - c_T, t_1 + \tilde{\delta}_T^{TM} + \delta_T^{TM}, (R, 2))$
9: **if** C accepts the package **then**
10: C and T spend CT1 (so T releases r)
11: M spends CT3 using r
12: **else**
13: T brings package back to M
14: T and M send money from CT3 to T's address
15: **end if**

Section 9.2), coded in the transactions. Building transactions this way, I accomplish two things. On the one hand I accomplish a traceable coordination of multiple parties without using external tracking devices. On the other hand, C doesn't need to be on line after the first payment to M and from that payment on the P2T protocol is non interactive from his point of view. This is a huge advantage for C since in this way he can use a device once (for example a public computer in a library) without the need to subsequently check the status of his order.

Fundamental steps of the P2T protocol are summarized in pseudo-code in Algorithm 11 and it works as follows. Customer C decides to buy an item I at time t_0 from the webstore of the merchant M, and he needs I to be shipped to a place which is more close to him. I assume that the cost for the item is c_I, and throughout the protocol the cost of the transportation is assumed to be c_T for each chosen transporter T. An order from C can have multiple information, such as the item's identification number, the item's quantity or the maximum date of delivery which I denote as t_2 (see below for a constraint on t_2). C is required also to provide a blockchain address P_C both as the (only) identification for that shipment and to prove he has enough coins to pay for the item I. On the other hand, C is *not* required to provide a delivery address nor any other identifying information. I emphasize that a blockchain address created specifically for this trade cannot be considered as an identifying element for the customer C. In fact, assuming that C is keen on maintaining its privacy, this address can be funded using the particular technologies of blockchain projects. Examples are the z-shielded transactions in ZCash, the use of a large number of *mixins* in Monero, CoinJoins or similar technologies in Bitcoin or, in general, the use of mixers. Furthermore, this address is used directly only once, so there is no risk of address reuse.

Based on the information provided by C, M and C agree on a transporter T[2]. During the agreement, C specifies what I call the *minimal required zone* (MRZ). A MRZ is the minimal information needed by T to estimate delivery costs and date of delivery. For example, if T charges the same delivery costs and estimates the same delivery time for a whole country, then C communicates only the country where he intends to pick up the item I. Therefore by the agreement T must take the item I from M and take it to his pick up place before date t_2. Here $t_1 + \tilde{\delta}_T^{TM} - \epsilon_T \leq t_2 \leq t_1 + \tilde{\delta}_T^{TM}$ where t_1 is the occurrence of the first payment from C to M (see below for details). Of course, if C and M agree on T, they also agree on the additional costs c_T. All parties M, C and T provide to each other some contact information for possible notifications, e.g. for the arrival of the package at T's pick up place[3]. As soon as T has been decided and engaged in transport, T generates a random number r and creates $R = H(r)$ where H is a hash function. T sends R to C using the contact information provided before. R is the puzzle for the secret, as described in Section 9.2.

After this step, M checks that there are at least $c_I + c_T$ funds in P_C[4] and if so M creates a bill contract x that he sends to C. In x there are some static information about M, i.e. information that persists for more than one order, and some dynamic information regarding the specific order. In particular, the address P_T of T is included in x. C verifies that the information on x are sound and if he agrees on them he sends the equivalent of c_T to address $d_{addr}(P_M, x)$. I call this transaction the *non-redeemable commitment transaction of* C and it is done at time t_1. This payment represents three things about C: it is a proof that C controls the funds in address P_C, it is a proof that C accepts all terms written in x and, being non redeemable by C, it is an incentive not to spam M with fake requests which would result in a DoS attack.

When the merchant M receives the payment from customer C, he is sure that C has serious intentions in buying the object, but he does not know anything about T. For this reason M sends $H(x)$ to T waiting for his commitment transaction.

Before doing that, T needs a commitment from C, so C sends another commitment transaction, this time to T. This commitment transaction is different from the previous one because it is redeemable by C. This is a $(C, T, c_I, t_1 + \tilde{\delta}_T^{TM}, (R, 1))$ conditional transaction with secret R in the first branch of the transaction (see Section 9.2) and I call it CT1. C has to do this payment before time $t_1 + \delta_T^{TM}$, otherwise T cannot go to M in time and T risks to delay the whole shipment process. In case C is too slow to pay, T decides to abort the protocol and notifies other parties. In case T can commit to M, he sends $c_I - c_T$ coins to address $d_{addr}(P_T, x)$ doing a redeemable conditional commitment transaction without secret $(T, M, c_I - c_T, t_1 + \delta_T^{TM})$ which I call CT2. M considers valid T's coin transfer only if

[2] T can be chosen from a billboard or by some convention between the merchant M and T himself

[3] The contact information should not reveal identity of C. For example C can use a throwaway e-mail address or a burner phone

[4] The merchant M can do that because the blockchain is public

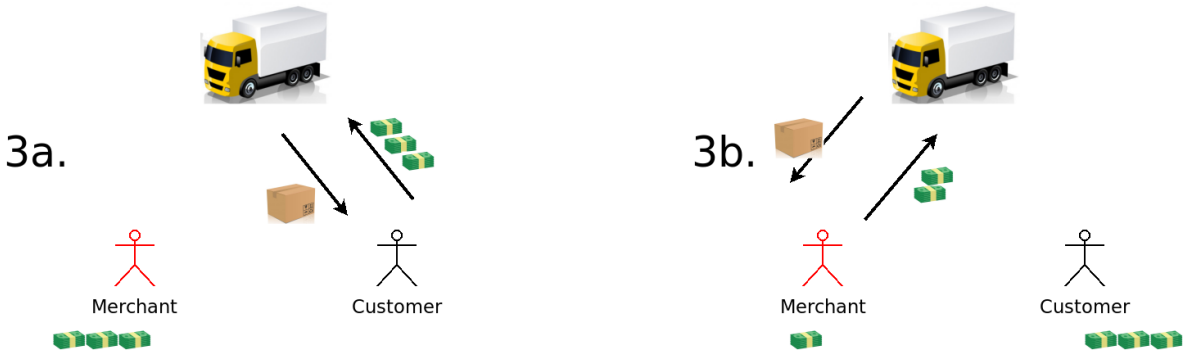


Figure 13.2: Possible endings of the Basic Protocol with One Transporter. **3a.** C physically goes to T and if the package is intact, then C accepts it and pays T, **3b.** otherwise, if C refuses the package, T gives M the package back while M returns T his money

the transaction is built in the way described above, otherwise M aborts the protocol and notifies C of that.

After the merchant M saw the payment, he produces and physically prints a visual representation V (e.g. a QRcode) of $H(x)$ and use it to seal the package. At time $t_1 + \delta_T^{TM}$, T can take this package with item I inside it from M. If M is not malicious, the package of item I is in perfect conditions and T verifies that V is the visual representation of $H(x)$ (recall that T has received $H(x)$ before to create address $d_{addr}(P_T, x)$), both M and T sign the transaction from $d_{addr}(P_T, x)$ to $d_{addr}(P_M, x)$. This is a $(M, T, c_I - c_T, t_1 + \tilde{\delta}_T^{TM} + \delta_T^{TM}, (R, 2))$ conditional transaction CT3 with secret R in the second branch. At this stage, M has received (but cannot use yet) $c_T + (c_I - c_T) = c_I$ coins, so the merchant has received the full price of the item I and the item is shipped. The second conditional transaction with a secret is done to account for the case in which C could refuse the package.

T takes the package to his pick up point. When T and C physically meet at T's pick up point, C checks that the package is intact, that the seal V is not broken and that it represents $H(x)$. If that is case, then C and T spends their conditional transaction CT1 sending funds to an address belonging to T. This way T has to reveal r such that $R = H(r)$ and M can use it to spend CT3. On the other hand, if there is some problem with the package, C refuses the package and T has to bring it back to M. T is sure he can have his coins back because of the conditional transaction CT3.

13.2 Analysis

Privacy and Anonymity From C's point of view, P2T is highly private. In fact, the customer C provides to the merchant M and the transporter T only a public key with funds and a geographical zone (the MRZ) where he intends to pick the package. Depending on the blockchain method used, the source of funds can be obfuscated in a way to detach it from the real identity of C (see Section 9.2). Therefore P2T satisfy also customer anonymity. Note that privacy and anonymity comes at a cost for C. M could steal funds of C and never give him any product, gaining c_T. While this is the case, I assume C won't use merchants or transporters that have any or a bad reputation for big payments. On the other hand, even though C loses funds, he only loses transportation costs. Still, this is better than today's policy for which C must pay the whole cost in advance, and therefore he risks losing both the cost of the item c_I and the delivery cost c_T.

Authentication and Deniability In the P2T protocol there is an intrinsic authentication method. In fact, the public addresses and keys of the merchant M and the transporters T are public and the payments are made to addresses deriving from those public keys using the homomorphic properties of the construction of the addresses. The fact that the entities are able to spend these funds in the derived addresses is proof of their identity. Furthermore, since all entities use derived addresses and not their publicly accessible addresses, an external observer cannot prove that the parties involved have completed a particular exchange thanks to the hardness of the discrete logarithm problem assumption[5].

[5]This is the underlining assumption for the construction of public keys on all blockchain projects.

Therefore all parties can plausibly deny their involvement in an order.

Confirmation and Receipt The use of the blockchain and the particular way parties following the protocol build the addresses and transactions gives both the receipt and confirmation of orders and payments. Furthermore, by following the state of the blockchain the entities involved can track the state of the order by seeing which payments have been already done.

Punctuality This also provide the punctuality property of P2T: time constraints in the transactions force parties to respect all prearranged times or they risk losing funds.

Atomicity and Honesty Transactions are constructed taking into account the possible dishonesty of each participant. Since every transaction is atomic, if a participant does not respect the protocol (that is, he is not honest), he does not receive the coins that would be due to him. Each participant is therefore encouraged to be honest. In other words, following the protocol is the most rewarding behavior.

Trustlessness In P2T, participants do not have to trust the others. This is due to the particular constructions of the transactions. I analyze the protocol from the point of view of each of the participants.

From the point of view of the merchant M, there is no way to lose both the money and the product. In fact, once the product has been given to the transporter T, the transaction CT3 assures the merchant that (if customer C accepts the package) he will be able to spend his coins. This is because M supervises the creation of this transaction (M and T are in the same place at the same time) and can verify that the hash placed by T is the same as the one in transaction CT1. Furthermore M will know about the preimage of the hash the moment T redeems CT1. If, on the other hand, C does not accept the product, theoretically T may decide not to return the package to M. But this would not be a rational choice for T. The transporter, in fact, does not know what is contained in the package (therefore a priori may not be interested in the article) and he is therefore encouraged to return it in order to redeem the money stuck in the multisig with M.

As far as the transporter T is concerned, he is interested in not losing the money invested to earn the transport commissions. T cannot lose money in CT2 (its first transaction) since it is atomic and T only executes it after CT1 has been confirmed. T risks losing money in CT3 if C doesn't accept the package and M doesn't show up for the return. In this case the time-lock would expire and M could redeem the transaction. This is not possible because M must also solve the hash-lock contract, and to solve it M needs the preimage revealed by T. T reveals this secret only if C accepts the package. In this regard, note that if the secret had been created by C, T would not have had the same assurances. In fact, given the anonymity of C, he and M could be the same entity, or colluding. If that were the case, then C could refuse the package and M could still redeem the coins in CT3 because he is aware of the preimage.

Finally, C doesn't need to trust anyone too. Once the shipping costs have been paid (which C agrees to lose if the package is refused) C creates and sends the atomic transaction CT1 . On the scheduled date, C goes to the pick up point of T and decides whether to accept the package and sign the transaction with T or to refuse the package. In this latter case, C only has to wait for the time-lock to expire in order to use its coins again: in the meantime T cannot spend them because they are in a multisig.

13.3 Conclusion

In this chapter, I presented P2T, a payment protocol for the exchange of coins for physical goods. P2T is trustless, privacy preserving and preserves the anonymity of customers without using external tracking systems, arbitrators or deposits. The protocol uses mechanisms common to all blockchain protocols, so that it is possible to implement it in all these projects. I implemented a proof of concept that uses the Bitcoin blockchain and that can be found online. In addition to privacy and anonymity, the protocol satisfies other properties such as plausible deniability and encourages participants' honesty by using atomic transactions.

Chapter 14

Payment Processing

One of the current problems mentioned in Chapter 6 is the unavailability of multi-token solutions for accepting payments. As seen in Section 6.5.4, while many ad-hoc solutions are currently used to receive payments, the blockchain space lacks an integrated multi token payment processor. The consequences are manifold: exchanges are forced and costly and the user experience is bad since the user needs to exchange funds before buying a product/service he or she wants. This create friction and consequently less revenue for the seller and less overall benefits for users. Here I present the Universal Token Swapper [BSS21], which solves these problems. UTS is a smart contract for EVM compatible blockchains acting as a payment processor between buyers and merchants in a multi-token environment. UTS gives merchants the ability to accept any token by performing instant conversions (Figure 14.1). This way, the buyer does not have to convert the tokens before a payment and the merchant is not subject to financial ruins if he does not want bear the risk on speculation on the value of the token received. In addition, UTS can be implemented to follow any regulation by linking the contract to on-chain oracles, bringing regulatory compliance to DeFi and helping merchants take advantage of the new possibilities.

Through the payment processor the buyer B can pay the merchant M with any token chosen by B. UTS automatically leverages aggregators to operate a swap into a token specified by M. Since UTS is a smart contract for a public blockchain, it is decentralized in nature: any merchant can deploy its version of it. I provide an implementation[1] of the UTS. The chapter is organized as follows. In Section 14.2 I describe how UTS works and detail the implementation. In Section 14.3 I explain the possible applications of the functions that compose the payment processor and in Section 14.4 I analyze security and costs of the proposed method.

14.1 Decentralized Markets in Ethereum

In this section I briefly explain what solutions are present to date for exchanging coins and tokens. This is one of the aspects of recent decentralized finance (DeFi). For further details see [WPG+21].

The transition from centralized to decentralized markets has not been easy, and it's still not finished. Initially one of the problems was liquidity: decentralized markets did not have access to the same amount of capital as centralized markets [LBC+19]. One of the first DEX, EtherDelta [eth18], used a book of "resting" orders, i.e. signed intents to trade stored off-chain so that each order doesn't waste ethers/gas to be submitted. The EtherDelta smart contract allows the user to deposit or withdraw Ether or any ERC-20 Ethereum token in a decentralized exchange. Another problem of decentralized exchanges was order-confirmation latency and resource-wasting [HHW18]. Bancor [HBB17] was one of the first protocols presented to address this issue using a bonding curve for price setting[2]. In other words, given a token with a total-supply $totSup$, then the supply Sup can be computed as

$$Sup = totSup - outSup$$

[1] See: https://github.com/tryvium-travels/uts-paper-example

[2] A bonding curve is a mathematical curve that describes a relationship between price and token supply. Generally the bonding curve formula is such that the price increases as the supply of the token increases. While counter-intuitive, that's obvious once considered that in a bonding curve based token an increase of the supply occurs only as a consequence of an increase in demand [HBB17].

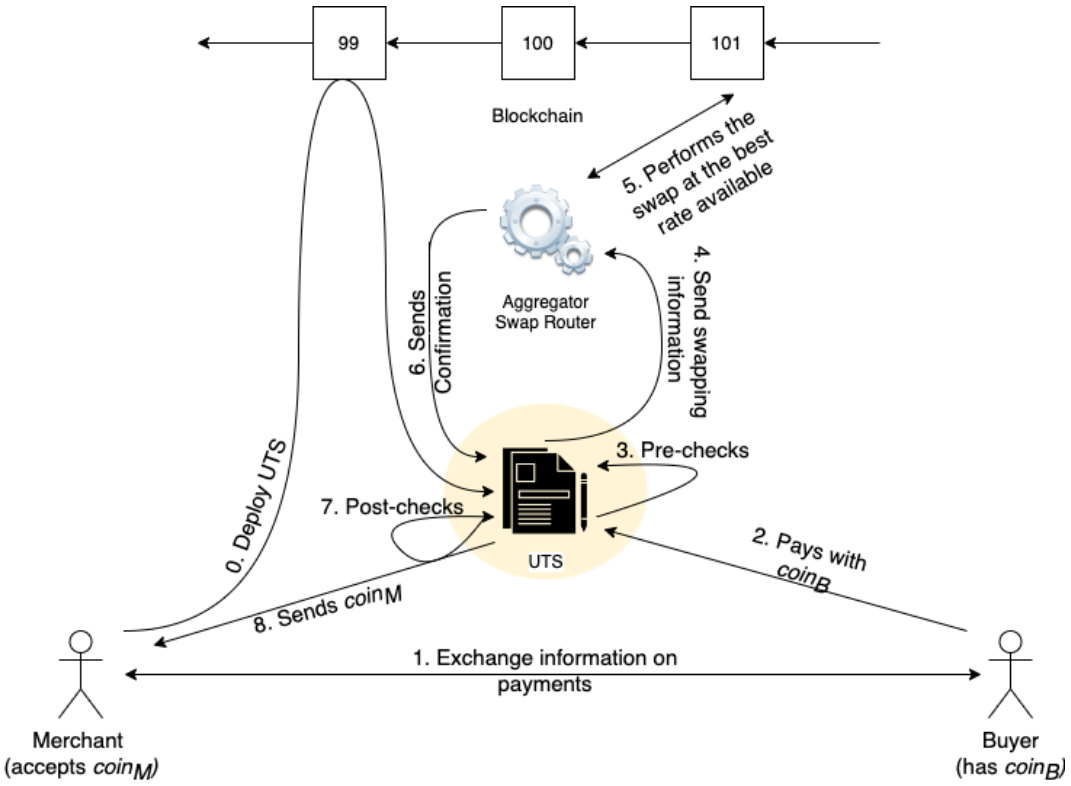


Figure 14.1: The DeFi ecosystem lacks a payment processor to link the blockchains to real life commerce and services. UTS solves this problem.

where *outSup* is the outstanding supply. So an increase in *Sup* means a decrease in *outSup* and consequently an increase in the rarity of the token supply in the future. The bonding curve has given birth to what are called "continuous tokens", because these tokens that are created in real time and sold according to the formula given by the bonding curve. This way there was no need to waste time and resources in order-posting since a user would automatically use the rate imposed to the market by the bonding curve.

However, Bancor introduced new problems. As informally explained by Buterin [But17] and Gün Sirer and Daian [SD17], Bancor has efficiency problems with respect to on-chain market making. The main problem is that the liquidity of users is managed by a smart contract that by definition cannot keep track nor interpret of events in the real world. This means that in case of a panic in the market generated by a false news, the smart contract generates market prices based on a formula that risks putting in danger the finances of the users. On the contrary, if users could manage their finances by themselves, they could decide not to sell during the panic, and wait for the real news to calm the market. After Bancor, Buterin proposal of a new formula to price a new token *tokenB* with respect to an old token *tokenA*, the *constant product formula*, spawned the new generation of DEXes. The best way to intuitively understand this system is to think of decentralizing the liquidity (i.e. the funds) of centralized markets: instead of having accredited investors and professional market makers, liquidity pools accept capital from anyone who has the necessary tokens. Those who provide liquidity are incentivized by earning a portion of the commissions on trades or loans.

Uniswap [AZS+21] has been one of the first projects to use the constant product formula and liquidity pools to implement an AMM (automated market maker). Details on AMMs can be found in [XVP+21, Wan20, BCL21].

Multiple markets based on different assumptions have brought arbitrage opportunities and the consequent birth of systems called "aggregators". The aggregator protocol of 1inch [iT21] was one of the first services able to capture the difference between a quoted price and the executed price where that difference is in favor of the customer. This is called the "spread surplus". Other swap routers like Matcha [Tea20] aggregate the best prices across multiple liquidity sources with the goal of maximizing

the value a user receives on each trade.

Despite the possibilities of the DeFi ecosystem, it is not yet possible to automate the token exchange process for non-financial applications[3]. In particular, it is not possible to automate the payment process for services or products. Consequently, today a buyer must keep his cash in many tokens in case a merchant accepts only one of them. This exposes the buyers to financial risks that they may not want to manage. At the same time, merchants also cannot accept multiple tokens for the same reasons. The goal of UTS is to solve this problem by creating a payment processor with automatic token swapping.

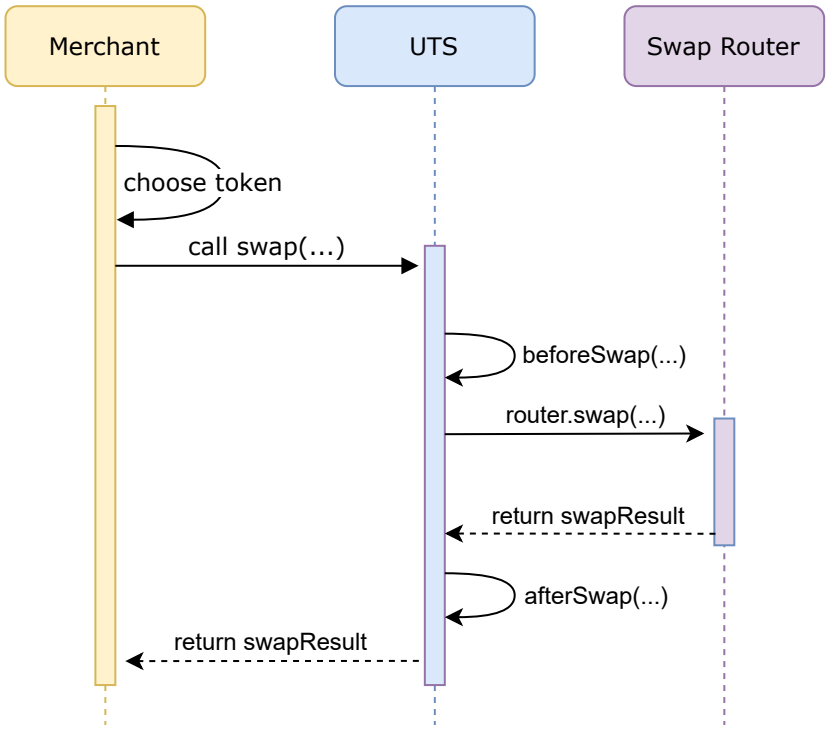


Figure 14.2: Design of the UTS payment system.

14.2 Design

I assume that there is a buyer B who buys a service S issued by a merchant M. The other assumption is that the coin $coin_B$ that B wants to use to pay for the service S is different from the coin accepted by M which I denote with $coin_M$. For simplicity I assume that $coin_B$ and $coin_M$ are connected by a rate *alpha*:

$$coin_M = \alpha \cdot coin_B$$

meaning that it is possible to exchange 1 coin $coin_M$ in exchange of α coins $coin_B$. α is variable since there are different DEXs that are able to exchange $coin_B$ for $coin_M$ and vice versa. Specifically, if there are $D_1, D_2, \ldots, D_k$ that are k different DEXs, I denote by α_i the current exchange rate in the DEX D_i. UTS doesn't interact with the DEXs directly. It uses existing services such as aggregators' swap routers such as 1Inch [iT21] or Matcha [Tea20]. I denote these services R.

14.2.1 UTS Design

I now describe the system using Figure 14.2 as reference. After the buyer B chooses S, the front end of the merchant site will direct B to a checkout page with the information needed to make the payment. There B can choose the token he wants to use to pay for S. After that and a confirmation from B, the merchant's front end call the `swap` function of the UTS smart contract. The smart contract

[3]It is possible to practice automated trading strategies, but decisions are made by off-chain algorithms that trigger on-chain trades.

then starts a sequence of three routines: `beforeSwap`, `routerSwap` and `afterSwap`. In the following I explain them in detail.

`beforeSwap`: In `beforeSwap` the contact performs generic checks on the addresses and coins involved. For example, in the Tether token some addresses are blocked and cannot perform any transaction due to legal reasons, e.g. fraud or theft[4]. If for example B's address is in the list, the `beforeSwap` routine would stop the swap and abort the selling of service S. Of course, the deployer of UTS can perform any other check in this routine: for more details on the possible uses of this function see Section 14.4.

`routerSwap`: The main function `routerSwap` calls the swap router R. The main goal for R is to find the most convenient (for M) exchange rate $\alpha_i, i = 1 \dots k$ between $coin_B$ and $coin_M$ and consequently perform the swap. The inner working of R is outside the scope of the Chapter and it follows implementations such in 1inch or Matcha. The function returns the swap-transaction hash in case of success or abort otherwise, e.g. in case of a too high slippage.

`afterSwap`: In case of success, the goal of the `afterSwap` function is to manage the newly received funds. For example, M can automatically stake the funds or provide liquidity for a loan platform. Otherwise, `afterSwap` can call `beforeSwap` again if `routerSwap` fails and try a new swap again. When `afterSwap` completes the process, it sends the `swapResult` to M's dApp. If successful, B receives a payment confirmation. This ends the work of the payment processor.

Two other functions are need to manage potential problems. I introduce them here for completeness:

`transferFunds`: This function can only be called by the contract owner and works only in case the contract is in a `paused` state (see below). This function transfer all funds in the smart contract to another smart contract chosen by the owner during the function call. This function is used in case of possible problems as described in Section 14.4.1.

`pause`/`unpause`: These two functions secure the contract in case of bugs in the smart contract or in the swap router used. It is critical that these functions be part of this contract in every implementation. In fact in case of bugs, the owner of the contract can call the `pause` function to stop the execution. When the contract is in the pause state, no operation can be performed. This is enforced because all interface functions (the only public ones besides these) check that the contract is not in a `paused` state. If the bug has been solved the owner of the contract can call the `unpause` function to restart the execution.

14.3 Applications

In the following I explore how these functions, especially `beforeSwap` and `afterSwap`, enable new possibilities for real life commerce.

14.3.1 Easiness of commerce

One of the reasons there aren't many merchants accepting tokens in traditional commerce is that merchants don't know which coins to accept. On the one hand, if a merchant M accepts only a subset of tokens then M is effectively losing potential buyers since buyers who don't have the tokens must necessarily exchange them in some market before they can pay for the service which ruins their experience. Consequently buyers will choose other merchants, thus decreasing M's potential profits. On the other hand, accepting all tokens is impossible for M: if the merchant does not exchange the received tokens for a token he trusts immediately after receiving them, then M exposes himself to financial risk; if M does exchange them immediately then it is a time-consuming manual process which necessarily diminishes the benefits of opening up to new payment methods. Moreover, it is also impossible to practically accept all tokens because new tokens are created every day.

UTS can to improve this situation. The merchant M can open up to new payment methods even if he does not have any special financial or computer knowledge using the automation properties inherent in the use of smart contracts for payment processing.

[4] See for example `https://dune.xyz/phabc/usdt--banned-addresses` for a list.

Solc version: 0.8.13		Optimizer enabled: true		Runs: 1000000	Block limit: 30M gas
Contract	Method	Min (Gwei)	Max (Gwei)	Avg (Gwei)	# calls
`SimpleOneInchV4TokenSwapper`	`swap`	123,981	181,336	143,099	6
`SimpleOneInchV4TokenSwapper`	`pause`	-	-	27,749	4
`SimpleOneInchV4TokenSwapper`	`unpause`	-	-	27,702	2

Table 14.1: The table lists the costs of the functions as the Hardhat output.

14.3.2 Compliance with Regulations

Another problem of traditional commerce in relation to the new methods of payment through a blockchain is the compliance with regulations. In fact, at the moment regulations are not clear nor shared between to all countries [Wro21, Pad20, UA21]. In this sense the functions `beforeSwap` and `afterSwap` make the system adaptable to regulation changes and transparent.

The `beforeSwap` function lets M be sure that there won't be any legal problems in accepting a payment: the function can effectively perform operations such as anti-money laundering (AML) and know-your-customer (KYC) checks before starting any payment. To do that, the function would call other smart contracts or on-chain oracles to abide to any legal framework [BCC+21]. This is similar to any payment via a credit-card circuit, but in a decentralized way. This is a clear advantage for M: since the payment rules are on a public blockchain auditable by anybody, it is easy for authorities to check that M is performing a service complying to the financial regulation and it can likely speed up commerce-establishment processes.

The `afterSwap` function also allows the merchant to treat the revenues in compliance with the law, e.g. by allocating a portion of the funds to pay taxes. This is also transparent for the aforementioned reason.

14.3.3 Easier (decentralized) financial access

Another benefit for M is the possibility to employ the newly received funds instantaneously. In fact thanks to the `afterSwap` function M can decide how to allocate funds for each sell. For example, M can decide to allocate after-tax funds up to a threshold t_1 in a liquidity pool: this way M earns more money through trading commissions in DEX. Another possibility is to keep funds up to another threshold t_2 in an accrual deposit. Assuming for example a buyer B who decides to buy S in multiple installments, M can request a deposit from B: in this way if B does not pay regularly, M can withdraw the funds he is entitled to from the deposit. The use of a deposit is also an advantage for B. In fact if the deposit is a multisignature shared between B and M, and the deposit is staked, then the deposit value increases while being a secure way for M to receive the installment payments.

In general, then, the `afterSwap` feature allows both M and B to automatically benefit from all the financial opportunities inherent in the new decentralized finance.

14.4 Analysis

In this section I will analyze UTS from the point of view of security, costs and easiness of use.

14.4.1 Security

The authors in [ABC17] group the possible vulnerabilities according to the layer in which they are introduced, namely Solidity (the language I implemented the code in), EVM and Blockchain. In this section I analyze two cases (one for Solidity and one for EVM/Blockchain) to prove how UTS is resilient in case of attacks based on those vulnerabilities.

I first describe how implementation decisions regarding UTS allow a merchant M to securely react to possible Solidity bugs described in [ABC17]. In particular, I analyze the case of a *reentrancy* bug, a bug that let malicious actors to drain the smart contract. Assume this bug appears in the `afterSwap` function that can withdraw funds from the UTS contract deployed by M, and assume an attacker A exploit the problem. In this case, the combination of function `pause` with function `transferFunds` allows M to mitigate the problem before it becomes a major one. In fact M can stop the execution

Protocol	Avg Gas (Gwei)
Curve	112,845
SushiSwap	141,176
UTS	**143,099**
Uniswap V2	149,273
Mooniswap	149,767
DODO	157,185
Crypto.com	180,082
Balancer	206,765

Table 14.2: Comparison of the `swap` function in different DeFi protocols as seen at https://crypto.com/defi/dashboard/gas-fees. The UTS payment processors is ranking third, but still lacks of gas optimizations.

of the contract related to the new swap router and transfer the funds towards another smart contract, probably a new version of the same one but without the vulnerability: as seen in Section 14.2, the function `transferFunds` can be called only by the owner of the contract (M) and it works only if the contract is in a paused state.

However, not all threats come from possible internal errors. The two parts-implementation of UTS can mitigate attacks generated in other parts of the blockchain. Assume that a swap router used by M has a problem, but that M knows of this bug after the swap part of the smart contract is deployed. Swap router problems are not under the control of M which therefore cannot control what or when those problem happens. However, by having a dedicated part of the contract for each swap router, M can mitigate this problem and use a combination of function `pause` with function `transferFunds` as in the previous case and block the use of the buggy swap router for the payment processor. This does not block the total operation of UTS since it is possible to use other swap routers. Therefore, the payment processor, even if automatic, remains under the control of M.

A similar argument can be made if a problem doesn't regard swap routers but arises with the DeFi protocol used in the `afterSwap` function for handling new funds. For example, it may happen that the liquidity pool protocol where M deposits the funds is hacked and it is necessary to block the contract. The process of handling this issue is similar to the previous one.

14.4.2 Smart Contract Costs

In Table 14.1 I reported the gas costs of the contract functions as reported by the Hardhat suite. The function that costs the most is undoubtedly the swap function[5] with an average cost of 143,285 Gwei. The swap function uses a swap router (the 1Inch one in the tests) to find the best α_i exchange rate for M. In this sense, UTS is comparable to a DEX using a swap router. Therefore, in Table 14.2 I compared UTS with other DEXs. UTS comes in third, but has not yet been optimized in this respect.

14.5 Conclusions

I have described UTS, a smart contract that aims to link traditional commerce with decentralized finance via and we proposed an implementation of its functions. Its modular structure on the one hand allows to implement payments in a way that supports regulations and on the other hand gives merchants the possibility to take advantage of the new decentralized and efficient credit, borrowing and lending mechanisms implemented so far in DeFi.

[5] We also evaluated the functions `pause` and `unpause`. Both require about 27,000 Gwei for the gas. The `beforeSwap` and `afterSwap` functions were not reported as they depend on the implementation by M.

Part IV

Conclusions

Results

The first goal of my research has been the analysis and the creation of new ways to make these changes easier, more secure and more private from a technical point of view: this goes under the name of *Blockchain Interoperability*.

Since we don't exchange information only, but also material products, I decided to explore the dynamics between blockchain and commerce. Therefore the second goal of my research has been the creation of new ways to make these *product* exchanges easier: this goes under the name of *Blockchain and Commerce*.

Blockchain interoperability

The first building block was not directly an interoperability protocol. At first I investigated how it is possible to obfuscate the history of transactions on a blockchain using what I called DMix [BS20a] in Chapter 9: the protocol requires coordination between a group of people to be effective and being its privacy assumptions based on the size of the anonymity set, the higher the number of people involved the better DMix works.

I leveraged the privacy properties and the multi-party setting of DMix to obtain MP-HTLC [BS22b], an atomic swap protocol to achieves interoperability. The big improvement of MP-HTLC over HTLCs is its multi-party architecture. This alone strongly mitigates the privacy problems of HTLCs we saw in Section 5.3.2.

But MP-HTLC is not an "ultimate solution" for all interoperability problems because of two opposite problems. MP-HTLC was born as a general purpose solution: it doesn't need smart contract capabilities (even if can be used in these cases too, as we proved in Section 10.4 and Section 10.5), it highly leverages digital signatures (which is why I gave an extended treatment of the topic in Chapter 4) and "waiting time" as synchronicity mechanism. These techniques can be leveraged in Bitcoin, currently the most used blockchain, and are fairly efficient for that blockchain. They *can* work on other blockchains too, albeit in a non-efficient way: some blockchains have highly expressive languages (as in Ethereum) and other blockchains have no scripting capabilities at all (as in Monero). So my research went toward those opposite extremes.

For those blockchains that have no scripting capabilities and are used only for transmission of funds, I designed BTLE [BMS21], which I presented in Chapter 11. In this protocol all computation and verification is moved off-chain thanks to cryptographic primitives such as time-lock puzzles and zero-knowledge proofs of knowledge. The blockchain is used only as settlement layer. Incidentally, BTLE can also be used as a matching engine since it is modular: I split the two components into BTLE-MA (Matching Algorithm), which deals with the parties discovery and matching (also off-chain based) and BTLE-AS (Atomic Swap), which deals with the actual exchange.

For those blockchains that have high scripting capabilities and are used for general purpose smart contracts I proposed BxTB [BS22a] and presented it in Chapter 12. This protocol obtains private interoperability for Bitcoin on any EVM-compatible blockchain, such as Ethereum. We obtain that by implementing a special-purpose light client for the EVM-compatible blockchain (here is how I leveraged the available scripting capabilities) which secures the exchange of bitcoins for any minted wrapped bitcoin token. I said the interoperability method is private since this protocol bypass the needs of KYC or AML regulations required in current token systems.

Blockchain and Commerce

The problems found in digital-products exchanges are also found in current physical-products exchange protocols. In particular, the lack of privacy and the need for large expressive capabilities of programming languages slow down the adoption of blockchain-based mechanisms in commerce.

For this reason, I created P2T [BS20b], which stands for Pay-to-Transport, the method of transporting physical products that I presented in Chapter 13. The protocol focuses only on the actual transport of the product once it has been chosen. The basic idea of the protocol lies in the fact that the transporter pays part of the price of the merchandise to the seller: this is different from the usual buying flow, where the customer pays the entire price of the product plus the shipping costs to the merchant. By doing so, it is possible for the customer and the merchant to know the transporter's

movements only: outside observers cannot follow anything since they do not have necessary information. Nevertheless, the transactions are indistinguishable from normal transactions. As a result, customer data remains private since the tracking system is based only on transactions that appear on chain.

After the delivery itself, I focused on how to remove the friction in the payments themselves. In fact, beside the aforementioned problems, the availability of thousands of tokens is currently seen as a problem instead of a feature of the blockchain space. To start solving this problem, I designed UTS [BSS21], the universal token swapper, presented in Chapter 14. This is a self-deploying smart contract a merchant can use to accept any token as long as its quantity has the same value as the bought product's value. The smart contract essentially swaps the tokens received for the token the merchant wants to hold. To do that, it leverage the composability properties of the DeFi ecosystem and therefore I highlighted how it can be used to do regulatory check and automatic investment plans.

Future Works

As they say, "the road to perfection is always under construction" and new and additional work is needed to effectively create easy-to-use ways to move funds between different blockchains and platforms and easy enough protocols and applications that can be used by non-technical users. In the following I delve into those improvements that I personally want to continue work on.

Solving the Anonymity Set Issues

The big thing my privacy-related protocols have in common (see for example DMix [BS20a] in Chapter 9) is that they all rely on an anonymity set. This is one of many privacy-mitigation techniques. It has many advantages, like the possibility of deeply measure *how much* anonymity/privacy is increased and the easiness of use and deployment in protocols. On the other hand the technique essentially relies on the network effect of the protocol or application it is used into. In fact, to be effective, an anonymity set must be "big": more formally, the theoretical probability of being recognized in an anonymity set is always $\frac{1}{N} = N^{-1}$ where N is the cardinality of the set. So the bigger N, the smaller the probability of being recognized.

Depending on the goal of the protocol relying on the anonymity set, this probability may decrease too slowly and impact the real world application of the protocol, especially since that is a theoretical measure and in real world applications users can compromise the entire security of the protocol if they do not know how to manage their own security[6]. Currently the academic literature is focusing on how much of a problem that is (see e.g. [KKS22]). Undoubtedly, a better probability function would be $\frac{1}{N^2} = N^{-2}$ or any function $f(N)$ such that $f = o(\frac{1}{N}), N \to \infty$[7]. Unfortunately there is no work in this direction (that I am aware of).

Form this situation, I see two ways for moving forward. The first one is to find a way to create a structure such that its probability function related to the anonymity is $o(\frac{1}{N}), N \to \infty$. This would be a very theoretical study, but it does not seem very promising since there are not works in this direction (again, that I am aware of). The second way is to leverage other privacy threat-mitigation assumptions and either incorporate them into methods already created or create new ones so as to expand the range of attacks these protocols can withstand. One example is to study more deeply how zero-knowledge proofs are currently used to guarantee privacy (and sometimes anonymity) in blockchain protocols and include the state of the art into new protocols while hopefully expand that state.

Solving the Implementation Issues

For the majority of my works I provided proof of concepts or demos. Yet, some of these implementations are not ready for production or can not be used by non-technical users. This greatly reduces the number of people who can use the code. Besides being something I want to improve for its own sake, because

[6] For example, imagine the case where three users A, B, and C control an address together via a 2-of-3 threshold signature method, and e.g., A saves his share of the key in an unprotected cloud-storage: an attacker at this point no longer has to attack 2 people, but only 1, a 50% reduction in security due to a mistake by only one participant.

[7] A function $f(x)$ is an *small-o* of $g(x)$ for $x \to \infty$ if $\frac{f(x)}{g(x)} \to 0$ if $x \to \infty$. In this case $\frac{1}{N^2}$ is $o(\frac{1}{N})$ since $\frac{1}{N^2} / \frac{1}{N} = \frac{N}{N^2} \to 0$ if $N \to \infty$.

it is bad to me that few people use what I have made, it is also a security problem in those applications that rely on the anonymity set.

One of my future goals is to improve the usability of the protocols. In particular, the UTS payment processor [BSS21] (Chapter 14) needs templates that can be deployed on a blockchain by anyone. Another example is the P2T delivery protocol [BS20b] (Chapter 13) for which it would be nice to be able to create a working platform where merchants and transporters can sign up and buyers can buy products (without signing up, to maintain privacy as per the protocol).

A similar argument can be made for MP-HTLC [BS22b] (Chapter 10) and BTLE [BMS21] (Chapter 11). Giving users the ability to exchange tokens smoothly would help the interoperability of the different blockchains, and since my protocols were born to be private, this would be a strong improvement over current services.

New Directions in Blockchain

it is vastly believed that blockchains are an improvement respect at the current methods of information exchange, both from a networking point of view and from cryptographic (and therefore security) point of view.

In this thesis I analyzed blockchain interoperability as if this were its the end goal, but this is not the only way to approach this topic. As we saw in this thesis, permissionless blockchains, such as Bitcoin and Ethereum, are decentralized networks that allow anyone to participate as a node and validate transactions. We mentioned that one of the main challenges with these types of networks is scalability, as the number of transactions that can be processed per second is limited by the design of the protocol. This results in slow transaction speeds and high fees, which hinders the adoption and use of the blockchain. To address the scalability issue, researchers and developers in the blockchain space have focused on interoperability (beside other methods), so the concept of blockchain interoperability can be seen as a sub-topic of scalability too.

In this regard, one approach to interoperability is downstream from the use of multi-layer solutions, such as rollup in Ethereum and the Lightning Network in Bitcoin, which aim to increase the transaction throughput of the base layer by moving some of the computation to an auxiliary layer. Another approach to scaling permissionless blockchains is through the use of sharding, which involves dividing the network into smaller groups or shards, each of which can process transactions in parallel. This can increase the overall transaction throughput of the network, but it also introduces additional complexity to the protocol and can potentially compromise the security and decentralization of the blockchain.

In the following I briefly explain the current problems in the two approaches and then explain why my research (and a continuation of it) can help in solving them.

Layer To Layer Interoperability There are two ways to frame the topic of interoperability in the context of layers of a blockchain: interoperability with the mainnet and layer-to-layer interoperability. Interoperability with the mainnet refers to the ability of a layer two (L2) solution to communicate and exchange information with the base layer of the blockchain, also known as the mainnet. Layer-to-layer interoperability, on the other hand, refers to the ability of different layer two solutions to communicate and exchange information with each other.

From a research perspective, interoperability between the L2 and the mainnet is generally easier to achieve because the L2 protocol has been designed to be interoperable with other systems from the start. In Ethereum, for example, the use of smart contracts allows L2 solutions to interact with the mainnet in a standardized way. Additionally, the implementation of *protodanksharding* (EIP-4844) in Ethereum has made it easier to achieve interoperability by allowing the main net to store "blobs" of data that can be accessed by layer two solutions. This helps to reduce the overhead of storing large amounts of data on the main net and allows layer two solutions to operate more efficiently.

For these reasons the concept or L2-to-L2 (or L2/L2) interoperability seems (at least to me) more interesting. L2-to-L2 interoperability does not have a technical and shared definition yet, but it may be defined as the possibility of asset exchange from an L2 to another one without using the mainnet. In fact, it is very easy to pass from and L2 to another using the mainnet, like it is easy to exchange assets from one blockchain to another by using an off-chain method (e.g. central trusted party such as online exchange). Unfortunately relying on the mainnet for asset exchanges on L2s is problematic due to the high fees and long wait times associated with depositing and withdrawing from the mainnet

(up to two weeks for optimistic based rollups). These issues can lead to centralization, as users are discouraged from switching between L2 solutions and may opt to remain on a single platform.

The consequences of these problems can be significant. If users are unable to easily move between L2 solutions, then each L2 will need to have a complete ecosystem of decentralized apps (DApps) in order to remain competitive. This can lead to a lack of specialization, as each L2 will need to offer a wide range of services in order to attract and retain users.

However, if the issues of high fees and long wait times can be resolved, it will be possible to create a more diverse and specialized ecosystem of L2 solutions. Users will be able to easily move between different L2 platforms, allowing for a greater range of specialized services. This will also prevent (or at least make it less probable) the possibility of centralization or monopoly in the L2 ecosystem, as users will have more options to choose from. Additionally, the improved interoperability between L2 solutions will likely lead to the development of a more robust ecosystem of L3 solutions, as it will be easier for users to access these services.

Cross-shard exchanges We mentioned sharding in Section 2.2.2, i.e. the partitioning of the committee of validators into multiple committees which validate different transactions. One consequence is the creation of different non-intersecting successions of events (in the form of transactions) which as in blockchains are joined into different blocks. In this sense, the abstraction of shards as "parallel blockchains" seems fitting. One consequence then is the so called *cross-shard transaction* problem. The problem can be so stated: How is it possible to manage a transaction starting from shard $shard_A$ affecting data in $shard_B$? This problem is also called the "Hotel-and-train" problem [But] since the scenario of the example is quite simple and lets readers understand the actual problem. The example involves a user that wants to purchase a train ticket and reserve a hotel. As usually it is the case, the user wants to make sure that the operation is atomic, i.e. either both the hotel and train reservation succeed or neither do. This property is quite simple to obtain if the train ticket and hotel booking applications are on the same shard: this would happen naturally in systems like Ethereum where either all steps of a transaction succeed or the whole transaction reverts. On the other hand, if the two reservations are on different shards this is not so simple since it requires each shard to have "some knowledge" about the other shards.

One way to achieve atomicity in a transaction between shards is by using *lock* and *unlock* operations. Using the aforementioned example, a client C that initializes a cross-shard transaction from $shard_A$ to $shard_B$ will see its transaction locked in $shard_A$ until the validators/miners of the shard can process it. After that, the transaction is unlocked and sent to $shard_B$. The problem with this setting is the time needed for the locking and unlocking which can span multiple blocks. Another problem concerns the possible censorship that validators of $shard_A$ can do by just deciding not to unlock a transaction. For a detailed account on how different sharding proposals manage the cross-shard problem see the work by Han et al. in [HYL+21].

It's worth mentioning that some current project, such as Zilliqa [Zil17], do use sharding as part of their operations while purposefully *not* dealing with the cross-shard problem: see for example the Elastico proposal [LNZ+16] which is the base of Zilliqa and vaguely approach the topic. The most promising way to deal with it, however, comes from the Danksharding proposal which as of the time of this writing (January 2023) uses an active and central coordinator to enforce the cross-shard transactions[Pro]: "The main innovation introduced by Danksharding [..] is the merged fee market: instead of there being a fixed number of shards that each have distinct blocks and distinct block proposers, in Danksharding there is only one proposer that chooses all transactions and all data that go into that slot."

It is obvious to me that while sharding is a promising solution for the scalability problem of the blockchains, it also has an interoperability problem inside, which I am very passionate about.

New Access Structures Distributed cryptography is a method of performing cryptographic tasks such that a group of independent parties works together to achieve a common goal. In a distributed cryptography system, the parties involved may be geographically separated and may not necessarily trust each other, but they are able to collaborate effectively through the use of cryptographic methods.

One obvious data structure that enables (or belongs, depending on the point of view) distributed cryptography is a blockchain system, but many other methods, for example MPC and its application, do enable this relatively new kind of cryptography.

Therefore it is fair to say that distributed cryptography has become prevalent in modern times. However, many of the distributed cryptographic schemes currently in use rely on *threshold* cryptography, which assumes that all parties involved may potentially act maliciously and are therefore treated equally. This results in a monoculture view of the system which is downstream from the binary classification of actors (either malicious or not malicious). In contrast, a more general form of distributed cryptography allows for the specification of arbitrary sets of authorized parties to perform a task, rather than relying on a simple threshold. In other words new Access Structures (AS) [AC22] are needed to describe real world power structures.

New ASs have the potential to enable a wide range of applications, one of which is the expansion of threshold signatures to *generally-distributed signatures*. This can be used to improve interoperability between different blockchain systems with varying access structures, potentially reducing the risk of funds being lost due to correlated failures [Vog06].

Another application of distributed cryptography is in private fund management. Currently, it is theoretically possible to transfer the private keys associated with a specific address to a new owner, but there is no way for the new owner to be certain that the old owner has deleted their copy of the key. With distributed cryptography, it may be possible to design an access structure that allows the new owner to be confident that the old owner no longer has access to the funds.

Bibliography

[66613] IEEE standard for identity-based cryptographic techniques using pairings. *IEEE Std 1363.3-2013*, pages 1–151, Nov 2013.

[Aav22] Aave protocol version 1.0 - decentralized lending pools. Aave, June 2022.

[ABC17] Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli. A survey of attacks on ethereum smart contracts (sok). In *International conference on principles of security and trust*, pages 164–186. Springer, 2017.

[AC22] Orestis Alpos and Christian Cachin. Do not trust in numbers: Practical distributed cryptography with general trust. *IACR Cryptol. ePrint Arch.*, page 1767, 2022.

[ACD19] Joël Alwen, Sandro Coretti, and Yevgeniy Dodis. The double ratchet: Security notions, proofs, and modularization for the signal protocol. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part I*, volume 11476 of *Lecture Notes in Computer Science*, pages 129–158. Springer, 2019.

[ACP20] Isra Mohamed Ali, Maurantonio Caprolu, and Roberto Di Pietro. Foundations, Properties, and Security Applications of Puzzles: A Survey. *arXiv:1904.10164 [cs]*, April 2020.

[Adl19] John Adler. Minimal viable merged consensus. EthResear.ch, 2019.

[AEYG17] Riham AlTawy, Muhammad ElSheikh, Amr M. Youssef, and Guang Gong. Lelantos: A blockchain-based anonymous physical delivery system. In *2017 15th Annual Conference on Privacy, Security and Trust (PST)*. IEEE, aug 2017.

[AHS20] Jean-Philippe Aumasson, Adrian Hamelink, and Omer Shlomovits. A survey of ECDSA threshold signing. *IACR Cryptol. ePrint Arch.*, page 1390, 2020.

[AKR+13] Elli Androulaki, Ghassan Karame, Marc Roeschlin, Tobias Scherer, and Srdjan Capkun. Evaluating user privacy in bitcoin. In Ahmad-Reza Sadeghi, editor, *Financial Cryptography and Data Security - 17th International Conference, FC 2013, Okinawa, Japan, April 1-5, 2013, Revised Selected Papers*, volume 7859 of *Lecture Notes in Computer Science*, pages 34–51. Springer, 2013.

[And11a] Gavin Andresen. Address format for pay-to-script-hash, 2011.

[And11b] Gavin Andresen. Bip-11: M-of-n standard transactions. Github, 2011.

[And13] Gavin Andresen. Bip-16: Pay to script hash. Github, 2013.

[Aso98] Nadarajah Asokan. *Fairness in electronic commerce*. PhD thesis, IBM, 1998.

[ASW98] Nadarajah Asokan, Victor Shoup, and Michael Waidner. Asynchronous protocols for optimistic fair exchange. In *Security and Privacy - 1998 IEEE Symposium on Security and Privacy, Oakland, CA, USA, May 3-6, 1998, Proceedings*, pages 86–99. IEEE Computer Society, 1998.

[AW04] Hagit Attiya and Jennifer Welch. *Distributed computing - fundamentals, simulations, and advanced topics (2. ed.)*. Wiley series on parallel and distributed computing. Wiley, 2004.

[Axi21] Axie Infinity Whitepaper, 2021.

[AZS+21] Hayden Adams, Noah Zinsmeister, Moody Salem, Rivière Keefer, and Dan Robinson. Uniswap v3 core. *Whitepaper*, 2021.

[Bac02] Adam Back. Hashcash - A Denial of Service Counter-Measure. page 10, 2002.

[BBBF18] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable Delay Functions. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018*, volume 10991, pages 757–788. Springer International Publishing, Cham, 2018.

[BCC+21] Lorenz Breidenbach, Christian Cachin, Benedict Chan, Alex Coventry, Steve Ellis, Ari Juels, Farinaz Koushanfar, Andrew Miller, Brendan Magauran, Daniel Moroz, et al. Chainlink 2.0: Next steps in the evolution of decentralized oracle networks. *Whitepaper*, 2021.

[BCD+14] Adam Back, Matt Corallo, Luke Dashjr, Mark Friedenbach, Gregory Maxwell, Andrew Miller, Andrew Poelstra, Jorge Timón, and Pieter Wuille. Enabling Blockchain Innovations with Pegged Sidechains. *Whitepaper*, page 25, 2014.

[BCJ08] Ali Bagherzandi, Jung Hee Cheon, and Stanislaw Jarecki. Multisignatures secure under the discrete logarithm assumption and a generalized forking lemma. In Peng Ning, Paul F. Syverson, and Somesh Jha, editors, *Proceedings of the 2008 ACM Conference on Computer and Communications Security, CCS 2008, Alexandria, Virginia, USA, October 27-31, 2008*, pages 449–458. ACM, 2008.

[BCL21] Massimo Bartoletti, James Hsin-yu Chiang, and Alberto Lluch-Lafuente. A theory of automated market makers in defi. In Ferruccio Damiani and Ornela Dardha, editors, *Coordination Models and Languages - 23rd IFIP WG 6.1 International Conference, COORDINATION 2021*, volume 12717 of *Lecture Notes in Computer Science*, pages 168–187. Springer, 2021.

[BG96] Mihir Bellare and Shafi Goldwasser. Encapsulated Key Escrow. Technical report, Massachusetts Institute of Technology, 1996.

[BGW+20] Dan Boneh, Sergey Gorbunov, Riad S. Wahby, Hoeteck Wee, and Zhenfei Zhang. BLS Signatures. Internet-Draft draft-irtf-cfrg-bls-signature-04, Internet Engineering Task Force, September 2020. Work in Progress.

[BKKT20] Rhys Bowden, H. Paul Keeler, Anthony E. Krzesinski, and Peter G. Taylor. Modeling and analysis of block arrival times in the bitcoin blockchain. *Stochastic Models*, 36(4):602–637, 2020.

[BKLZ20] Benedikt Bünz, Lucianna Kiffer, Loi Luu, and Mahdi Zamani. FlyClient: Super-light clients for cryptocurrencies. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 928–946, 2020.

[Bla] Blackcoin website. https://blackcoin.org/.

[BLS04] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. *Journal of Cryptology*, 17(4):297–319, 2004.

[BM16] Emanuele Bellini and Nadir Murru. An efficient and secure RSA-like cryptosystem exploiting Rédei rational functions over conics. *Finite Fields and Their Applications*, 39:179–194, May 2016.

[BMS19] Stefano Bistarelli, Ivan Mercanti, and Francesco Santini. An Analysis of Non-standard Transactions. *Frontiers in Blockchain*, 2:7, August 2019.

[BMS21] Fadi Barbàra, Nadir Murru, and Claudio Schifanella. Towards a broadcast time-lock based token exchange protocol. In *Euro-Par 2021: Parallel Processing Workshops - Euro-Par 2021 International Workshops*, Lecture Notes in Computer Science, 2021.

[BN00] Dan Boneh and Moni Naor. Timed Commitments. In Gerhard Goos, Juris Hartmanis, Jan van Leeuwen, and Mihir Bellare, editors, *Advances in Cryptology — CRYPTO 2000*, volume 1880, pages 236–254. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000.

[BN06a] Paulo S. L. M. Barreto and Michael Naehrig. Pairing-Friendly Elliptic Curves of Prime Order. In Bart Preneel and Stafford Tavares, editors, *Selected Areas in Cryptography*, Lecture Notes in Computer Science, pages 319–331, Berlin, Heidelberg, 2006. Springer.

[BN06b] Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-Key model and a general forking lemma. In *Proceedings of the 13th ACM Conference on Computer and Communications Security - CCS '06*, pages 390–399, Alexandria, Virginia, USA, 2006. ACM Press.

[Bra97] Scott O. Bradner. Key words for use in rfcs to indicate requirement levels, 1997.

[Bro09] Daniel R. L. Brown. Sec 1: Elliptic curve cryptography. *Certicom Res., Mississauga, ON, Canada, Tech. Rep. SEC2-Ver-2.0*, 2009.

[Bro10] Daniel R. L. Brown. Sec 2: Recommended elliptic curve domain parameters. *Certicom Res., Mississauga, ON, Canada, Tech. Rep. SEC2-Ver-2.0*, 2010.

[BRZ18] Jeremiah Blocki, Ling Ren, and Samson Zhou. Bandwidth-hard functions: Reductions and lower bounds. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 1820–1836. ACM, 2018.

[BS20a] Fadi Barbàra and Claudio Schifanella. DMix: decentralized mixer for unlinkability. In *2nd Conference on Blockchain Research & Applications for Innovative Networks and Services, BRAINS 2020, Paris, France, September 28-30, 2020*, pages 1–8. IEEE, 2020.

[BS20b] Fadi Barbàra and Claudio Schifanella. P2T: pay to transport. In Bartosz Balis, Dora B. Heras, Laura Antonelli, Andrea Bracciali, Thomas Gruber, Jin Hyun-Wook, Michael Kuhn, Stephen L. Scott, Didem Unat, and Roman Wyrzykowski, editors, *Euro-Par 2020: Parallel Processing Workshops - Euro-Par 2020 International Workshops, Warsaw, Poland, August 24-25, 2020, Revised Selected Papers*, volume 12480 of *Lecture Notes in Computer Science*, pages 66–77. Springer, 2020.

[BS22a] Fadi Barbàra and Claudio Schifanella. BxTB: cross-chain exchanges of bitcoins for all bitcoin wrapped tokens. In *BCCA2022*, 2022.

[BS22b] Fadi Barbàra and Claudio Schifanella. MP-HTLC: Enabling blockchain interoperability through a multiparty implementation of the htlc. *Concurrency and Computation: Practice and Experience*, 2022.

[BSS21] Fadi Barbàra, Alessandro Sanino, and Claudio Schifanella. UTS: the universal token swapper. In *3rd Workshop on Blockchain theoRy and ApplicatIoNs BRAIN 2022*, Lecture Notes in Computer Science, 2021.

[But] Vitalik Buterin. Sharding FAQ.

[But15] Vitalik Buterin. Eip-2: Homestead hard-fork changes. Github, 2015.

[But16] Vitalik Buterin. Chain interoperability. *R3 Research Paper*, 2016.

[But17] Vitalik Buterin. On path independence, 2017.

[But22] Vitalik Buterin. [AMA] we are the ef's research team (pt. 7: 07 january, 2022) (comment). https://old.reddit.com/r/ethereum/comments/rwojtk/ama_we_are_the_efs_research_team_pt_7_07_january/hrngyk8/, 2022.

[Cal22] Giulio Caldarelli. Wrapping Trust for Interoperability: A Preliminary Study of Wrapped Tokens. *Information*, 13(1):6, January 2022.

[CBC22] Panagiotis Chatzigiannis, Foteini Baldimtsi, and Konstantinos Chalkias. Sok: Blockchain light clients. 13411:615–641, 2022.

[Cho17] Andrew Chow. Bip-174: Partially signed bitcoin transaction format. Github, 2017.

[Cho21] Andrew Chow. Bip-370: Psbt version 2. Github, 2021.

[CM88] K. Mani Chandy and Jayadev Misra. Parallel program design: A foundation, 1988.

[Dam93] Ivan Damgård. Practical and provably secure release of a secret and exchange of signatures. In Tor Helleseth, editor, *Advances in Cryptology - EUROCRYPT '93, Workshop on the Theory and Application of of Cryptographic Techniques, Lofthus, Norway, May 23-27, 1993, Proceedings*, volume 765 of *Lecture Notes in Computer Science*, pages 200–217. Springer, 1993.

[DEF+19] Manu Drijvers, Kasra Edalatnejad, Bryan Ford, Eike Kiltz, Julian Loss, Gregory Neven, and Igors Stepanovs. On the Security of Two-Round Multi-Signatures. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1084–1101, May 2019.

[Des87] Yvo Desmedt. Society and group oriented cryptography: A new concept. In Carl Pomerance, editor, *Advances in Cryptology - CRYPTO '87, A Conference on the Theory and Applications of Cryptographic Techniques, Santa Barbara, California, USA, August 16-20, 1987, Proceedings*, volume 293 of *Lecture Notes in Computer Science*, pages 120–127. Springer, 1987.

[DK01] Ivan Damgård and Maciej Koprowski. Practical threshold RSA signatures without a trusted dealer. In Birgit Pfitzmann, editor, *Advances in Cryptology - EUROCRYPT 2001, International Conference on the Theory and Application of Cryptographic Techniques, Innsbruck, Austria, May 6-10, 2001, Proceeding*, volume 2045 of *Lecture Notes in Computer Science*, pages 152–165. Springer, 2001.

[DKLS18] Jack Doerner, Yashvanth Kondi, Eysa Lee, and Abhi Shelat. Secure two-party threshold ecdsa from ecdsa assumptions. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 980–997, 2018.

[DPW+17] Johnny Dilley, Andrew Poelstra, Jonathan Wilkins, Marta Piekarska, Ben Gorlick, and Mark Friedenbach. Strong Federations: An Interoperable Blockchain Solution to Centralized Third-Party Risks. *arXiv:1612.05491 [cs]*, January 2017.

[DSW14] Christian Decker, Jochen Seidel, and Roger Wattenhofer. Bitcoin Meets Strong Consistency. *arXiv:1412.7935 [cs]*, December 2014. arXiv: 1412.7935.

[DW14] Christian Decker and Roger Wattenhofer. Bitcoin transaction malleability and mtgox. In Miroslaw Kutylowski and Jaideep Vaidya, editors, *Computer Security - ESORICS 2014 - 19th European Symposium on Research in Computer Security, Wroclaw, Poland, September 7-11, 2014. Proceedings, Part II*, volume 8713 of *Lecture Notes in Computer Science*, pages 313–326. Springer, 2014.

[EJN17] Steve Ellis, Ari Juels, and Sergey Nazarov. ChainLink,A Decentralized Oracle Network, September 2017.

[EKR18] David Evans, Vladimir Kolesnikov, and Mike Rosulek. A pragmatic introduction to secure multi-party computation. *Found. Trends Priv. Secur.*, 2(2-3):70–246, 2018.

[ES18] Ittay Eyal and Emin Gün Sirer. Majority is not enough: bitcoin mining is vulnerable. *Commun. ACM*, 61(7):95–102, 2018.

[ESES18] William Entriken, Dieter Shirley, Jacob Evans, and Nastassia Sachs. Eip-721: Non-fungible token standard. Github, 2018.

[eth18] etherdeg. Etherdelta smart contract overview. Github, 2018.

[Eth21] Ethereum. Btc relay. https://github.com/ethereum/btcrelay, 2021.

[Fer18] Manuel Fersch. *The provable security of elgamal-type signature schemes.* doctoralthesis, Ruhr-Universität Bochum, Universitätsbibliothek, 2018.

[FHSS+22] Armando Faz-Hernández, Sam Scott, Nick Sullivan, Riad S. Wahby, and Christopher A. Wood. Hashing to Elliptic Curves. Internet-Draft draft-irtf-cfrg-hash-to-curve-14, Internet Engineering Task Force, February 2022. Work in Progress.

[FHZ+19] Qi Feng, Debiao He, Sherali Zeadally, Muhammad Khurram Khan, and Neeraj Kumar. A survey on privacy protection in blockchain system. *J. Netw. Comput. Appl.*, 126:45–58, 2019.

[FLP85] Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. Impossibility of Distributed Consensus with One Faulty Process. *J. ACM*, 32(2):374–382, 1985.

[Fou22a] Ethereum Foundation. Ethash: A lightweight proof-of-work algorithm. Github, 2022.

[Fou22b] Ethereum Foundation. Ethereum Proof-of-Stake Consensus Specifications. ethereum, June 2022.

[GG18] Rosario Gennaro and Steven Goldfeder. Fast multiparty threshold ECDSA with fast trustless setup. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 1179–1194, address, 2018. organization, ACM.

[GGV20] Sanjam Garg, Shafi Goldwasser, and Prashant Nalini Vasudevan. Formalizing data deletion in the context of the right to be forgotten. In *Advances in Cryptology – EUROCRYPT 2020*, pages 373–402. Springer International Publishing, 2020.

[GJKR01] Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. Robust threshold dss signatures. *Information and Computation*, 164(1):54–84, 2001.

[GJKR06] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. *Journal of Cryptology*, 20:51–83, 2006.

[GKL15] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 281–310. Springer, 2015.

[Gol01] Oded Goldreich. *The Foundations of Cryptography - Volume 1: Basic Techniques.* Cambridge University Press, 2001.

[Gol04] Oded Goldreich. *The Foundations of Cryptography - Volume 2: Basic Applications.* Cambridge University Press, 2004.

[GSM+20] Sarada Prasad Gochhayat, Sachin Shetty, Ravi Mukkamala, Peter Foytik, Georges A. Kamhoua, and Laurent Njilla. Measuring Decentrality in Blockchain Based Systems. *IEEE Access*, 8:178372–178390, 2020.

[HAB+17] Ethan Heilman, Leen Alshenibr, Foteini Baldimtsi, Alessandra Scafuro, and Sharon Goldberg. Tumblebit: An untrusted bitcoin-compatible anonymous payment hub. In *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*. The Internet Society, 2017.

[HBB17] Eyal Hertzog, Guy Benartzi, and Galia Benartzi. Bancor protocol. *Whitepaper*, 2017.

[Hea05] C. M. Heard. Guidelines for authors and reviewers of MIB documents, 2005.

[HHW18] Yung-Chen Hsieh, Chih-Wen Hsueh, and Ja-Ling Wu. The exchange center: A case study of hybrid decentralized and centralized applications in blockchain. In *2018 1st IEEE International Conference on Hot Information-Centric Networking (HotICN)*, 2018.

[HKZG15] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. Eclipse attacks on Bitcoin's Peer-to-Peer network. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 129–144, Washington, D.C., August 2015. USENIX Association.

[HLG20] Ethan Heilman, Sebastien Lipmann, and Sharon Goldberg. The arwen trading protocols. In Joseph Bonneau and Nadia Heninger, editors, *Financial Cryptography and Data Security - 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10-14, 2020 Revised Selected Papers*, volume 12059 of *Lecture Notes in Computer Science*, pages 156–173. Springer, 2020.

[Hom] Qtum webpage. https://qtum.org/en.

[Hop22] Daira Hopwood. Zcash/zips. Technical report, Zcash, aug 2022.

[HS18] Haya R. Hasan and Khaled Salah. Blockchain-based solution for proof of delivery of physical assets. In *Lecture Notes in Computer Science*, pages 139–152. Springer International Publishing, 2018.

[HTC+21] Yuming Huang, Jing Tang, Qianhao Cong, Andrew Lim, and Jianliang Xu. Do the rich get richer? Fairness analysis for blockchain incentives. In Guoliang Li, Zhanhuai Li, Stratos Idreos, and Divesh Srivastava, editors, *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*, pages 790–803. ACM, 2021.

[HYL+21] Runchao Han, Jiangshan Yu, Haoyu Lin, Shiping Chen, and Paulo Jorge Esteves Veríssimo. On the security and performance of blockchain sharding, 2021.

[Int22] Sushiswap documentation. https://docs.sushi.com//docs/intro, 2022.

[iT21] 1inch Team. The 1Inch API v3.0, 2021.

[JMV01] Don Johnson, Alfred Menezes, and Scott Vanstone. The elliptic curve digital signature algorithm (ECDSA). *Int. J. Inf. Secur.*, 1(1):36–63, aug 2001.

[JSK19] Hussam Juma, Khaled Shaalan, and Ibrahim Kamel. A survey on using blockchain in trade supply chain solutions. *IEEE Access*, 2019.

[JSZ+19] Aljosha Judmayer, Nicholas Stifter, Alexei Zamyatin, Itay Tsabary, Ittay Eyal, Peter Gazi, Sarah Meiklejohn, and Edgar Weippl. Pay to win: Cheap, crowdfundable, cross-chain algorithmic incentive manipulation attacks on pow cryptocurrencies. Technical report, Cryptology ePrint Archive, address, 2019.

[Kel20] Marcel Keller. MP-SPDZ: A versatile framework for multi-party computation. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, address, 2020. organization.

[KKM14] Philip Koshy, Diana Koshy, and Patrick D. McDaniel. An analysis of anonymity in bitcoin using P2P network traffic. In Nicolas Christin and Reihaneh Safavi-Naini, editors, *Financial Cryptography and Data Security - 18th International Conference, FC 2014, Christ Church, Barbados, March 3-7, 2014, Revised Selected Papers*, volume 8437 of *Lecture Notes in Computer Science*, pages 469–485. Springer, 2014.

[KKS+17] Yujin Kwon, Dohyun Kim, Yunmok Son, Eugene Y. Vasserman, and Yongdae Kim. Be selfish and avoid dilemmas: Fork after withholding (FAW) attacks on bitcoin. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 195–209, address, 2017. organization, ACM.

[KKS22] Christiane Kuhn, Aniket Kate, and Thorsten Strufe. The danger of small anonymity sets in privacy-preserving payment systems. *CoRR*, abs/2204.09282, 2022.

[KLS16] Aggelos Kiayias, Nikolaos Lamprou, and Aikaterini-Panagiota Stouka. Proofs of proofs of work with sublinear complexity. In Jeremy Clark, Sarah Meiklejohn, Peter Y.A. Ryan, Dan Wallach, Michael Brenner, and Kurt Rohloff, editors, *Financial Cryptography and Data Security*, pages 61–78, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[KMP16] Eike Kiltz, Daniel Masny, and Jiaxin Pan. Optimal security proofs for signatures from identification schemes. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*, volume 9815 of *Lecture Notes in Computer Science*, pages 33–61. Springer, 2016.

[KN12] Sunny King and Scott Nadal. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. https://www.peercoin.net/read/papers/peercoin-paper.pdf, 2012.

[KYMM18] George Kappos, Haaroon Yousaf, M. Maller, and S. Meiklejohn. An Empirical Analysis of Anonymity in Zcash. In *USENIX Security Symposium*, 2018.

[Lam89] Leslie Lamport. A simple approach to specifying concurrent systems. *Commun. ACM*, 32(1):32–45, 1989.

[Lan96] Susan K. Langford. Weakness in some threshold cryptosystems. In Neal Koblitz, editor, *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, volume 1109 of *Lecture Notes in Computer Science*, pages 74–82. Springer, 1996.

[LBC+19] Lindsay X Lin, Eric Budish, Lin William Cong, Zhiguo He, Jonatan H Bergquist, Mohit Singh Panesir, Jack Kelly, Michelle Lauer, Ryan Prinster, Stephenie Zhang, et al. Deconstructing decentralized exchanges. *Stanford Journal of Blockchain Law & Policy*, 2, 2019.

[Lin17] Yehuda Lindell. Fast secure two-party ECDSA signing. In *CRYPTO*, 2017.

[LJKW18] Jia Liu, Tibor Jager, Saqib A. Kakvi, and Bogdan Warinschi. How to build time-lock encryption. *Des. Codes Cryptogr.*, 86(11):2549–2586, 2018.

[LLN+22] Yulin Liu, Yuxuan Lu, Kartik Nayak, Fan Zhang, Luyao Zhang, and Yinhong Zhao. Empirical Analysis of EIP-1559: Transaction Fees, Waiting Time, and Consensus Security. *arXiv:2201.05574 [cs, econ, q-fin]*, January 2022. arXiv: 2201.05574.

[LNZ+16] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. A Secure Sharding Protocol For Open Blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 17–30, Vienna Austria, October 2016. ACM.

[LP01] Anna Lysyanskaya and Chris Peikert. Adaptive security in the threshold setting: From cryptosystems to signature schemes. In Colin Boyd, editor, *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings*, volume 2248 of *Lecture Notes in Computer Science*, pages 331–350. Springer, 2001.

[LSP82] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The Byzantine Generals Problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

[LW16a] Johnson Lau and Pieter Wuille. Bip-143: Transaction signature verification for version 0 witness program. Github, 2016.

[LW16b] Johnson Lau and Pieter Wuille. Bip-146: Dealing with signature encoding malleability. GitHub, 2016.

[Mat20a] Matt. Eip-3005: Batched meta transactions [draft]. Github, 2020.

[Mat20b] Matt. Native meta-transaction proposal roundup. EthResear.ch, 2020.

[Max13] Gregory Maxwell. CoinJoin: Bitcoin privacy for the real world. https://bitcointalk.org/?topic=279249, August 2013.

[May93] Timothy May. Timed-release crypto. https://cypherpunks.venona.com/date /1993/02/msg00129.html, 1993.

[MH96] Markus Michels and Patrick Horster. On the risk of disruption in several multiparty signature schemes. In Kwangjo Kim and Tsutomu Matsumoto, editors, *Advances in Cryptology - ASIACRYPT '96, International Conference on the Theory and Applications of Cryptology and Information Security, Kyongju, Korea, November 3-7, 1996, Proceedings*, volume 1163 of *Lecture Notes in Computer Science*, pages 334–345. Springer, 1996.

[MN22] Malte Möser and Arvind Narayanan. Resurrecting address clustering in bitcoin. 13411:386–403, 2022.

[MNF17] Felix Konstantin Maurer, Till Neudecker, and Martin Florian. Anonymous CoinJoin Transactions with Arbitrary Values. In *2017 IEEE Trustcom/BigDataSE/ICESS*, pages 522–529, Sydney, Australia, August 2017. IEEE.

[MO15] Sarah Meiklejohn and Claudio Orlandi. Privacy-Enhancing Overlays in Bitcoin. In Michael Brenner, Nicolas Christin, Benjamin Johnson, and Kurt Rohloff, editors, *Financial Cryptography and Data Security*, volume 8976, pages 127–141. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.

[MOR01] Silvio Micali, Kazuo Ohta, and Leonid Reyzin. Accountable-subgroup multisignatures: extended abstract. In Michael K. Reiter and Pierangela Samarati, editors, *CCS 2001, Proceedings of the 8th ACM Conference on Computer and Communications Security, Philadelphia, Pennsylvania, USA, November 6-8, 2001*, pages 245–254. ACM, 2001.

[MPJ+13] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M. Voelker, and Stefan Savage. A fistful of bitcoins: Characterizing payments among men with no names. In *Proceedings of the 2013 Conference on Internet Measurement Conference - IMC '13*, pages 127–140, Barcelona, Spain, 2013. ACM Press.

[MPSW19] Gregory Maxwell, Andrew Poelstra, Yannick Seurin, and Pieter Wuille. Simple Schnorr multi-signatures with applications to Bitcoin. *Designs, Codes and Cryptography*, 87(9):2139–2164, September 2019.

[MR97] Dahlia Malkhi and Michael K. Reiter. Byzantine quorum systems. In Frank Thomson Leighton and Peter W. Shor, editors, *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, May 4-6, 1997*, pages 569–578, address, 1997. organization, ACM.

[MSH+18] Malte Möser, Kyle Soska, Ethan Heilman, Kevin Lee, Henry Heffan, Shashvat Srivastava, Kyle Hogan, Jason Hennessey, Andrew Miller, Arvind Narayanan, and Nicolas Christin. An Empirical Analysis of Traceability in the Monero Blockchain. *arXiv:1704.04299 [cs]*, April 2018.

[MT19] Giulio Malavolta and Sri Aravinda Krishnan Thyagarajan. Homomorphic Time-Lock Puzzles and Applications. In *Advances in Cryptology – CRYPTO 2019*, volume 11692, pages 620–649. Springer International Publishing, Cham, 2019.

[MWLD10] Changshe Ma, Jian Weng, Yingjiu Li, and Robert H. Deng. Efficient discrete logarithm based multi-signature scheme in the plain public key model. *Des. Codes Cryptogr.*, 54(2):121–133, 2010.

[Nak08] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. *Whitepaper*, page 9, 2008.

[Nes19] Mark Nesbitt. Deep chain reorganization detected on ethereum classic (ETC). https://blog.coinbase.com/ethereum-classic-etc-is-currently-being-51-attacked-33be13ce32de, 2019.

[NIP19] Kyber Network, BitGo Inc, and Republic Protocol. Wbtc Whitepaper v0.2. https://wbtc.network/assets/wrapped-tokens-whitepaper.pdf, 2019.

[NKDM03] Antonio Nicolosi, Maxwell N. Krohn, Yevgeniy Dodis, and David Mazières. Proactive two-party signatures for user authentication. In *NDSS*, 2003.

[NKMS16] Kartik Nayak, Srijan Kumar, Andrew Miller, and Elaine Shi. Stubborn mining: Generalizing selfish mining and combining with an eclipse attack. In *IEEE European Symposium on Security and Privacy, EuroS&P 2016, Saarbrücken, Germany, March 21-24, 2016*, pages 305–320, address, 2016. organization, IEEE.

[NKW21] Tejaswi Nadahalli, Majid Khabbazian, and Roger Wattenhofer. Timelocked bribing. In Nikita Borisov and Claudia Díaz, editors, *Financial Cryptography and Data Security - 25th International Conference, FC 2021, Virtual Event, March 1-5, 2021, Revised Selected Papers, Part I*, volume 12674 of *Lecture Notes in Computer Science*, pages 53–72. Springer, 2021.

[NMRP19] Michael Neuder, Daniel J. Moroz, Rithvik Rao, and David C. Parkes. Selfish behavior in the tezos proof-of-stake protocol. *CoRR*, abs/1912.02954, 2019.

[Nol13] Tier Nolan. Alt chains and atomic transfers. https://bitcointalk.org/index.php?topic=193281.msg2224949, 2013.

[NRS] Jonas Nick, Tim Ruffing, and Yannick Seurin. MuSig2: Simple Two-Round Schnorr Multi-Signatures. page 39.

[NRS20] Kevin Alarcón Negy, Peter R. Rizun, and Emin Gün Sirer. Selfish Mining Re-Examined. In Joseph Bonneau and Nadia Heninger, editors, *Financial Cryptography and Data Security - 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10-14, 2020 Revised Selected Papers*, volume 12059 of *Lecture Notes in Computer Science*, pages 61–78. Springer, 2020.

[NRSW20] Jonas Nick, Tim Ruffing, Yannick Seurin, and Pieter Wuille. MuSig-DN: Schnorr Multi-Signatures with Verifiably Deterministic Nonces. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 1717–1731, Virtual Event USA, October 2020. ACM.

[NSW09] Gregory Neven, Nigel P. Smart, and Bogdan Warinschi. Hash function requirements for Schnorr signatures. *Journal of Mathematical Cryptology*, 3(1), January 2009.

[Nxt] Nxt documentation. https://nxtdocs.jelurida.com/Getting_started.

[Pad20] Rafael Padilla. Defi, law and regulation. Technical report, Mimeo, 2020.

[Pal17] Santiago Palladino. The parity wallet hack explained, 2017.

[PB17] Joseph Poon and Vitalik Buterin. Plasma: Scalable autonomous smart contracts. *White paper*, 2017.

[PG99] Henning Pagnia and Felix C Gartner. On the impossibility of fair exchange without a trusted third party. Technical report, Citeseer, address, 1999.

[Pie18] Krzysztof Pietrzak. Simple verifiable delay functions. In *10th innovations in theoretical computer science conference (itcs 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[Pnu77] Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 46–57. ieee, 1977.

[Poe13] Bertram Poettering. Deterministic usage of the digital signature algorithm (dsa) and elliptic curve digital signature algorithm (ecdsa). RFC 6979, 2013.

[Pra18] Pragmatic signature aggregation with BLS - Sharding. https://ethresear.ch/t/pragmatic-signature-aggregation-with-bls/2105, May 2018.

[Pro] Proto-Danksharding FAQ. https://notes.ethereum.org/@vbuterin/proto_danksharding_faq#What-is-Danksharding.

[PS00a] David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3):361–396, 2000.

[PS00b] Guillaume Poupard and Jacques Stern. Short proofs of knowledge for factoring. In Hideki Imai and Yuliang Zheng, editors, *Public Key Cryptography, Third International Workshop on Practice and Theory in Public Key Cryptography, PKC 2000, Melbourne, Victoria, Australia, January 18-20, 2000, Proceedings*, volume 1751 of *Lecture Notes in Computer Science*, pages 147–166. Springer, 2000.

[PW20] Ross Pure and Zian-Loong Wang. RenVM secure multiparty computation. https://github.com/renproject/rzl-mpc-specification/blob/master/z0_spec.pdf, 2020.

[Qua11] QuantumMechanic. Proof of stake instead of proof of work, 2011.

[RMK14] Tim Ruffing, Pedro Moreno-Sanchez, and Aniket Kate. CoinShuffle: Practical Decentralized Coin Mixing for Bitcoin. In Mirosław Kutyłowski and Jaideep Vaidya, editors, *Computer Security - ESORICS 2014*, volume 8713, pages 345–364. Springer International Publishing, Cham, 2014.

[RSW96] Ronald L Rivest, Adi Shamir, and David A Wagner. Time-lock puzzles and timed-release Crypto. Technical report, 1996.

[Saf] Gnosis Safe. https://gnosis-safe.io.

[SBFG20] Ashish Rajendra Sai, Jim Buckley, Brian Fitzgerald, and Andrew Le Gear. Taxonomy of Centralization in Public Blockchain Systems: A Systematic Literature Review. *arXiv:2009.12542 [cs]*, September 2020. arXiv: 2009.12542.

[Sch91] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, 1991.

[SD17] Emin Gün Sirer and Phil Daian. Bancor is flawed, 2017.

[Sho00] Victor Shoup. Practical threshold signatures. In Bart Preneel, editor, *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding*, volume 1807 of *Lecture Notes in Computer Science*, pages 207–220. Springer, 2000.

[SSZ16] Ayelet Sapirshtein, Yonatan Sompolinsky, and Aviv Zohar. Optimal selfish mining strategies in bitcoin. In Jens Grossklags and Bart Preneel, editors, *Financial Cryptography and Data Security - 20th International Conference, FC 2016, Christ Church, Barbados, February 22-26, 2016, Revised Selected Papers*, volume 9603 of *Lecture Notes in Computer Science*, pages 515–532, address, 2016. organization, Springer.

[ST19] Kuheli Sai and David Tipper. Disincentivizing double spend attacks across interoperable blockchains. In *First IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications, TPS-ISA 2019, Los Angeles, CA, USA, December 12-14, 2019*, pages 36–45, address, 2019. organization, IEEE.

[STV+16] Ewa Syta, Iulia Tamas, Dylan Visher, David Isaac Wolinsky, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ismail Khoffi, and Bryan Ford. Keeping authorities "honest or bust" with decentralized witness cosigning. In *IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016*, pages 526–545. IEEE Computer Society, 2016.

[Sza97] Nick Szabo. Formalizing and securing relationships on public networks. *First monday*, 1997.

[Sza01] Nick Szabo. Trusted third parties are security holes. Nakamoto Institute, 2001.

[Szi17] Péter Szilágyi. Eip-225: Clique proof-of-authority consensus protocol. Github, 2017.

[TD18] Peter K. Todd and Zindros Dyonis. Merkle mountain ranges, December 2018.

[Tea20] Matcha Team. The Matcha documentation, 2020.

[Tea22] Synthetix Team. Synthetix litepaper. https://docs.synthetix.io/litepaper/, 2022.

[TMEJ19] Tin Tironsakkul, Manuel Maarek, Andrea Eross, and Mike Just. Probing the mystery of cryptocurrency theft: An investigation into methods for cryptocurrency tainting analysis. *CoRR*, abs/1906.05754, 2019.

[Tod14] Peter Todd. OP_CHECKLOCKTIMEVERIFY. https://github.com/bitcoin/bips/blob/master/bip-0065.mediawiki, 2014.

[TYME21] Itay Tsabary, Matan Yechieli, Alex Manuskin, and Ittay Eyal. MAD-HTLC: because HTLC is crazy-cheap to attack. pages 1230–1248, 2021.

[UA21] Ryosuke Ushida and James Angel. Regulatory considerations on centralized aspects of defi managed by daos. In Matthew Bernhard, Andrea Bracciali, Lewis Gudgeon, Thomas Haines, Ariah Klages-Mundt, Shin'ichiro Matsuo, Daniel Perez, Massimiliano Sala, and Sam Werner, editors, *Financial Cryptography and Data Security. FC 2021 International Workshops*, pages 21–36, Berlin, Heidelberg, 2021. Springer Berlin Heidelberg.

[VB15] Fabian Vogelsteller and Vitalik Buterin. Eip-20: Token standard. Github, 2015.

[Vic20] Friedhelm Victor. Address Clustering Heuristics for Ethereum. In Joseph Bonneau and Nadia Heninger, editors, *Financial Cryptography and Data Security*, volume 12059, pages 617–633. Springer International Publishing, Cham, 2020.

[Vog06] Werner Vogels. Life is not a State-Machine, August 2006.

[VS13] Nicolas Van Saberhagen. Cryptonote v 2.0, 2013.

[Wan20] Yongge Wang. Automated market makers for decentralized finance (defi). *CoRR*, abs/2009.01676, 2020.

[Wes20] Benjamin Wesolowski. Efficient verifiable delay functions. *J. Cryptol.*, 33(4):2113–2147, 2020.

[WHF19] Fredrik Winzer, Benjamin Herd, and Sebastian Faust. Temporary censorship attacks in the presence of rational miners. In *2019 IEEE European Symposium on Security and Privacy Workshops, EuroS&P Workshops 2019, Stockholm, Sweden, June 17-19, 2019*, pages 357–366, address, 2019. organization, IEEE.

[WNR20] Pieter Wuille, Jonas Nick, and Tim Ruffing. Bip-340: Schnorr signatures for secp256k1. GitHub, 2020.

[WNT20] Pieter Wuille, Jonas Nick, and Anthony Towns. Bip-341: Taproot: Segwit version 1 spending rules. GitHub, 2020.

[WPG+21] Sam M. Werner, Daniel Perez, Lewis Gudgeon, Ariah Klages-Mundt, Dominik Harz, and William J. Knottenbelt. Sok: Decentralized finance (defi). *CoRR*, abs/2101.08778, 2021.

[Wro21] Christoph Wronka. Financial crime in the decentralized finance ecosystem: new challenges for compliance. *Journal of Financial Crime*, 2021.

[Wui16] Peter Wuille. Segregated witness benefits. https://bitcoincore.org/en/2016/01/26/segwit-benefits/, 2016.

[WZL+21] Yibo Wang, Qi Zhang, Kai Li, Yuzhe Tang, Jiaqi Chen, Xiapu Luo, and Ting Chen. ibatch: saving ethereum fees via secure and cost-effective batching of smart-contract invocations. In Diomidis Spinellis, Georgios Gousios, Marsha Chechik, and Massimiliano Di Penta, editors, *ESEC/FSE '21: 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Athens, Greece, August 23-28, 2021*, pages 566–577, address, 2021. organization, ACM.

[XAD21] Jiahua Xu, Damien Ackerer, and Alevtina Dubovitskaya. A game-theoretic analysis of cross-chain atomic swaps with htlcs. In *41st IEEE International Conference on Distributed Computing Systems, ICDCS 2021, Washington DC, USA, July 7-10, 2021*, pages 584–594. IEEE, 2021.

[XVP+21] Jiahua Xu, Nazariy Vavryk, Krzysztof Paruch, Simon Cousaert, et al. Sok: Automated market maker (amm) based decentralized exchanges (dexs). Technical report, 2021.

[ZAZ+21] Alexei Zamyatin, Mustafa Al-Bassam, Dionysis Zindros, Eleftherios Kokoris-Kogias, Pedro Moreno-Sanchez, Aggelos Kiayias, and William J. Knottenbelt. Sok: Communication across distributed ledgers. 12675:3–36, 2021.

[Zil17] Team Zilliqa. Zilliqa Whitepaper. Technical report, 2017.

[Zol20] Micah Zoltu. Eip-2711: Sponsored, expiring and batch transactions. [draft]. Github, 2020.