TÉCNICO LISBOA

Trustable Blockchain Interoperability: Securing Asset Transfers involving Permissioned Blockchains

Catarina Guerreiro Gomes Pedreira

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisors: Prof. André Ferreira Ferrão Couto e Vasconcelos
Prof. Miguel Ângelo Marques de Matos

Examination Committee

Chairperson: Prof. Paolo Romano
Supervisor: Prof. André Ferreira Ferrão Couto e Vasconcelos
Member of the Committee: Prof. João Fernando Peixoto Ferreira

November 2021

This work was created using LaTeX typesetting language
in the Overleaf environment (www.overleaf.com).

Acknowledgments

Without the guidance, contributions, patience and advice of the people described below, this thesis would not be possible.

I would like to begin by thanking my thesis advisors, Professor André Vasconcelos and Professor Miguel Matos, for their constant support, respect, valuable guidance and encouragement throughout this past year. For the immense insights and their trust in my way of working.

My endless gratitude goes to Rafael Belchior, who helped me make sense of an initial confusion and always went above and beyond to provide support, extensive knowledge, guidance and understanding. I could not imagine having had a better mentor.

My deep appreciation for everyone in the Hyperledger Cactus and Polkadot open-source communities who offered to help in this work. A special thanks to Peter Somogyvari for his unlimited patience, insight and support.

My sincere thanks go to my parents and brothers for their friendship, encouragement and caring for all these years, for always being there for me through thick and thin and without whom this project would not be possible. I would also like to thank my grandparents, aunts, uncles and cousins for their understanding and support throughout all these years.

I want to express my special gratitude to my boyfriend, Gonçalo Batalha, for all his unlimited love and encouragement, never once ceasing to believe in me. Thank you.

To my high-school teachers Carlos Jesus and Catarina Rijo, for seeing potential in me and helping me to change and do better. Without you, I would not be where I am today.

Thank you to Joana Ceia, Margarida Pardal, Inês Messias, Beatriz Magalhães, Joana Francisco, Clara Vieira, and Sara Alves for being my close friends ever since I can remember, and for supporting me through everything in life.

To all my friends and Instituto Superior Técnico colleagues Inês Albano, Viviana Bernardo, Carolina Carreira, Rafael Soares, Miguel Barros, Miguel Oliveira, Miguel Silveira, Joana Coutinho, Pedro Rodrigues, Sílvia Urmal, Ana Betiol, Catarina Lourenço, Matilde Pereira, Inês Alves, Daniel Cordeiro - college would not have been nearly as good without you.

To each and every one of you, and the dear reader, I dedicate this thesis.

Abstract

The blockchain technology has been drawing a great deal of attention since its arrival in 2008, with Bitcoin. It does not come as a surprise that this technology has an enormous potential to be applied in a vast number of areas. However, currently blockchains exist in silos, often competing when they could be cooperating and communicating. Interoperability is essential to allow for communication between blockchains and thus motivate mass adoption. In permissioned blockchains, interoperability is harder given their opaque nature. Some solutions have already been created in this context, however many require a trusted private third party, which may be insecure and it is not ideal given the nature of the technology.

In this work, we propose T-ODAP (*Trustless Open Digital Asset Protocol*), a secure multi-layered protocol that enables a trustless solution for permissioned blockchain interoperability. T-ODAP is more secure than other, centralized, solutions given that it eliminates the need for trust in the protocol's participants. It provides a Decentralized View Storage (DVS), a Polkadot Connector that connects permissioned blockchains to the latter, and a trustless version of the *Open Digital Asset Protocol* which leverages the DVS and the connector. The protocol models the participants as rational agents and implements game theory techniques in order to punish them in case they deviate the protocol. T-ODAP is implemented using *Polkadot* and *Hyperledger Cactus*. We tested that the implemented solution worked properly. In the theoretical evaluation, we were able to evaluate the full system's robustness in face of attacks and concluded that the system is resilient to attacks, having the same robustness level as Hashed Time-Lock Contract (HTLC)-based payment schemes.

Keywords

Blockchain; Interoperability; Permissioned; Game Theory; Security; Trustless.

Resumo

A tecnologia blockchain tem chamado atenção desde a sua chegada em 2008, com o Bitcoin. Não é novidade que a tecnologia tem um enorme potencial para ser aplicada num vasto número de áreas. No entanto, atualmente muitas blockchains existem em isolamento, várias vezes competindo quando poderiam estar a cooperar e a comunicar. A interoperabilidade é essencial para permitir a comunicação entre blockchains, motivando uma adoção em massa. Em blockchains com permissões, a interoperabilidade é mais complicada devido à sua natureza opaca. Algumas soluções já foram criadas neste sentido, porém muitas requerem uma entidade externa confiável, o que pode ser inseguro e não é ideal devido à natureza da tecnologia. Neste trabalho, propomos o T-ODAP (*Trustless Open Digital Asset Protocol*), um protocolo multi-camadas seguro que possibilita uma solução *trustless* para interoperabilidade de blockchain com permissões. O T-ODAP é mais seguro do que outras soluções centralizadas, uma vez que elimina a necessidade de confiança nos participantes do protocolo. O protocolo fornece um Armazenamento de Vistas Descentralizado (DVS), um Conector Polkadot que conecta blockchains com permissões a este último e uma versão *trustless* do *Open Digital Asset Protocol* que utiliza o DVS e o conector. O protocolo modela os participantes como agentes racionais e implementa técnicas de teoria dos jogos para puni-los caso estes se desviem do protocolo. O T-ODAP é implementado recorrendo ao *Polkadot* e ao *Hyperledger Cactus*. Testámos a solução implementada. Na avaliação teórica, avaliámos a robustez do sistema no que toca a ataques e concluímos que o T-ODAP é resistente a ataques, tendo o mesmo nível de robustez de esquemas de pagamento baseados em Hashed Time-Lock Contract (HTLC).

Palavras Chave

Blockchain; Interoperabilidade; Permissionada; Teoria de Jogos; Segurança; Trustless

Contents

List of Figures

List of Tables

Acronyms

ODAP	Open Digital Asset Protocol
T-ODAP	Trustless Open Digital Asset Protocol
WASM	Web-Assembly
ABI	Application Binary Interface
DVS	Decentralized View Storage
DSL	Domain Specific Language
HTLC	Hashed Time-Lock Contract
WASM	WebAssembly

1

Introduction

Contents

Blockchains are becoming more and more relevant in today's world as they have been proven to have the potential to revolutionize applications and redefine the digital economy [7]. While the first blockchain to ever exist - Bitcoin [8] - has economic purposes, many other use cases for the technology, such as healthcare support [9], have emerged over time.

A blockchain is a distributed, immutable ledger that stores transactions containing application dependent data. The participants contribute to the functioning of the system, distributing the trust among all and thus avoiding a *single point of failure*. A blockchain that places restrictions on who its participants are, only allows them to perform certain actions and is controlled by a node or group of nodes (committee) is considered to be permissioned (or private), while a blockchain that allows anyone to join and contribute to the network is permissionless (or public) [10].

There are currently a large number of blockchain projects that encompass different characteristics - such as the *consensus* mechanism or hashing algorithms. They may also specialize in very distinct areas, ranging from work such as SSIBAC (Self-Sovereign Identity Based Access Control) [11], that leverages a permissioned blockchain to provide users with an authentication based on self-sovereign identity, to projects such as CryptoKitties [12], a game built on top of a public blockchain. In this context, organizations are able to choose from a wide range of options. However, this is a delicate choice, since it is hard to learn the technology and expensive to invest in it [3].

Given the high number of specialized blockchains, there is the tendency to deploy them *siloed* from each other, preventing the technology from reaching its full potential - according to Hardjono et al. [13], the situation that exists today is similar to the one that we saw back in the 1970's and 1980's with the Internet. Multiple Local Area Network (LAN) systems existed, however they had distinct approaches and thus were isolated. Once the Internet was able to achieve interoperability, connecting the small networks with each other, it became resilient and a commercial success. In parallel, the authors state that in order for blockchain to survive and thrive, playing a fundamental role in the future, it requires an architecture that satisfies the same goals of the Internet architecture.

Thus, blockchain *interoperability* is of utmost importance, since it allows risk reduction by enabling migration across different blockchains. This way, once a blockchain becomes obsolete, it is possible to replace it. Additionally, *interoperability* enables the creation of new use cases, exploiting synergies between different solutions and scaling of existing ones [3], potentially fostering the technology's adoption.

Both permissioned and permissionless blockchains require blockchain *interoperability*. However, as said before, the latter have structural differences between them - in a simplistic way, one blockchain type is closed while the other is open. As such, interoperability looks different for each of the types.

For permissionless blockchains, there are several emerging solutions that provide interoperability while still maintaining the decentralized aspect that the technology defends, such as XCLAIM [14]. This is challenging to achieve, but still feasible due to the open nature of these blockchains. In order to have

blockchain *interoperability*, blockchains need to communicate with each other, i.e. they need to share their states with each other. In the context of permissionless blockchains, it is easier to validate a given blockchain state due to the very fact that it is public.

When it comes to permissioned blockchains, however, the situation is different. In general, these blockchains are opaque and thus it is against their nature to share internal states with the external world. This is challenging to solve - in order to know the internal state of a blockchain of this type, we have to take the word of at least one node belonging to the latter and the state we obtain might be incorrect if we are dealing with malicious nodes. Some *interoperability* solutions have also been arising for permissioned blockchains, yet most are centralized which may not be completely secure and goes against the decentralized nature of the blockchain technology.

This type of interoperability is very relevant given the fact that it enables new use cases leveraging permissioned blockchains, such as cross-border asset transfers between banks. These are, in general, still a very inconvenient form of payment given the high transaction fees, the lack of transparency and the high latency (usually around 2-3 days). In this scenario, with permissioned blockchain interoperability, each bank could be associated to a permissioned blockchain (given that a bank's data can not be public) and be able to transfer assets from one blockchain to the other in a much faster, cheaper and secure way. Moreover, in this context, the interoperability mechanism should be trustless for a more secure solution - the less we have to place trust on centralized intermediaries, the better, given that we are dealing with sensitive information.

Open Digital Asset Protocol (ODAP) is an example of a recent interoperability solution. It is a cross-communication protocol that operates between two gateway devices to transfer assets between blockchains represented by those gateways. This asset transfer is unidirectional and comparable to atomic swaps, where an asset is *locked* on one blockchain and it's representation is created on another [6].

A more decentralized, trustless and secure solution for permissioned blockchains' *interoperability* is needed. Thus, we propose T-ODAP, a multi-layered secure solution for cross-chain asset transfers with a focus on permissioned blockchains. In the first layer, T-ODAP encompasses a trustless system that performs the publication of permissioned blockchain's internal state proofs in a Decentralized View Storage (DVS), implemented in Polkadot [15]. The second layer comprises a connector built in Hyperledger Cactus [16], that is compatible with several permissioned blockchains and can connect the latter to the DVS. Hyperledger Cactus and Polkadot are interoperability mechanisms introduced in Section 2.3. Finally, the third layer entails the use of the DVS and state proofs to build a more trustless and secure version of the ODAP protocol. In order to model the behavior of the protocol's participants, we used game theory techniques. Please note that the third layer consists of a theoretical model and was not

implemented yet due to circumstances outside of our control; the implementation is intended for future work.

T-ODAP's biggest focus is providing a trustless, more secure solution than others that currently exist for permissioned blockchain interoperability. This entails a cost and complexity trade-off, which we are willing to accept as long as our work's objectives are met.

Finally, we present a theoretical evaluation for T-ODAP. We evaluate the full system's robustness in face of attacks and conclude that the system is *(k,t)-weak-robust*, similarly to mechanisms such as HTLC-based payment schemes or side-chain protocols [4].

We tested the correct functioning of the first two layers of T-ODAP through Hyperledger Cactus [16], which (among its other functionalities) enables blockchain and smart contract testing. We also presented the metrics we would have evaluated if we had had the opportunity, as well as expressing our predictions for the results to expect, in relation to ODAP as our baseline.

1.1 Work Objectives

The main goal of our work is to provide a secure and robust system that allows for trustless permissioned blockchain interoperability through the use of the DVS. The DVS is implemented in the form of a Polkadot smart contract and the connector is implemented in Hyperledger Cactus. The implementation of the theoretical model (i.e. the adaptation of the ODAP protocol) is intended for future work.

The following research questions should be tackled by our solution:

1. How to guarantee the internal state proofs' correctness and integrity if permissioned blockchains are opaque?
2. How can we effectively model the dynamics of the protocol in regards to its rational participants, using game theory?
3. How to make T-ODAP strongly robust in terms of resilience to attacks?

1.2 Organization of the Document

This thesis is is organized as follows: Chapter 2 provides background on concepts fundamental to the blockchain technology, blockchain interoperability and game theory. Additionally, the chapter presents related work that describes solutions that tie in with the work proposed in this document. Then, Chapter 3 presents requirements, assumptions and preliminaries on Trustless Open Digital Asset Protocol (T-ODAP). It also discusses the system model, technologies leveraged, the protocol and the threat model for T-ODAP. Finally, it presents a theoretical use case for the solution. Next, Chapter 4 presents

the technologies leveraged to build the solution and also the implementation details of T-ODAP. The evaluation methodology for assessing T-ODAP is presented in Chapter 5, where the theoretical evaluation is discussed, along with details of the tests of the implementation. Lastly, in Chapter 6 we present present our conclusions for this work, along with the contributions made, the system's limitations and future work.

2

Background and Related Work

Contents

This chapter presents background on blockchain, interoperability and game theory applied to blockchain, as well as the corresponding state-of-the-art. First, the concept of blockchain is explained, along with its basic characteristics. Then, the different types of blockchain - permissioned and permissionless - is presented. Hyperledger Fabric is also introduced as an example of a permissioned blockchain. Subsequently, we present background on blockchain interoperability and the current available interoperability solutions, along with game theory basics, games, and game theory applied to blockchain. Finally, the state-of-the-art is reviewed, ending with a comparison between existing solutions.

2.1 An Introduction to Blockchain

A blockchain can be described as a decentralized, tamper-proof distributed ledger [1], that allows trusted transactions among untrusted participants [17].

2.1.1 Transaction

A *transaction* is proposed by a user (blockchain participant) and it is an essential component of the blockchain. The data it contains depends on the blockchain's scope - e.g. if it is financial, among other data, the transaction contains the value in concern and the addresses of the sender and receiver. More broadly, the transaction can also be called a *record*.

2.1.2 Block

A blockchain stores different sets of transactions into *blocks*, where each block is connected to others forming a chain (hence its name). Besides the set of transactions (the *payload*), a block contains a header. The latter usually includes, among other parameters:

i A hash of itself (block digest), used to protect the integrity of the transactions in the block;

ii A hash of the previous block, adopted to foster immutability of the chain - if a participant attempts to tamper a block, this will be detected since its hash will change;

iii A nonce, that is used to check if the block is valid.

iv A timestamp, in order to authenticate the block's time of creation.

Figure 2.1 illustrates the headers of the first four blocks of a chain. The first block (block 0) is called Genesis block. It has no previous hash since there is no previous block to refer to.

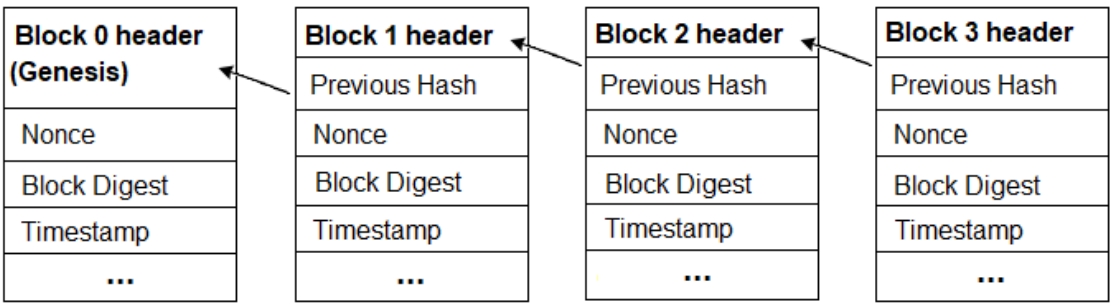

Figure 2.1: Blockchain data structure example (adapted from [1])

2.1.3 Nodes

The blockchain nodes form and maintain the network's infrastructure. Each node communicates with other nodes and contains a local replica of the chain - when a block is verified, each node attaches it to its local replica. This replica is usually the same on all nodes, although on some blockchains there may be temporary or even permanent distinct local replicas. This can either be a consequence of the nature of the blockchain (for example, due to a probabilistic consensus algorithm) or derived from a need to protect privacy - in a private blockchain, it is desirable for participants to be able to hide certain parts of the state they hold [18]. Though this may be the case for local replicas, the *global state* (the set of states that compose the blockchain) remains consistent.

2.1.4 Consensus

The network nodes reach *consensus* when they agree on a global state for the blockchain. There are several algorithms used to reach consensus, some more complex than others, depending on the type of blockchain (further explained in the following subsections).

2.1.5 Permissioned Blockchains

Permissioned (or private) blockchains have been popular around enterprises. In this type of blockchains there are restrictions in terms of the users that can participate in the blockchain and what these users can do once they enter it - a pre-selected set of nodes controls the consensus process, thus, the global state of the blockchain. Note that entities external to the network are unable to access any information. As a result, all peers are identifiable and well-known - in case of an attack, this can be useful for (at least) identification of the attacker. This type of blockchain utilizes Byzantine Fault Tolerant protocols (BFT) as consensus mechanisms. Every permissioned blockchain belongs to an organization (or consortium of organizations), where each node represents a given stakeholder within that organization (or organizations). By using a blockchain of this type, an enterprise is able to acquire the benefits of a

blockchain without needing to reveal private information (as it would in a public blockchain such as Bitcoin). Some examples of permissioned blockchains include Hyperledger Fabric (that will be explained in detail later) [2], Quorum [19] and R3's Corda [20].

There are two different architecture types when it comes to these blockchains:

i *Order-Execute Architecture* - This type of architecture defines that the blockchain transactions must be first ordered (using a given consensus protocol), and only then executed. This execution must happen in all nodes that participate in the consensus process, sequentially [2]. Order-Execute architecture was adapted by most of the existing blockchains systems, both permissioned (such as Tendermint [21] and Quorum [19]) and permissionless (such as Ethereum [22]).

ii *Execute-Order-Validate* - In this architecture, that arose with Hyperledger Fabric, the transactions are first locally executed, then ordered (through an ordering service), and finally validated by all peers, after which they are sequentially applied to each peer's local ledger. This architecture will be further mentioned when we explain Hyperledger Fabric in depth, in Section 2.2.

It is important to note the drawbacks of permissioned blockchains. These include centralization and lack of transparency, which some believe goes against the purpose of the blockchain technology itself [10]. Security and trust issues are also a considerable concern in this type of blockchain, since if the authorized nodes are not trustworthy, the network is compromised (given that these control its global state).

2.1.6 Permissionless Blockchains

Unlike permissioned blockchains, this type of blockchain is transparent and open to everyone that wants to join. Anyone can read its contents, add transactions to it, and take part in its consensus process, participating in deciding what the current state is and what blocks are added to the chain. Permissionless (or public) blockchains are considered fully-decentralized since there is no single entity (or consortium) controlling the network. This brings the valuable advantage of not requiring trust in individual nodes, but instead, trust in the system itself, which leads to greater security. Additionally, the more participants work in the blockchain, the more secure it becomes.

Within permissionless blockchains, classical consensus protocols such as BFT cannot be applied. The reason for this is that these protocols require a set of well-known nodes, who can authenticate each other as members of the group. Since in this type of blockchain anyone can join the network, a *Sybil Attack* [23] could be executed. In this context, the attacker is able to overwhelm the consensus protocol's fault tolerance threshold by creating enough virtual participants.

Thus, consensus protocols such as Proof-of-Work (detailed later) are required, where nodes are monetarily incentivized (with cryptocurrency) to maintain the network.

Within a decentralized system, security is more costly (consensus algorithms tend to be much computationally heavier than BFT's). Scalability is also usually worse given the fact that, in this context, transactions require validation from multiple entities (e.g. Bitcoin is known for being robustly decentralized, however it is slow). However, since the trust is distributed (instead of trust in individual entities), the system tends to be more secure.

On another hand, permissioned blockchains tend to be more centralized and scalable, but potentially more vulnerable to attacks given the centralization.

The blockchain trilemma addresses the challenges that blockchain developers face when trying to create a system that is decentralized, scalable and secure, without having to compromise any aspect. At this moment, no such blockchain exists [24] - each blockchain is often forced to make trade-offs and choose at maximum two of the three core aspects, having to compromise on the third - but it might in the future.

In permissionless blockchains, the consensus processes are complex and usually still need a much longer time to complete when compared with traditional, centralized solutions (e.g. in Bitcoin, the maximum estimated throughput is approximately between 3.3 and 7 transactions per second [25] while Visa is capable of achieving around 24,000 transactions per second [26]), negatively impacting its *scalability*.

Bitcoin is an example of a permissionless blockchain. It utilizes a consensus algorithm named Proof-of-Work (PoW), that consists in solving a hard and computationally expensive cryptographic puzzle (more specifically, finding a hash value which satisfies a given condition) [1]. Several nodes (called *miners*) attempt to solve the puzzle at the same time and the first one to solve it builds a block to add to the network (this is called mining a block). In order for this block to be added the chain, the transactions it contains must first be verified and validated by other nodes. If the block's transactions are valid, this miner is rewarded with cryptocurrency. The nodes that validated the block express their acceptance by working on creating the next block of the chain using the hash of the accepted block as the previous hash, thus creating a coherent global state.

In this context, if a malicious node attempts to tamper a transaction in one block, it has to calculate the PoW for that block and for all the following blocks, which requires an enormous amount of computational resources, due to the fact that each block contains the hash of the previous block. Besides this, in order for the global state to be accepted, it will have to do this faster than an honest miner would mine a block, which is extremely difficult. Plus, given the assumption that the probability of an honest miner finding the next block is higher than the probability of the attacker finding the next block, the attacker's chance of success decreases exponentially as the number of blocks it has to catch up with increases [8]. However,

despite the defense mechanisms, this attack is still possible both in Bitcoin and in other permissionless blockchains. The 51% attack happens when an attacker possesses more than 50% of the network's computational power, being able to perform any action it wants. The latter is usually associated with *mining pools* (a group of miners who combine their computational power over a network, with a common goal). This attack will be further explained in Section 2.4. Additionally, authors in [27] discovered that an attacker with just 38.2% of the network's computational capacity (a number less challenging to achieve through a mining pool) was also able to successfully perform this attack [1].

It is also important to mention forks, which are multiple, competing versions of the chain. This is dangerous, since it creates doubt around who owns which coins [28].

In Bitcoin, to guarantee a consistent global state despite the possible forks, Nakamoto proposed the *longest chain rule*, that states that each node must accept the longest chain as the valid version of the chain, because it is the one which has the greatest Proof-of-Work effort invested in it [8]. Since the blockchain's security and robustness is based on its decentralized consensus, that agrees on the global state of the chain, mining on a fork chain (an alternative chain) is an attack to the consistency and security of the whole system. Miners may want to fork instead of mining honestly due to the possibility of obtaining higher rewards. However, these rewards are only relevant if the block is included in the long-term agreed state (the consensus chain).

In Ethereum (another permissionless blockchain), much like in Bitcoin, all peers first attempt to form a block that contains valid transactions. After this, a cryptographic puzzle (PoW) must be solved. The peers that can solve the puzzle disseminate the block to the other peers via a gossip protocol. Finally, when a peer receives a block, it validates its PoW solution, and all the transactions that the block contains.

2.1.7 Permissioned vs. Permissionless

In permissioned blockchains, the BFT consensus protocols execute a lot faster than permissionless blockchain's consensus protocols (such as PoW). These protocols are more efficient and performant since they allow for a higher transaction throughput, given the fact that there is no overhead derived from performing expensive computation as a pre-requirement to create a block (unlike in permissionless blockchains). Permissioned blockchains (such as Hyperledger Fabric) also allow some nodes to keep sensitive information private while still maintaining a consistent global state - this may be particularly useful to enterprises that intend to control what each stakeholder (modeled as a node) can see. Nonetheless, permissioned blockchains have the disadvantages of centralization, security (since trusted nodes may collude) and trust (the credibility of a permissioned blockchain depends on the credibility of its authorized nodes). Additionally, as mentioned before, this type of blockchains is opaque to external

observers. While this is advantageous in terms of privacy of information, it makes interoperability a challenging task due to the difficulty of observing a permissioned blockchain's internal state. Some solutions have emerged to tackle this problem, which we further explain in Section 2.8.

On another hand, permissionless blockchains are open and provide full decentralization (since no central authority or authorities exist). They also provide transparency, since all participating nodes can see all transactions that ever occurred. Both of the mentioned qualities are extremely valuable since they remove the need for trust in individual nodes: all of the data published in the blockchain is open to the public for verification. This way, no intermediaries are necessary, causing the blockchains to be more secure. However, in these blockchains the consensus process is slower, due to complex algorithms (such as PoW) being necessary to maintain the security of the system. This also causes the transaction throughput to be much slower than in permissioned blockchains, harming its performance. Finally, energy consumption is also a concern in permissionless blockchains, since its consensus algorithms spend high amounts of energy.

Both of the blockchain types can provide great advantages. However, depending on the specific use case and the industry it is inserted in, one option might be better than the other.

2.1.8 Blockchain adoption challenges

Although ”blockchain” is, without a doubt, one of the biggest buzzwords of the last decade, there are still some significant obstacles preventing mass adoption, particularly in the adoption of enterprise blockchains.

The most relevant ones are:

- *Expensive implementation* - The price of a commercial blockchain-based solution can vary a lot depending on several factors such as the type of blockchain, complexity of the solution, or the amount of users interacting with it. However, on average, this type of implementation is still very expensive (a full-scale solution may cost more than 100,000 dollars) [29].

- *Data and value silos* - Different use cases and stakeholders require various blockchain features. This has been leading to an heterogeneous and fragmented ecosystem, where blockchains are focused on resolving specific challenges, causing scaling and synergies between different solutions to be extremely difficult, if not impossible. Additionally, as a result, many blockchains are still immature [3].

- *Hard to learn* - This technology might be daunting at first, especially for someone which does not have an information technology or mathematical background. It is not considered mature yet, there are many concepts to learn and, due to the aforementioned data and value silos, often times each

blockchain contains a specific set of new concepts. Besides this, as so many people around the world are writing and speaking about the subject (experts or not), a portion of the information is either wrong or incomplete.

- *Blockchains also become obsolete* - Technology becomes obsolete when it is replaced by a newer or better technology, sometimes extremely fast. Blockchains are no exception. Investing in a blockchain technology comprises a significant risk, since it might become obsolete (or vulnerable, or shutdown). Although some blockchain migration solutions have been appearing recently (e.g. [30]), the current solutions are only applicable to a small set of public blockchains [3].

These obstacles are very strong reasons why blockchain interoperability is critical for this technology to achieve its full potential and thrive.

It is important to note that permissionless blockchains also have their challenges. Despite more people investing in cryptocurrencies such as Bitcoin and Ethereum and the huge increases in prices, there have been many conversations around a deeper problem - the colossal amounts of energy spent by public blockchains with consensus algorithms such as Bitcoin's Proof-of-Work. Throughout 2017, the latter's energy consumption was higher than the energy consumption of 129 countries [31].

2.2 Hyperledger Fabric

As mentioned before, Hyperledger Fabric is an example of a permissioned blockchain that first introduced the Execute-Order-Validate architecture. A distributed application for Fabric consists in:

i *Smart Contract (chaincode)* - The code that implements the application logic. *System chaincodes* are special smart contracts that manage and maintain the blockchain system.

ii Endorsement Policy - Applied to a given chaincode, this policy defines the subset of peers that are in charge of *endorsing* the chaincode's transactions, i.e. executing each transaction. It is created and modified by pre-selected administrators.

In terms of entities, this blockchain considers the following:

A *Client* - Is in charge of submitting *transaction proposals* (i.e. transactions for execution), helping or guiding the execution phase and broadcasting the endorsed transactions for the ordering phase.

B *Peer* - Executes transaction proposals and validates ordered transactions. Each peer works to maintain the ledger.

C *Ordering Service Node (orderer)* - The *Ordering Service* establishes the total order of all transactions and, optionally, *access control* checks (to see if a given client has certain permissions in the system). The orderers are unaware of the system state and are unable to execute or validate transactions, making consensus process as modular as possible.

There is one more essential architecture component to Fabric named Membership Service Provider (MSP). The MSP service is responsible for the mechanisms of authentication and authorization of all entities in the system. This is extremely important due to the Permissioned nature of this blockchain. It also provides authentication to operations such as transactions or endorsements.

There are three phases in Hyperledger Fabric's architecture: Execute, Order and Validate. This scheme introduces a *hybrid replication* design - *passive* when it comes to the execution phase, since parallelism is allowed and since it is not executed by all peers, and *active* in regards to the validation process, since it is deterministic and executed individually by each peer.

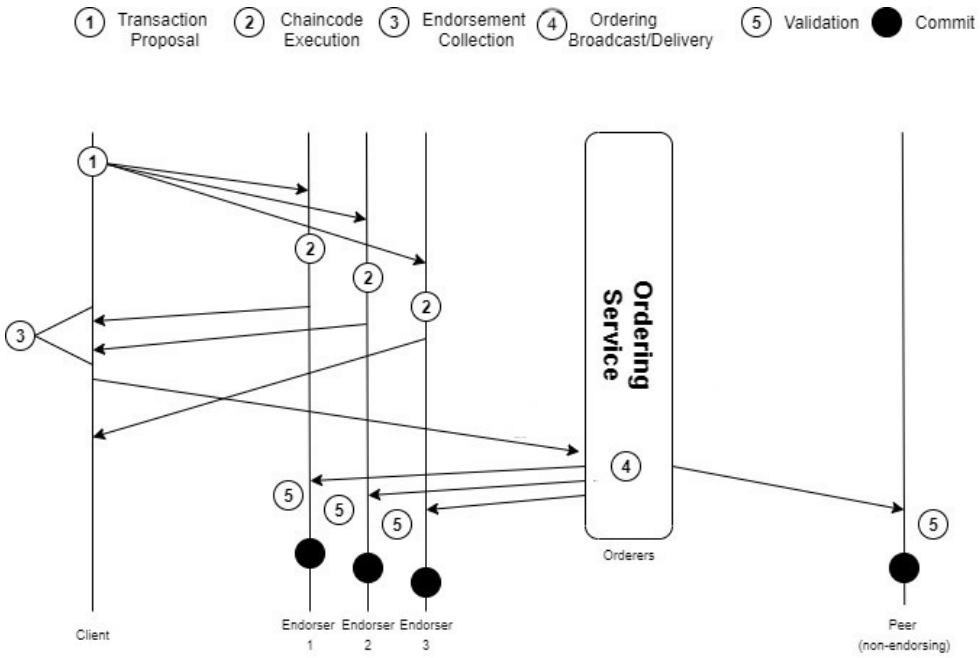

Figure 2.2: High-level transaction flow (adapted from [2])

Fabric's transaction flow, depicted in Figure 2.2, can be described as follows:

2.2.1 Execution Phase

The flow is initiated with the Execution Phase. First, clients sign and send a transaction proposal to one or more endorsers (step 1), that are specified by the endorsement policy of the chaincode the transaction belongs to. Then, each endorser *simulates* the proposal, i.e. it is only executed in the endorser's local

blockchain and not propagated to other peers (step 2). The simulation results do not persist in the ledger state. The output corresponds to a *writeset* (with state updates produced by the simulation) and a *readset* (the version dependencies of the simulation).

Next, each endorser cryptographically signs a message that contains the writeset and readset, along with metadata - an *endorsement*. This *endorsement* is then sent to the client for it to continue the flow.

In order for a transaction to be created by a client, the latter needs to collect *endorsements* that satisfy the endorsement policy associated with the transaction's chaincode. This can be seen in step 3. Here, all the different execution results received from endorsers must be equal between themselves.

Finally, if the client receives valid *endorsements*, that are according to the endorsement policy, it creates the transaction and sends it to the Ordering Service.

The fact that this phase does not imply persistent changes in the ledger brings an advantage to Fabric - if a transaction in non-deterministic, the *writesets* and *readsets* produced by different endorsers in the Execution phase will differ, causing the endorsement policy to not be satisfied, thus causing the transaction to not be created by the client. This way, the transaction will not cause inconsistencies in the ledger state, as it happens with Order-Execute architecture [2].

2.2.2 Ordering Phase

In the Ordering Phase (step 4), the Ordering Service (constituted by the orderers) establishes a total order among the received transactions. The output of this phase is a hash-chained sequence of blocks containing transactions [2] which are then sent to the network peers. When it comes to consensus, Fabric presents the possibility to choose between a CFT (Crash Fault Tolerant) or BFT implementation.

Fabric can be configured to use a built-in dissemination service to disseminate the blocks to all the peers. This is because the system can have many peers, but few orderers. This service can also perform *state transfer* to peers that newly joined, and peers that were disconnected for a long time.

2.2.3 Validation Phase

In the last of the three phases, depicted in step 5, each peer executes the following steps sequentially:

i The endorsement policy is evaluated for all transactions in the block, i.e. each endorsement is verified according to the chaincode's policy. If one endorsement does not satisfy the policy, the transaction is considered invalid and its effects will not persist in the ledger;

ii A *read-write* conflict check is done for all transactions in the block. More specifically, for each transaction's *readset*, it compares the version of the keys with the corresponding keys in the local

ledger. If they do not match, the transaction is considered invalid and it will not become permanent in the ledger;

iii Finally, the local ledger of the peer is updated (the block is appended to the chain) causing the update of the blockchain state (as can be seen in Figure 2.2, in the ”Commit” step).

It is important to note that invalid transactions are also included in the appended block, due to certain use cases in which it can be useful to track them (e.g. audits, or tracking down clients that initiated a DoS attack).

2.3 Blockchain Interoperability

There is no single way of defining ”Interoperability”. One simple yet sound definition that applies is the following: “Interoperability is the ability of two or more software components to cooperate despite differences in language, interface, and execution platform” [32].

The National Interoperability Framework Observatory (NIFO) defines six layers of interoperability [33]: technical interoperability, semantic interoperability, organizational interoperability, legal interoperability, integrated public service governance, and interoperability governance. In this work, we focus on technical and semantic interoperability - the most relevant layers for the blockchain technology.

Blockchain interoperability is a relatively new theme - interest from academia and industry did not start growing until about three years ago [3]. This type of interoperability emerged due to the desire to create new synergies between blockchains, thus creating new use cases. Besides this purpose, it is key to blockchain migration - the transfer of states from one blockchain to another. Migration is extremely important because failures can (and most certainly will) happen at some point: blockchains can be attacked, become obsolete, or shutdown [3]. Without migration, the risks associated to blockchains are high, severely affecting scalability and holding back mass adoption.

Due to the core differences between permissioned and permissionless blockchains, the interoperability problem is distinct for each of the types. Even within the same type, it can take many different forms due to the huge variety of existing blockchain infrastructures - there are no standards for blockchain interoperability yet, however it is ongoing work [3].

2.3.1 Cross-Blockchain Communication

Before diving into different mechanisms within this context, it is essential to understand cross-blockchain communication, given that it is a requirement for blockchain interoperability.

Cross-blockchain communication involves two blockchains: a *source* blockchain (where a transaction is initiated), and a *target* blockchain (where a transaction is executed) [3]. It comprises several base

concepts and components [3] [34]:

- *Homogeneous Blockchains* - Two blockchains are homogeneous if they both belong to the same blockchain system (e.g. EVM-based blockchains [22]).

- *Heterogeneous Blockchains* - Two blockchains are heterogeneous if they are not homogeneous (e.g. Bitcoin and Hyperledger Fabric).

- *Cross-Chain Transaction (CC-Tx)* - A transaction between different, homogeneous chains.

- *Cross-Blockchain Transaction (CB-Tx)* - A transaction between different, heterogeneous chains.

- *Cross-Blockchain Distributed Application (CC-dApp)* - A decentralized application that leverages cross-blockchain transactions to implement its business logic.

- *Internet of Blockchains (IoB)* - Is a system “where homogeneous and heterogeneous decentralized networks communicate to facilitate cross-chain transactions of value.”.

- *Blockchain of Blockchains (BoB)* - Is a structure that organizes blocks with transactions that belong to a CC-dApp, spread across multiple blockchains.

- *CCCP (Cross-Chain Communication Protocol)* - Is a protocol that ”allows homogeneous blockchains to communicate” (e.g. the XCLAIM framework [14]).

- *CBCP (Cross-Blockchain Communication Protocol)* - Is a protocol that allows heterogeneous blockchains to communicate (e.g. the Interledger Protocol [35]).

In this document, we will refer to *CX-Tx* as a transaction that can either be cross-chain or cross-blockchain.

Several authors presented interesting results regarding *CX-Tx*. Zamyatin et al. concluded that secure cross-blockchain communication (CCC) is impossible without a trusted third party, by reducing CCC to the Fair Exchange problem [36].

On another study, Borkowski et. al presents the ”lemma of rooted blockchains” that states that currently a source blockchain is not able to efficiently verify the existence of data on a target blockchain [37]. This would require the source blockchain to mimic the target’s consensus mechanism and to store a large set of the other’s blockchain block history [3].

More recently, on a study applied mainly to public blockchains, Lafourcade and Lombard-Platet [38] stated that true blockchain interoperability corresponds to two blockchains being able to work together,

being interoperable with each other. They state that a blockchain A is interoperable with a blockchain B if an interoperable transaction does not violate A's rules, and B's ledger is not empty. In their study, they conclude that interoperability is a contradiction of blockchain's nature itself since the latter is constructed to be self-sufficient and to not rely on external data. However, under a weaker definition (A is interoperable with B if a user can transfer assets from A to B and the transactions that represent those transfers are only valid on B in case the sender has enough funds on A) and considering a blockchain with two ledgers, it is indeed possible to achieve it [3].

The aforementioned studies point to an important conclusion: Cross-blockchain communication is unfeasible (or at least extremely challenging) without a trusted third party. Trusted third parties can be either centralized (trusted validators), semi-decentralized (a consortium of trusted validators) or decentralized (e.g. another blockchain) - being the latter the focus of this report.

Since the heart of the blockchain technology is to maintain a secure and immutable ledger in a way that is as decentralized as possible, the true gain of this mechanism arises when the trusted third party is decentralized (which is the focus of this thesis) or at least when accountability mechanisms are in place, in order to motivate the party to behave properly and follow the protocol.

2.3.2 Existing Interoperability Mechanisms

Existing mechanisms to achieve blockchain interoperability can be divided in different categories, which we detail below, based in [3].

2.3.2.A Public Connectors

This category only regards public blockchains (which usually implement cryptocurrencies) and includes several mechanisms - Sidechains, Notary Schemes, Hashed Time-Locks (HTLCs) and Hybrid solutions.

A *sidechain* can be defined as "a mechanism for two existing blockchains to interoperate, scale (e.g., via blockchain sharding), and be upgraded" [3]. In this context, in each pair of blockchains there is one that acts as the main chain (that maintains a ledger of assets), while the other is considered to be an extension of the main one - the sidechain. The mainchain and the sidechain communicate via a Cross-Communication Protocol.

The most common use to this mechanism corresponds to the transfer of assets between the mainchain and the sidechain. This interaction is called *two-way peg*.

It is important to note that two blockchains can be sidechains of each other, meaning that a sidechain is not less important than a mainchain.

In this context, **Zendoo** is a sidechain creation platform for Bitcoin-based systems that allows for the creation and communication of sidechains without knowing their internal structure [39]. The system maintains its security through the use of zk-SNARKSs proofs [40], a type of zero-knowledge proofs.

Notary Schemes are another type of Public Connector. They can be defined as a third-party that monitors and performs operations on several blockchains. The mechanism is conceptually simple, however this solution is commonly centralized, which is not ideal given the focus of the blockchain technology. It is possible to diminish the centralization by having more than one notary.

This category also comprises *HTLCs*, a class of payments that uses *hashlocks* [41] and *timelocks* [42] to ensure atomicity of operations [3]. It is usually used between two parties, where one commits to make a given transaction, by providing a cryptographic proof to the other before a certain timeout. This way, it is possible to establish trust between them and produce multiple outputs, such as conditional payments, or *CX-Tx* payments. *Atomic swaps* are a type of *CX-Tx* payments that enable the exchange of one cryptocurrency for another, in a decentralized way.

Finally, the category comprises some hybrid solutions that combine the previously mentioned and other novel approaches. One of these are *Distributed Private key* approaches. In this solution, a private key is not held by a single entity, but by many - each participating entity holds a ”part” of the key [43], distributing the control and the assets among them. *Distributed private key* schemes can be used to implement decentralized notaries and decentralized two-way pegs [3]. Another Hybrid solution is the use of sidechains with *escrow parties* that rely on smart contracts, where an *escrow* is an agreement in which a third party controls a transaction or group of transactions between two entities, while typically holding the assets of one of them [3].

2.3.2.B Blockchain of Blockchains

A blockchain of blockchains can be described as a framework that provides ”reusable data, network, consensus, incentive, and contract layers for the creation of customized blockchains, to power decentralized applications, that interoperate between each other” [3]. The blockchain of blockchains that exist nowadays allow developers to create their blockchains while still having interoperability with others (both created with the framework, and other existing blockchains) [3].

Polkadot [15] is an example of a blockchain of blockchains. It contains a *relay chain*, the central chain of Polkadot, that has the function to coordinate the system as a whole.

Then, the system comprises parachains and parathreads. A parachain is an application-specific data structure that often takes (but does not have to) the form of a blockchain. It may be specialized in features like smart contracts, privacy or in verifying proofs. Since it has a parallel nature, several parachains are able to parallelize transaction processing and achieve a very high scalability. A parathread can be thought of as a parachain thread - i.e. a temporary parachain that is able to do everything a parachain can, however with lower costs and resources associated with it.

There are three roles in Polkadot [44] - the *Nominator*, that helps particular validators to get into the

active validator set and thus produce blocks for the chain, the *Validator* that can produce blocks on the Relay Chain if it is on the active validator set and lastly the *Collator*, that collects parachain transactions and produces proofs for the validators on the relay chain. The latter can also send and receive messages from other parachains.

Polkadot's parachains and parathreads are interoperable with each other. The blockchain of blockchain also contains bridges that allow Polkadot to connect to existing blockchains, such as Bitcoin.

Polkadot is built using Substrate [45], a modular blockchain framework that allows developers to easily construct costumizable blockchains.

2.3.2.C Hybrid Solutions

These solutions are interoperability solutions that do not fall into any of the previous mechanisms. Hybrid solutions can be divided into Trusted Relays, Blockchain Agnostic Protocols and Blockchain Migrators.

Trusted relays are trusted parties that redirect transactions from a source blockchain to a target blockchain [3]. These schemes are usually related with a permissioned blockchain environment, where trusted third parties forward *CX-Tx* transactions.

One solution in this context is **Hyperledger Cactus** [16]. To achieve interoperability between several blockchains, Cactus uses trusted nodes (*Cactus nodes*). It is important to note that each of these nodes is connected to solely one organization, meaning that the *CX-Tx* transaction requests of one node are always from the same organization.

Each *Cactus node* fills two roles - connector and validator. The connector is an active component that makes changes in a blockchain, including invoking and deploying smart contracts. The transmission of *CX-Tx* transactions to a destination blockchain is executed by this component. The validator is a passive component that includes the attestator role - a role that verifies arbitrary ledger facts. This includes verifying if a transaction was correctly sent and received by a destination blockchain.

Please note that in Cactus it is also possible to implement a decentralized validation mechanism via a consortium of validators. This can be implemented through the development of plugins, leading to a semi-trusted, more decentralized model.

Blockchain Agnostic Protocols are another type of Hybrid Solution that refer to a single platform that allows cross-blockchain or cross-chain communication between several different blockchain systems. This mechanism does not guarantee backwards compatibility, i.e, compatibility with current blockchains (they would have to change their source code).

An example of a blockchain agnostic protocol is **DeXTT** [46] - a solution that provides a cross-blockchain transfer protocol used to transfer tokens that may exist on several blockchains simultaneously, in a decentralized way.

Blockchain agnostic protocols also comprise BoBs. As mentioned in Section 2.3.1, a BoB is a struc-

ture that organizes blocks with transactions belonging to a CC-dApp, spread across several blockchains. BoB joins several blocks from different blockchains into larger blocks named ”meta-blocks”, organized through a consensus mechanism [3]. **Overledger** is a BoB that builds a unique group of transactions, that are then put together and ordered by CC-dApps [47]. It has the goal to stop digital ledger technology from existing in silos and to achieve interoperability by being built on top of different blockchain systems.

Blockchain Migrators are also a Hybrid Solution, that allows for an end user to perform a migration of the state of a blockchain to another [3]. One example of a blockchain migrator is presented by **Fynn et al.**, that present a mechanism where it is possible for a smart contract to be moved to and executed by another blockchain, moving the state required by the transaction (*move operation*) [48]. With this mechanism, it is possible to transfer arbitrary states between blockchains. So far, it has been tested [3] on Ethereum and Hyperledger Burrow [49].

Below, we have a table that sums up the aforementioned existing blockchain interoperability solutions, their sub-categories and each of those sub-categories' main use case.

Category	Sub-category	Main use case
Public Connectors	Sidechains	Scalability, asset exchange
	Notary Schemes	Cryptocurrency exchanges
	Hashed timelocks	Cryptocurrency trading
	Hybrid	Enabling cross-chain assets
Blockchain of Blockchains	–	Creation of customized blockchains
Hybrid Solutions	Trusted Relays	Efficient interoperation
	Blockchain-Agnostic	General protocols
	Blockchain of Blockchains	Cross-blockchain dApps
	Blockchain Migrators	Risk reduction

Figure 2.3: Blockchain Interoperability Solutions (adapted from [3])

2.4 Game Theory

Game theory has the goal of modeling a strategic interaction between different players in a context with predefined outcomes [50] - a *game*. It can be applied to the most significant challenges in the Blockchain technology, such as security problems, mining management, economical issues with the technology and energy trading [1].

We now present some fundamental concepts (based on [1]) for the better understanding of a *game*:

- *Player* - A player (or agent) is a decision-maker in the game;
- *Utility* - A *utility*, i.e. a revenue or payoff, reflects the outcome expected by the player, acting as way to quantify its preferences - if a given output is preferred, the utility associated with it will be higher than if that output is not desirable;
- *Strategy* - A set of actions that the player can choose from, where the ultimate goal is to achieve its expected outcome. This strategy is based not only in the player's own strategy, but also in the other players' strategies;´
- *Rational* - A rational player has clear preferences, is self-interested and always desires to maximize its own utility by choosing the strategy that comprises the optimal expected outcome.
- *Static Game* - A game in which all player make a decision simultaneously, thus not being aware of other players' actions in advance.
- *Dynamic Game* - A game in which players take actions sequentially - the later players have information about the first players' choices.

2.4.1 Preliminaries on Game Theory

In this subsection we introduce formal notation in the context of game theory, that will be used in the descriptions of games and attacks [1].

Considering N players:

- P_i - The set of strategies of the player *i*;
- *P* - Is equal to $P_1 \times$ ... $\times P_N$, and corresponds to the cartesian product of the sets of all individual strategies;
- $p_j \in P_i$ - Corresponds to the strategy *j* of player *i*;
- *p* = (p_1, ..., p_N) - Is a vector of the strategies of N players, after each has been chosen from the corresponding set;
- π = (π_1(p), ..., π_N(p)) - Is a vector of the corresponding payoffs, where π_i(p) is the *utility* of player *i*.

- $p^* = (p_1^*, ..., p_N^*) \in P$ - Corresponds to a *Nash Equilibrium* if no player can gain higher utility by changing its own strategy, when the strategies of the other players remain unchanged. This can be translated by the following inequality:

$$\forall_i, p_i \in P_i : \pi_i(p_i^*, \bar{p_i^*}) \geq \pi_i(p_i, \bar{p_i^*}), \tag{2.1}$$

  Where $\bar{p_i^*}$ = (p_1, ..., $p_i - 1$, $p_i + 1$, ..., p_N) is a vector that contains strategies of all players except player *i*. Inequality 2.1 demonstrates the state of *equilibrium* of the game.

- BR_i is the best response mapping of player *i* - i.e. the strategy (or set of strategies) that produce the most favorable outcome for a player, knowing the other players' strategies.

- $\mathcal{A}$ - An attacker;

- M_i - A rational miner (i.e. a miner that desires to maximize its own utility);

- $B_{\mathcal{M}_i}$ - A block that was found and mined by a miner M_i.

- B^* - The latest block in the chain;

- $B_{\mathcal{A}} = (\mathcal{H}_{\mathcal{A}}, \mathcal{D}_{\mathcal{A}})$ - A block found by the attacker $\mathcal{A}$, that corresponds to a header ($\mathcal{H}_{\mathcal{A}}$) and the data itself ($\mathcal{D}_{\mathcal{A}}$).

2.4.2 Existing Games

There are several types of *games* in this context, each with its strengths and drawbacks. Depending on each use case, it might me more appropriate to use one over the other. Below, we state some of the most relevant existing games.

2.4.2.A Non-cooperative Game

In this type of game, cooperation does not take place through forming coalitions or by reaching agreements. Cooperative behaviour can exist, but it is never the main goal of the players, since they are self-interested. Non-cooperative games can be used to analyze both the static and dynamic game [1].

In the non-cooperative game, the *Nash Equilibrium* (p^*) would correspond to a scenario in which no player would want to deviate from its current strategy, since it is the one which is associated with a higher utility value. It is important to note that in certain cases there is more than one Nash Equilibrium, or even none at all.

This game can be used, for example, to model the interaction between miners in a blockchain network. In this scenario, the miners (which are rational players) compete for a reward from mining successfully, and invest strategically in computational power.

2.4.2.B Extensive-form Game

This type of game describes the players' interactions with each other through the use of a game tree. This tree illustrates each of the possible decisions that each player can make, while containing information about the other players' moves and about the expected payoffs for all possible game outcomes [1].

An extensive-form game is composed of many smaller games - subgames - and each of these games can be perceived as a static non-cooperative game. In this type of game, the *Nash Equilibrium* is attained in a given subgame if it is attained in all the other subgames as well. To achieve this, a technique called *backwards induction* is often applied. The technique consists in first considering the end of the game - i.e, the decision that will be made in the last move - going back on each step until the first move of the game. In this way, it is possible to achieve a *Nash Equilibrium* if there is one.

In the context of blockchain, the extensive-form game can be applied to the fork chain selection scenario. In the start of each mining round, miners choose the chain in which they will mine on, depending on the actions taken by other players in previous rounds. Eventually, the blockchain will fork which will lead to a branching tree structure, in which the game can be applied [1].

Figure 2.4 illustrates this scenario. Here, we consider two players, miner 1 and miner 2, and two chains, C1 and C2. Miner 1 chooses its strategy first, between C1 and C2. Then, depending on the previous action of miner 1, miner 2 chooses between C1 and C2. The branching tree contains four different pay-offs represented by the four terminal nodes (C1,C1), (C1,C2), (C2,C1) e (C2,C2).

2.4.2.C Stackelberg Game

The Stackelberg Game includes two entities: *leaders* and *followers*. Both of these entities are rational, thus self-interested. The *leaders* have the advantage of being able to choose first, causing the *followers* to choose their strategies based on the strategies chosen by the first.

The goal of this game is to establish a Stackelberg Equilibrium.

In order to find this equilibrium, *backwards induction* is usually applied. This game guarantees that the *leader* will be able to attain *at least* the payoffs of the corresponding equilibrium.

Within a blockchain network context, the Stackelberg Game's entities can be a Service Provider as a leader and miner as a follower. The Service Provider can provide the miner with computational power, that it can then use to mine blocks. The more computational power a miner has, the better is its chance at receiving the mining rewards. In this example, the Service Provider would first set its price, and then the miners would decide their demand [1].

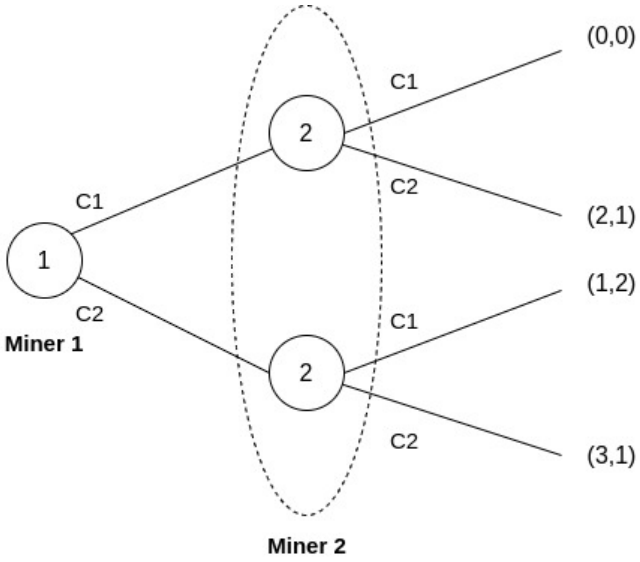

Figure 2.4: Fork chain scenario with two miners and two chains C1 and C2

2.4.2.D Stochastic Game

This game is played by a set of players. It can be seen as a set of static non-cooperative games, that are repeated over time. Each stage of the game - i.e, each static non-cooperative game - corresponds to a *state*. The transition to a new *state* is controlled by *stochastic transition probabilities*, that in turn are controlled jointly by the players.

In the Stochastic Game, each player's strategies may change, influenced by the other players' strategies and transition behaviours. In each *state*, the players receive a given payoff that depends on the chosen actions.

This game is repeated until it reaches a common solution called Markov Perfect Solution - a set of strategies that achieve the Nash Equilibrium in every state of the game.

The stochastic game can be applied to chain selection in blockchain, regarding the transitions of blockchain structure. The game starts at an initial state, and at stage t each miner observes the blockchain structure and decides its strategy (i.e. selects a chain to mine on). According to their strategy and the current state, each miner receives an immediate payoff. Then, the game progresses to a new state. This process is repeated until the game reaches a Markov Perfect Solution (which in this case corresponds to following the Nakamoto protocol, i.e. mining on the longest chain).

2.4.3 Game Theory Applied to Blockchain

There are several game theory techniques and games that can be applied to improve certain aspects of the Blockchain technology. We will now focus on the **security** aspect, presenting common attacks in the technology, along with games that can mitigate them.

First, we present some important definitions [4]:

- *Altruistic Player* - A player who follows the prescribed protocol;
- *Rational Player* - As before, a player who acts in order to maximize its own utility;
- *Byzantine Altruistic Rational Tolerant (BART)* - Type of protocol that assures safety and liveness properties when byzantine behaviour exists;
- *Incentive-Compatible Byzantine Fault Tolerant (IC-BFT)* - Type of protocol that incentivizes rational agents to follow the protocol instead of rationally deviating the latter.

Game theory is valuable in the design of IC-BTF protocols given that is can guarantee that rational players have a better outcome by following the protocol's instructions than by deviating from it. This is possible when "following the protocol" corresponds to a Nash equilibrium p^*. However, in blockchain systems it is possible for rational players to band together and cooperatively deviate from a prescribed protocol - in this situation, the concept of Nash equilibrium is not applicable.

Selfish Mining Attack - This is a type of attack that occurs in Proof-of-Work based blockchain systems. In this context, attackers can be one of two entities: malicious miners (in the case of Block withholding *BWH*) or malicious mining pools (in the case of Pool Block Withholding *PBWH*). The *Selfish Mining Attack* consists of attackers not broadcasting newly found blocks, but instead, either:

1. Withholding the block, or
2. Releasing it after a certain amount of time.

This causes honest miners to waste computing power trying to find a block that has already been found. It also increases the likelihood that the attacker will find the next block, due to its advantage in terms of computational power and time - while honest miners are trying to find the block already found, the attacker can start working on the next one.

Several analysis of players' interactions exist when it comes to this type of attack. So far, research has shown that game models such as non-cooperative, stochastic, mean-payoff [51] and splitting game [52] can be applied to different scenarios of the *Selfish Mining Attack*. Below, we present some of these findings [1].

Having two selfish pools as players, the authors of [53] adopts a non-cooperative game. Each player's objective is to optimize their strategy - the *infiltration rate*, maximizing their utility. This rate corresponds to the fraction of computational power necessary to infiltrate an adversary. When an attack is successful, the attacker pool receives utility not only from the attacked pool's honest miners but also through the

infiltrated individual miners that perform the *BWH* attack. It has been proved that there is a unique Nash Equilibrium (i.e, none of the players wants to change their strategy). The equilibrium found proves that launching the *PBWH* attack is always the best response for each of the players. However, if both pools attack, each of the pool's final utility is less than if an attack had not occurred.

In a distinct scenario, a stochastic iterative game is applied. Here, the focus is not on improving the players' own profits but instead to promote high social welfare (i.e. profits to the entire pool), [54] applies *Zero Determinant (ZD)* strategies. In this context, there are two players - one honest miner, (focused on improving the social welfare) and a selfish miner (solely focused on maximizing its own profit). Each player can choose to either cooperate (mining honestly) or to launch the BWH attack to the other player. The paper states that as long as the honest miner chooses its strategies under certain conditions, the outcome of the opponent can be controlled - regardless of the strategies adopted by the selfish miner, to him/her it is more profitable to either cooperate mutually or attack mutually. This way (i.e. applying *ZD* stategies) the honest miner can motivate the selfish miner to cooperate, by choosing to cooperate himself. The downside of the proposed game is that it only focuses on the selfish miner's profit - the honest miner may not have any incentive to apply *ZD* strategies in the first place.

Majority Attack - Theoretically, it is possible for a given miner to own more than 50% of the network's computational power. In this scenario, the distributed consensus is broken and the blockchain's security is not be guaranteed: the attacker has a free-pass to perform malicious actions, such as reverting transactions or halting payments. The name of this attack is *51% attack*.

The *Majority Attack* corresponds to the *51% attack* where the attacker is a cartel of miners instead of a single miner. Various analysis of the players' behaviours in *Majority Attack* exist. Previous research has shown that Non-cooperative, Stochastic, Sequential and Stackelberg games can be applied to distinct scenarios of the attack [1].

Fork chain mining is a problem within the Majority Attack. At given points in time a blockchain may fork, meaning that it may be diverged into two potential paths forward. In the context of the *Majority Attack*, each miner has the choice to either mine on the honest chain, or to mine on a fork - either create or select a new branch.

The conditions under which a miner has an incentive to mine on a fork are studied in [28].

There are two possible scenarios:

(a) More than 50% of the network's computational power extends the longest chain;

(b) More than 50% of the network's computational power forks a chain.

In scenario a), if a miner chooses to deviate from the honest chain, it will end up wasting computational power, since the majority of the network's computational power is invested in the honest chain. In

this scenario, the miner will not achieve the PoW consensus and the mined block will not be included in the consensus chain. Thus, in this case, mining on the honest chain would be the best possible response to all the miners, meaning that there is one Nash Equilibrium where all miners extend the honest chain. The contrary applies to the second scenario - here, the best strategy for all miners is to mine on the majority's fork. Hence, there is a second Nash Equilibrium where all miners mine on that fork.

Standard Denial of Service Attack (DoS) - In the blockchain context, this attack consists of preventing honest miners from completing the mining process, thus not allowing them to win mining rewards and expected profit. In this attack, an attacker causes the corruption of the P2P network, which drains the resources of the attacked miners. In a DDoS (*Distributed Denial of Service*) attack, an attacker launches several DoS attacks at the same time.

Previous work has shown that non-cooperative, sequential and repeated game models are applicable to DDoS attacks. The analysis performed in [55], in the Bitcoin blockchain, is the result of one of those studies. In this context, in order to maximize the mining rewards, the attackers - mining pools - can choose to either:

(a) Trigger the DDoS attack, to lower the profits of the attacked mining pools;

(b) Invest in more computational power, in order to have a better possibility of solving the next PoW puzzle.

Consider a non-cooperative game that comprises two players - a big mining pool and a small mining pool - where the strategies of each are a function of the computational power distribution, the rate of the network's computational power over time and the probability of the attack being successful.

The authors of [55] have proved that when there is an inequality in the computational power distribution among these players, there is a Nash Equilibrium of both players investing in it. Both players launching the attack is also always a Nash Equilibrium, however it is weak, since players do not consider the other player's strategy choices.

Blockchain Denial-of-Service attack (BDoS) - Having the assumption that all miners are rational, unlike the aforementioned standard DoS attack that has a specific target, BDoS is an attack that targets the entire system's mechanism design - it exploits the reward mechanism with the goal of discouraging miner participation [56].

The core of this attack is the following (as explained by the authors of [56]):

There is a probability with which an attacker $\mathcal{A}$ can bring the system to a state where a block $B_{\mathcal{M}_i}$ (found by a miner M_i) is invalidated. This means M_i has a smaller probability of seeing any reward for its investment, although the costs of performing mining are the same as in honest mining.

This can be accomplished through the following steps [56]:

1. $\mathcal{A}$ mines on B^* (the latest block in the chain);

2. If $\mathcal{A}$ is able to successfully attach a new block $B_{\mathcal{A}}$ to B^*, instead of publishing it entirely, it publishes its header $\mathcal{H}_{\mathcal{A}}$ while concealing its payload $\mathcal{D}_{\mathcal{A}}$. At this point, $B_{\mathcal{A}}$ is not part of the main chain since it has not been published in full;

3. Until another miner creates a new block, $\mathcal{A}$ pauses its mining process. Then:

    - If all miners also stop, the system reaches a blocking state.
    - If a miner M_i - that is aware of $B_{\mathcal{A}}$ due to its published header - generates a new block, appending it to B^*, $\mathcal{A}$ precipitates to publish the complete $B_{\mathcal{A}}$ with the hope of being able to add it to the main chain.

If the attack is successful, M_i will not receive any reward for its investment and will have wasted computational power. This game can be analyzed as an infinite game (i.e. a game where the miners play indefinitely), where the game's average utility corresponds to the average utility of finite games. In this game, the players correspond to the miners, where each one possesses a given amount of computing power. All miners know the attack protocol of the adversary.

When the process begins everyone mines and it ends when no rational miner changes its strategy anymore. Thus, at the end of the process, rational miners reach a Nash equilibrium.

2.5 Game Theoretical Framework

As blockchains encompass several types of behaviour typical to economical systems, such as rational or selfish, it is important to be able to characterize their robustness in terms of resilience to such behaviours. The authors of [4] define a framework that characterizes the robustness of these systems in terms of both resilience to rational deviations and byzantine behaviors.

Before delving into further details, we present some fundamental concepts mentioned in the aforementioned work:

- *Joint strategy* - A joint strategy $\sigma = (\sigma_1, \sigma_2, \sigma_3, ..., \sigma_n)$ is a set of strategies such that the outcome pleases every player, and they have no incentive in changing their strategy σ_i;

- *Weakly dominated strategy* - A strategy is weakly dominated if, by choosing that strategy over any another, the player will have a lower outcome;

- *Strong resiliency* - A mechanism is strongly resilient if every subset of players has no incentive to deviate from the protocol;

- *Practical mechanism* - A mechanism (Γ, σ) is practical if σ is a Nash equilibrium p^* of the game Γ after deletion of weakly dominated strategies;

- *Optimal resiliency* - A mechanism is optimal resilient if it is practical and strongly resilient;

- *k-resiliency* - An equilibrium is k-resilient if no coalition of *k* players has an incentive to cooperatively change strategy;

- *t-immunity* - A property that guarantees that the utility of players is not inferior if at most *t* players deviate the protocol and play a different strategies;

- *t-weak-immunity* - A system is t-weak-immune if, for any altruistic player, the worst possible payoff received is the initial state (no matter how many players deviate from the prescribed protocol);

- *(k,t)-robust* - A system is (k,t)-robust if it is both k-resilient and t-immune;

- *(k,t)-weak-robust* - A system is (k,t)-weak-robust if it is both optimal resilient and t-weak-immune.

- *Game composition* - For different games *A* and *B*, the game *A* composed with *B* corresponds to players choosing a strategy from each game and receiving the sum of the utilities of games *A* and *B* as utility.

The definition t-immunity is too strong and often impossible to achieve in practice, which is why the authors present the concept of t-weak-immunity.

Since blockchain systems are designed in a modular way, the framework evaluates their robustness by analyzing the robustness of individual modules and inferring the properties of the system, through game composition.

One of the protocols evaluated by the game theoretical framework is a **cross-chain swap protocol**. In the latter, two users can swap assets that belong to two different blockchains which do not communicate with each other (e.g. Altcoin and Bitcoin).

In order to do so, two transactions must be published on each of those two blockchains. This process triggered once a private key *x* is disclosed between the two users. It is important to note that each of these two transactions must be published within a specific time interval, $\triangle_1$ or $\triangle_2$, depending on the corresponding blockchain.

Considering the condition proved in [57] and the context of [58], the authors of [4] conclude that $\triangle_1 >= 2\triangle_2$, which leads to the interval in which the transactions can be published - $[0, \triangle_2]$ (given that $\triangle_2$ is the minimum between $\triangle_1$ and $\triangle_2$).

Since blockchains are independent from each other, the protocol comprises two different mechanisms, one for each blockchain. Below, we present definitions for both [4]:

Definition 1 - The *Bitcoin game* is an extensive-form game (see 2.4) $\mathcal{G}_1$ with 2 players *N = A,B* and 5 nodes, 1 being the vertex (see Figure 2.5).

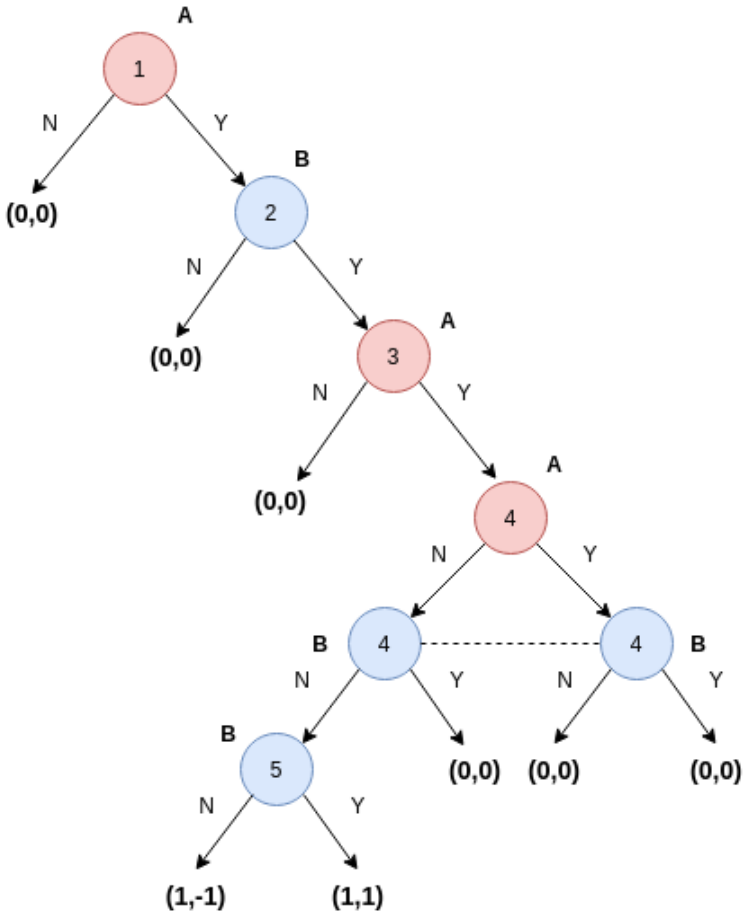

Figure 2.5: Game tree of $\mathcal{G}_1$ (adapted from [4])

In the Bitcoin game, each player can perform the following actions (where the order matters):

1. A can create TX1 and TX2 (Y) which leads to node 2, or not do it (N), with outcome (0,0);

2. B can sign TX2 (Y) which leads to node 3, or not do it (N), with outcome (0,0);

3. A can publish TX1 to the Bitcoin blockchain (Y), which leads to node 4, or do nothing (N), with outcome (0,0);

4. Both A and B can publish TX2 before *x* is revealed (Y) or not do it (N). If any of the two chooses action Y, the outcome is (0,0). Otherwise, A reveals *x* and (N,N) leads to node 5;

5. B can publish *x* on the Bitcoin blockchain (Y) with outcome (1,1) or not do it (N) with outcome (1,-1)

, where TX1 corresponds to a transaction that lets B receive a given amount of bitcoins if B provides *x*, and TX2 corresponds to a transaction that gives A back that amount in case B does not provide *x* within $\triangle_1$ hours.

In this scenario, the joint strategy that entails following the protocol is σ_1 = ({Y,Y,N}, {Y,N,Y}).

Definition 2 - The *Altcoin game* is an extensive form game $\mathcal{G}_2$ with 2 players *N = A,B* and 5 nodes, number 1 being the vertex (see Figure 2.6).

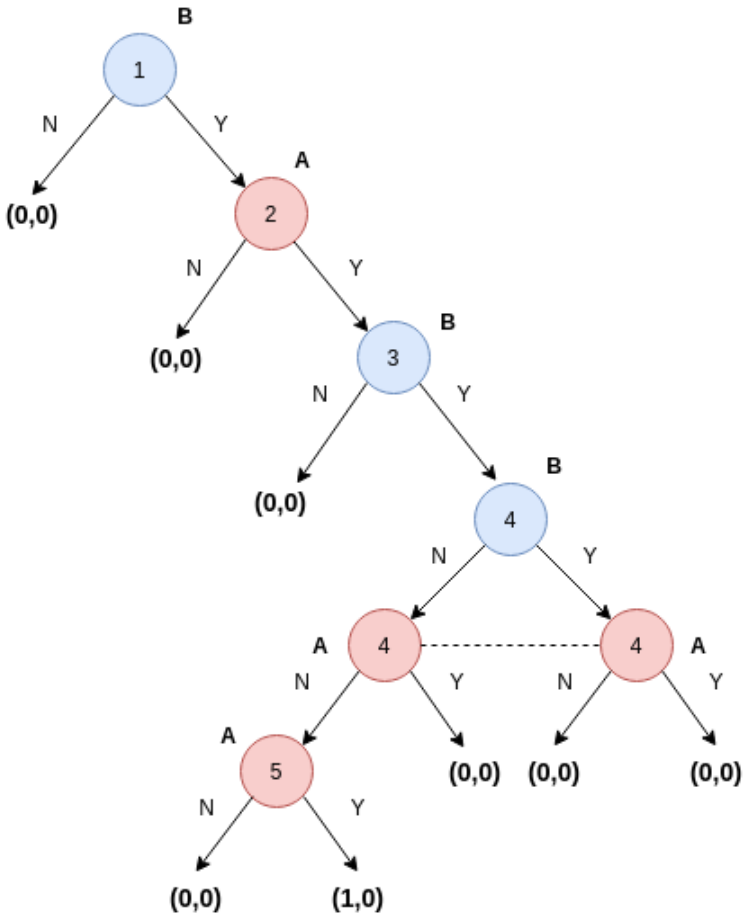

Figure 2.6: Game tree of $\mathcal{G}_2$ (adapted from [4])

In the Altcoin game, each player can perform the following actions (again, the order matters):

1. B can create TX3 and TX4 (Y), which leads to node 2, or do nothing (N), with outcome (0,0);

2. A can either sign TX4 (Y), which leads to node 3, or do nothing (N), with outcome (0,0);

3. B can publish TX3 on the Altcoin blockchain (Y), which leads to node 4, or do nothing with outcome (0,0);

4. Both A and B can publish TX4 before *x* is revealed (Y) or not do it (N). If any of the two chooses action Y, the outcome is (0,0). Otherwise, A reveals *x* and (N,N) leads to node 5;

5. A can either publish *x* on the Altcoin blockchain (Y), with outcome (1,0) or not do it (N), with outcome (0,0)

, where TX3 corresponds to a transaction that lets A receive a given amount of bitcoins if B provides *x*, and TX4 corresponds to a transaction that gives B back that amount in case A does not provide *x* within $\triangle_2$ hours.

Here, the joint strategy is σ_2 = ($\{$Y,N,Y$\}$, $\{$Y,Y,N$\}$).

The authors consider the full protocol to be the composition of the aforementioned two games. This protocol does not guarantee the property of t-immunity, given the fact that if one player deviates the protocol (e.g. if it does not create one transaction), the latter stops and can not proceed further.

On another hand, under the assumption that any transaction can be published in the time interval $[0, \triangle_2]$, we have that:

- In both games $\mathcal{G}_1$ and $\mathcal{G}_2$, σ_1 and σ_2 produce Pareto efficient outcomes (i.e. there is no other outcome that increases at least one player's payoff without decreasing the other's payoff) of (1,1) and (1,0) respectively;

- Given that the outcomes are Pareto efficient and the Nash equilibrium is strong, we have that both mechanisms are strongly resilient;

- Every strategy that is not σ_1 or σ_2 is weakly dominated (they provide either -1 or 0, which is less than 1), hence the mechanisms are practical;

- Being both practical and strongly resilient, the mechanisms are optimal resilient;

- In $\mathcal{G}_1$, when following respectively strategies $\sigma_1 A$ and $\sigma_1 B$, both A and B never get negative utility

(even if the other player deviates from the protocol). The same applies to $\mathcal{G}_2$ with $\sigma_2 A$ and $\sigma_2 B$. This means that both mechanisms are weak immune;

- Both games being optimal resilient and weak-immune, and considering that the full protocol is a composition of the latter, the full protocol is optimal resilient and weak-immune, thus it is (k,t)-weak-robust.

2.6 XCLAIM

XCLAIM is a framework that operates with *Cryptocurrency-Backed Assets (CbA)* to achieve interoperability in a trustless way (i.e. without the need of a centralized trusted third party).

The framework performs issuance, swapping, and redemption of assets. In order to do this while still maintaining a secure and decentralized solution, XCLAIM takes advantage of publicly verifiable smart contracts and *chain relays* (contracts that provide inspection and verification of transactions).

2.6.1 Definitions

There are several definitions fundamental to the understanding of the XCLAIM framework [14]:

- *Cryptocurrency-backed assets (CbAs)* - Assets that are deployed on top of a blockchain *I* that are backed by a cryptocurrency on a blockchain *B*.
- *b* - Cryptocurrency in blockchain *B*;
- *i(b)* - Asset present in blockchain *I* that is backed by b (CbA);
- *Issuing Blockchain (I)* - The blockchain on which the *i(b)* is issued;
- *Backing Blockchain (B)* - The blockchain that backs *i(b)* using cryptocurrency b;
- *Asset Value* - The amount of units of *b* used to generate *i(b)*.
- *Asset Redeemability* - Whether if units of *i(b)* are redeemable (i.e. recoverable) on *B* for *b*.
- *Asset Fungibility* - Whether or not units of *i(b)* are interchangeable. XCLAIM assumes that CbA's are fungible.

In XCLAIM, CbA's are *symmetric*, meaning that the total amount of units of *b* is equal to the total amount of units of *i(b)*, i.e. $|b| = |i(b)|$

2.6.2 System actors

The actors that make interoperability possible in XCLAIM are stated below [14]:

- *CbA Requester* - Locks a given amount of cryptocurrency on *B* to request a corresponding amount of CbA on *I*.
- *CbA Sender* - Owns a given amount of CbA and transfers ownership to another user on *I*.
- *CbA Receiver* - Receives and is assigned ownership over an amount of CbA on *I*.
- *CbA Redeemer* - Destroys a given amount of CbA on *I* to request the corresponding amount of cryptocurrency on *B*.
- *CbA Backing Vault* - A (non-trusted) intermediary responsible for fulfilling redeem requests of CbA for cryptocurrency on *B*. XCLAIM assumes that vaults are rational.
- *Issuing Smart Contract (iSC)* - The public (and verifiable) smart contract that is liable for managing the correct issuing and exchange of CbA on *I*. It also ensures the correct behaviour of the vault.

To ensure that the *CbA Backing Vaults* behave in a correct way (i.e. that they do not deviate from the protocol, e.g. stealing cryptocurrency) XCLAIM applies game theory techniques, implementing *collateralization* and *punishment*, enforcing a *proof-or-punishment* model [14]. In each process, if the *CbA Backing Vaults* do not present a valid proof stating the protocol was followed, they are punished.

To provide a practical example, we can assume the following scenario: Alice, that controls a given amount of Bitcoin on the Bitcoin blockchain, wants to create Bitcoin-backed assets and transfer them to Bob, that controls a given amount of Ether in Ethereum.

Alice first takes on the role of *CbA Requester* and sends one BTC to a *CBA Backing Vault* (i.e. Alice locks one BTC with the vault), so that the vault emits one CbA that represents that BTC in the Ethereum network.

The vault is incentivized to follow the protocol due to [14]:

1. *Collateralization and Punishment* - The process of a vault providing a given amount of its assets (a *collateral*) as a proof that it will properly behave. If the intermediary misbehaves (e.g. if it steals the assets), the *collateral* it provided is given to the damaged party;
2. *Fees* - If it behaves as expected, it will receive reward fees.

If the vault follows the protocol, it will, through the *iSC*, create and send the requested CbA to Alice. After this step, Alice will become the *CbA Sender* and, through the *iSC*, transfer the received CbA to

Bob. Then, still as the *CbA Sender*, Alice will atomically swap the CbA against Bob's ether on Ethereum. Finally, if Bob later wishes to redeem his unit of CbA for one Bitcoin, it will lock its CbA and, through the iSC, receive one bitcoin in return. The CbA is then destroyed.

It is worth noting that the vault only has an incentive to misbehave if the following is true [14]:

$$i_{col} < b.\varepsilon_(i, b) \tag{2.2}$$

Where i_{col} corresponds to the *collateral* provided by the vault and $\varepsilon_(i, b)$ corresponds to the exchange rate between the cryptocurrencies *i* and *b*.

2.7 CrowdPatching

CrowdPatching is a blockchain-based protocol (focused on permissionless chains, that are able to run smart contracts and comprise a cryptocurrency) that allows for updates to be delivered to IoT devices in a decentralized way [59]. The protocol delegates the distribution of updates to self-interested participants, called *distributors*, which are compensated in case the update is successfully delivered to the correspondent IoT device.

More specifically, the updates begin by being announced to the network through a smart contract (deployed by the manufacturers), that also issues cryptocurrency payments to any distributor that provides an unforgeable proof-of-delivery. This proof - a signature generated by an IoT device - is only provided if certain conditions are met. One of the conditions is the delivery of a specific type of proof - a zero-knowledge proof [60].

A *zero-knowledge proof system* is a protocol [59] in which a given *prover* wants to convince a given *verifier* that a certain statement is true, without exhibiting the secret values included in the content of that statement. Specifically, *CrowdPatching* uses *Zk-SNARKs* (Zero-Knowledge Succinct Non-interactive Arguments of Knowledge). This system comprises a light-weight verification process and makes it possible for proofs to be generated asynchronously and limited in size. Bulletproofs is a more recent alternative that doesn't require a trusted setup, unlike zk-SNARKs [61]. However, at the moment of writing, this system is not yet developed enough to apply in our protocol.

2.7.1 Protocol Entities

The protocol encompasses multiple entities [59]:

- *P2P file sharing network* - It is shared by every participant, and allows for peer discovery (through Distributed Hash-Tables);

- *Permissionless Blockchain* - The underlying blockchain where the updates are announced (through a smart contract) and through which the distributors and hub are compensated with cryptocurrency (if they follow the protocol);

- *Manufacturer* - Entity that initiates the protocol and desires to release an update for a set of IoT devices;

- *IoT Devices* - Devices that produce proof-of-delivery and proof-of-final-delivery (under specific conditions) and receive the updates from their manufacturer;

- *Hub* - A gateway that manages and is responsible for IoT objects in a local network, performing the most demanding steps of the protocol in behalf of them. *Hubs* are trusted by IoT devices but not necessarily by manufacturers, thus they are incentivized with cryptocurrency. They present a proof-of-final-delivery and are compensated if the latter is valid;

- *Distributors* - The self-interested participants who perform the distribution of updates from the manufacturer to the corresponding set of IoT devices. They present a proof-of-delivery and are compensated if the latter is valid. These can be divided into *first-hand distributors (FHD)*, that obtain the update file directly from the manufacturer and *second-hand distributors (SHD)* that obtain the update file through FHD, in exchange for cryptocurrency.

Note that the proof-of-final-delivery, produced by *IoT devices* and submitted by *hubs*, if valid, shows that the *IoT device* is finally able to see the update file.

It is important to note that, in order to provide flexibility, it is possible for IoT devices to perform both their own role and also the hub's role, eliminating the need for a *hub*.

In order to keep track of the reputation of *distributors*, *CrowdPatching* maintains a key-value store database, where each *distributor's* public key is associated with an integer number that corresponds to its score (when a *distributor* delivers for the first time, an entry is added to the database with the value 1).

2.8 Public Bulletin for Verifying Permissioned ledgers

This recent protocol [5] focuses on enabling an external party to observe and verify a permissioned blockchain's internal state, providing that there is at least one honest member present in the blockchain's committee. The state verification is achieved through the use of a secure public ledger that acts as a

bulletin board, in which snapshots of the permissioned blockchain's state (named "view" by the authors) are published at fixed intervals.

Besides the ability to assure the validity of state to the party, the protocol is able to reason about the currency of that state. The latter is influenced by the number of blocks *k* between each published view (i.e. with a smaller *k*, views are published more frequently thus external observers are provided with a greater time coherence with the ledger).

2.8.1 System Actors

- *Internal clients* - Members of the permissioned ledger which are not part of the management committee;

- *External clients* - Actors external to the ledger that need to obtain and verify states from the permissioned network;

- *Management committee* - Special members of the permissioned ledger with full access to the latter and which are responsible for its management.

Both the management committee and internal clients are considered participants of the permissioned network, while external clients are not.

When it comes to the management committee, three adversary scenarios are considered in this work:

- *Trustworthy Committee* - The committee as a whole is trustworthy, although some members might me malicious (the committee can tolerate faults);

- *Malicious but cautious committee* - The committee may behave arbitrarily, but only in situations where its behaviours can not be detected;

- *Malicious Committee* - The committee is fully malicious, and may behave arbitrarily at any situation.

On one hand, external clients do not have full visibility into the permissioned blockchain, which makes it hard to consider its committee to be trustworthy. On the other hand however, as committees are frequently formed by a respected consortium which wants to maintain its good reputation, it is also not viable to consider the latter to be fully malicious.

Therefore, the authors consider the committee to be malicious but cautious, with the assumption that at least one honest member exists in the latter. Otherwise, the permissioned ledger is completely unreliable.

2.8.2 Definitions

In this paper, Yu et al. define a permissioned ledger L^n as a state machine, which is replicated and managed by the management committee (according to defined policies). The machine transitions between states every time a sequence of transactions is executed. L^n is represented by a tuple in the following way:

Definition 1 - $L^n = (T_i, A_i, M_i, P_i)_{i=1}^n$,
where T_i corresponds to sequences of transactions, A_i to sets of possible application states, M_i to a set of management committees and P_i to a set of management policies.

Each state in the permissioned ledger $S_i \in S$ is also a tuple, represented by the following:

Definition 2 - $S_i = (A_i, M_i, P_i)_{i=1}^n$,
with a set A_i of application states, a management committee M_i and a management policy P_i. The management committee is responsible for state S_i, according to the management policy P_i. When a transaction is applied to state S_i, at least one of its three components will be updated, which results in the next state S_{i+1}.

It is important to note the difference between the internal state and the external view of a given permissioned ledger. The first is only visible to internal participants, while the second is the observation that external clients can make about the ledger. Moreover, for the same internal state there can be several (valid) external views.

In order to verify a given fact F against the internal state of a permissioned ledger, three entities must be involved - the management committee, an immutable public bulletin board where views of the ledger's state are published, and external clients that are interested in verifying the latter's state.

To achieve this, the authors proposed the following phases (see Figure 2.7):

1. *Generation and publication of views* - For each new block at height *i*, every committee member computes a digest of the corresponding state S_i. Then, from this digest, each node computes an external view, which corresponds to a tuple containing the digest, a signature over the digest and the corresponding block height.

   Every *k* blocks, a new round is initiated and some committee members can choose to publish external views for the current height in the public bulletin. The number *k* can vary, and changing it will influence both the cost of the protocol and the state coherence with the ledger (if it's higher,

the cost will be lower but also the state coherence and vice-versa).

It is important to note that, in reality, a publication to the bulletin board comprises more than an external view; it also contains a rolling hash. The latter contains all past views observed by the publishing member, up to the current view [5].

The existence of a rolling hash adds accountability to the publishing member, and provides a way to indirectly validate views from previous rounds (in which the member might not have participated in).

2. *Querying bulletin board for ledger view at time t* - An external client accesses the views of the permissioned ledger through the public bulletin. Since the latter allows the node to reason about the currency of state, the external actor will first query the public bulletin for a view at a given time *t*.

3. *Querying committee member for proof about facts against the ledger view* - After retrieving the view at a given time, the external client can query a committee member for a state against that view, and a proof about its validity.

4. *Verification of the proof by external client* - Once the client has the proof, it can verify its correctness. If the proof is valid, the received internal state can be trusted.

Besides these functions, the bulletin board tracks published views for accountability (through the rolling hashes associated with each published view), ensures that views do not conflict with each other and provides a platform for reporting conflicts.

The latter is especially important when there is only one honest member in the committee - if, in a given round, that member is not publishing any views but observes an invalid view being published, it will report the conflict to the bulletin board to warn other members about the event.

It is important to note that, although providing a platform for reporting conflicts, the paper states that these conflicts must be resolved externally (maybe with the use of a trusted third party).

2.9 ODAP

As mentioned in 1, ODAP is a cross-communication protocol that operates between two gateway devices to transfer assets between blockchains represented by those gateways. This asset transfer is unidirectional and comparable to atomic swaps, where an asset is *locked* on one blockchain and it's representation is created on another [6] [62].

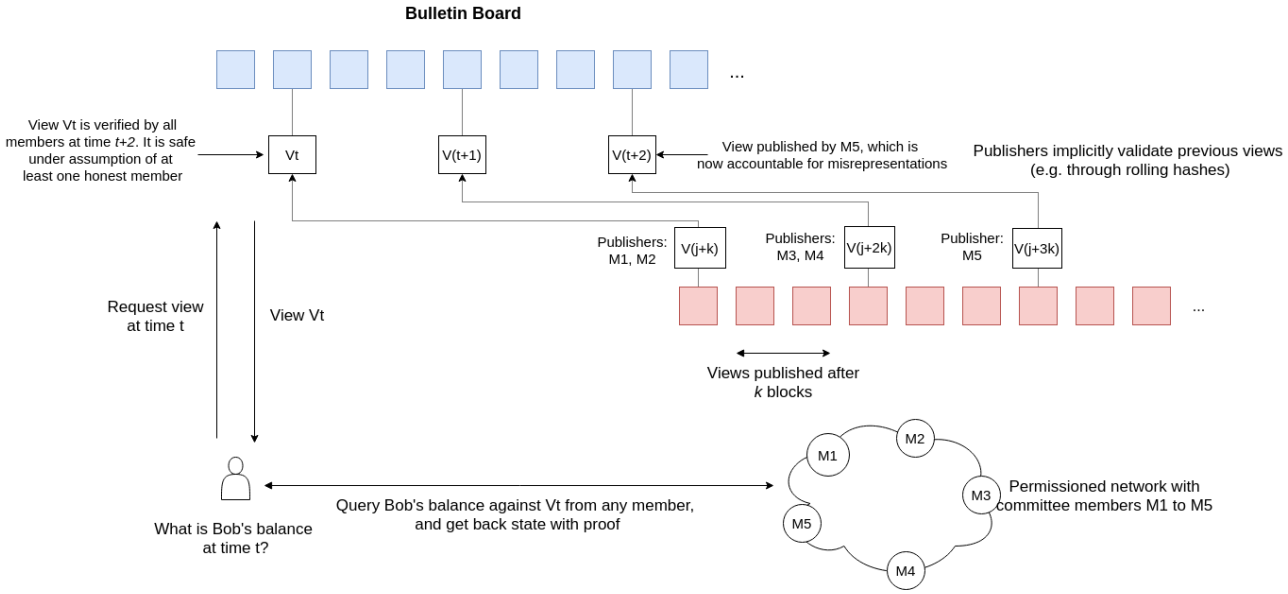

Figure 2.7: Public Bulletin protocol flow (adapted from [5])

2.9.1 System Actors

The protocol comprises the following actors:

- *Source ledger* $\mathcal{B}_S$ - Ledger that locks *x* units from asset type *a* so that they are created in the recipient ledger;
- *Source gateway* $\mathcal{G}_S$ - Gateway that transfers *x* units from asset type *a* to the recipient gateway;
- *Recipient gateway* $\mathcal{G}_R$ - Gateway that is the target of the transfer and follows instructions from the source gateway;
- *Recipient ledger* $\mathcal{B}_\mathcal{R}$ - Ledger in which the received asset's representation is created and becomes available.

Note that in ODAP, the target ledgers $\mathcal{B}_S$ and $\mathcal{B}_\mathcal{R}$ can be of any type (i.e. either permissioned or permissionless).

2.9.2 Protocol Flow

The simplified transfer process is depicted in Figure 2.8 and encompasses the following steps:

- *End-user triggers protocol* - Before the first phase, an end-user interacts with its local gateway $\mathcal{G}_S$ and informs it that it wants to perform an asset transfer.

- *Transfer Initiation Flow* - In this first phase, a secure channel is created, asset information is verified and identification procedures occur (such as each gateway resolving each other's identity, either directly or via a decentralized registry);

- *Lock-Evidence verification flow* - Then, in phase 2, the gateways exchange proofs regarding the status of the assets, and these proofs are persisted (which can be useful in resolving a dispute). The asset is also pre-locked, meaning that it will be transferred and thus it can not be used;

- *Commitment Establishment Flow* - In the last phase, the gateways commit to the asset transfer. Assets are locked (meaning that the asset is transferred and becomes unavailable in $\mathcal{B}_S$) and created on the target ledger $\mathcal{B}_\mathcal{R}$. This process avoids double-spend attacks.

In the transfer initiation flow phase, in order to verify asset information, the gateways leverage an *asset profile*. The latter corresponds to a generic schema, that presents an implementation-agnostic manner of representing a digital asset. This assures that heterogeneous ledgers refer to the same asset within a given transfer.

An asset profile contains several fields, including parameters such as an issuer, a digital signature, and keywords (list of keywords to make the asset easily searchable).

Besides the asset transfer process, ODAP presents a logging mechanism for fault tolerance (i.e. to recover from crashes) and also to provide accountability in case of an attack. Additionally, it comprises a distributed recovery mechanism named ODAP-2PC, a two phase commit protocol. We will not go into detail about ODAP-2PC since it is out of the scope of this thesis.

Both $\mathcal{G}_S$ and $\mathcal{G}_R$ (see Figure 2.8) are logging nodes and write log entry requests after certain operations, which can be of one of two types - *private operations* or *public operations*.

Private operations refer to operations executed solely by one gateway and thus are only known by that one gateway. In this context, three log entries are required - an entry that contains the intention to execute an operation, another that confirms that the latter was executed, and a third indicating if the operation succeeded or failed. This results in either (init-X, exec-X, done-X) if the operation succeeds, or (init-X, exec-X, fail-X) in case it fails.

On another hand, within public operations the state is known by more nodes than the one executing them. In this case, the log entries required are similar to the corresponding entries for private operations, with an additional acknowledgment entry for the non-executing nodes to acknowledge a given public operation. This results in either (init-X, exec-X, done-X, ack-X) if the operation succeeds, or (init-X, exec-X, fail-X, ack-X) in case it fails.

The authors also provide a log storage API in order to interact with the log storage while abstracting storage details.

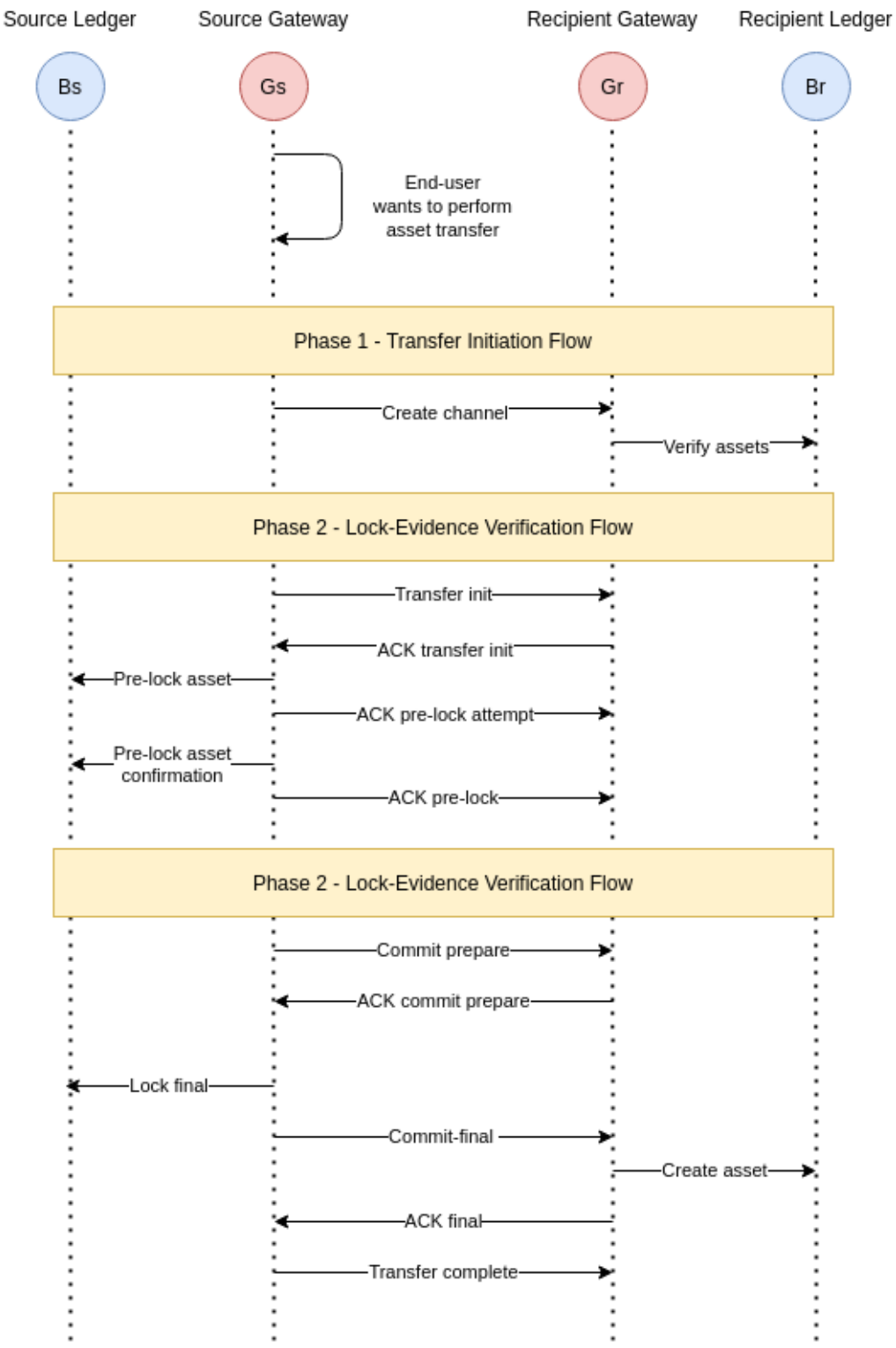

Figure 2.8: Simplified sequenced diagram illustrating ODAP (adapted from [6])

2.10 Comparison of Existing Solutions

Table 2.1 compares the aforementioned solutions using different metrics, more specifically whether each solution:

- Supports permissionless blockchains;
- Supports permissioned blockchains;
- Has a high level of trustlessness, meaning that the end-user does not need to trust any participant or group of participants in the system [63] (see 3.1 for a more in-depth definition);
- Supports smart contracts;
- Presents a reward mechanism;
- Presents a punishment mechanism;
- Is able to verify the internal state of a permissioned blockchains.

XCLAIM is flexible and adapts to both permissionless and permissioned blockchains (in case they leverage Proof-of-Authority consensus), provided that the issuing blockchain *I* supports Smart Contracts (SC).

The solution does not place trust in the vaults (they are assumed to be rational), but instead, it places trust in the protocol and the incentives it provides to the latter. XCLAIM supports smart contracts - it leverages the iSC contract in the protocol, which controls the behavior of the vaults and resides in the issuing blockchain I.

Additionally, as mentioned before, XCLAIM comprises several mechanisms that enforce the good behavior of the vaults. It uses collateralization as a punishment mechanism and, in a way, as part of the reward mechanism, since the *collateral* is given back to the vault if it follows the protocol.

Finally, it does not implement verification of permissioned blockchain's internal state, which can be unsafe. Instead, it states that chain relays - a mechanism that provides external blockchain data (such

Paper	Public blockchains	Private blockchains	Trustless	Smart Contracts	Reward mechanism	Punishment mechanism	Verify internal state
XCLAIM [14]	✓	✓*	✓	✓	✓	✓	✗
CrowdPatching [59]	✓*	✗	✓	✓	✓	✗	NA
Public Bulletin [5]	NA	✓	✓	✓	NA	NA	✓
ODAP [6]	✓	✓	✗	✓	✗	✗	✗
T-ODAP	✓	✓	✓	✓	✗	✓	✓

Table 2.1: Comparison of related work solutions

as transactions in the backing blockchain *B*) to the smart contract that executes on *I* - can be used for permissioned systems as well, in case they leverage a Proof-of-Authority consensus type.

Despite its strengths and potential, XCLAIM cannot resolve this thesis's problem since it only supports a limited type of permissioned blockchains. Moreover, the use of chain relay to provide external data from a permissioned blockchain might not be secure - as these blockchains are opaque, one can not know if the information directly provided from the inside is trustworthy. This is improved in our solution. Furthermore, in order to deal with fluctuating cryptocurrency rate exchanges (ensuring that *redeemability* remains possible) XCLAIM provides (among other mechanisms) *over-collateralization*. This mechanism translates into the use of a buffer to be able to tolerate sudden exchange rate drops. It helps to mitigate the rate fluctuation, however, due to the higher amount of locked-up capital, it can incur opportunity costs [64].

CrowdPatching supports only permissionless blockchains that support smart contracts and cryptocurrency. The solution operates with smart contracts, that announce the information of new updates to the network and issue payments for distributors and hubs that follow the protocol.

It it trustless, given that it leverages public escrow parties to transport IoT updates, instead of transporting the updates directly to the IoT devices. As XCLAIM, instead of trusting each party directly, it trusts the protocol and that the incentives given to them are enough.

Its reward mechanism comprises compensating its well-behaved participants through cryptocurrency payments, issued by a smart contract.

Finally, as the solution is intended for IoT devices and not blockchains, the verification of permissioned blockchains is not applicable.

CrowdPatching presents some drawbacks. It does not contain a punishment mechanism, which may harm the solution's security, since being incentivized with cryptocurrency might not be enough for the distributors to follow the protocol, and the leveraged proofs (zk-SNARKs) present some challenges, including their performance (requiring a trusted setup for each program in a smart contract, each contract requiring a high amount of data) and security (requiring to trust a third-party manager) [65].

The public bulletin work has the goal of verifying the internal state of permissioned ledgers, thus whether it supports public blockchains or not is not applicable.

The protocol removes the need to trust permissioned blockchains and their committees, by leveraging a smart contract which presents proofs of internal states. Thus, it is considered trustless.

Reward and punishment mechanisms are not applicable in this context, since the goal of the bulletin is merely to present the proofs provided by the permissioned blockchain's committee members, and to be a platform for reporting conflicts. We do not need to incentivize the permissioned blockchain to follow

the protocol, because if it provides an incorrect proof, it is invalid and thus not published in the public bulletin.

The last work in the state-of-the-art corresponds to ODAP (Open Digital Asset Protocol). The latter is rather flexible, allowing blockchains of both types (both permissioned or permissionless) to transfer assets to each other.

In ODAP Hermes, gateways are trusted, and it is assumed that they will not drop an asset before a given transfer or that they will not transfer it to the wrong gateway.

The protocol supports smart contracts since it supports gateways that interact with the latter.

Since in this protocol the gateways are trusted, it does not require game theory techniques such as reward or punishment mechanisms to incentivize the gateways.

The most significant disadvantage of ODAP is the fact that the gateways trust each other. Moreover, it is not able to verify permissioned blockchain's internal state, which might not be secure for the same reasons presented when talking about the disadvantages of XCLAIM.

A more trustless solution, that can be applied to private (and public) blockchains, is necessary.

Because of this, in our work, T-ODAP, we build a more secure and completely trustless version of ODAP through the use of a DVS (based on [5] and implemented in Polkadot), in which proofs are published about the state of the ledgers.

This way, before a *source gateway* executes a transfer of an asset to a *recipient gateway*, the *recipient gateway* can verify the *source gateway*'s internal state to ensure that the latter has enough funds to perform the operation, thus not having to trust the *source gateway*.

As an additional security layer and in order to incentivize the gateways to follow the protocol, T-ODAP comprises a punishment mechanism based on reputation - if a gateway misbehaves, its reputation is diminished and thus that gateway becomes less likely to be chosen in future iterations of the protocol. We believe that a reward mechanism does not make sense in our system since the gateways have their functions pre-established and already know that they have to follow the protocol.

In order to connect to the underlying platform Polkadot, our work also entails a Polkadot connector implemented on Hyperledger Cactus.

In the next chapter, we present the design and implementation of T-ODAP.

3

T-ODAP: Decentralizing Asset Transfers

Contents

In this chapter, we propose a solution for the problem introduced in Chapter 1. Recall that the main goal of this work is to provide a secure and robust system that allows for trustless permissioned blockchain interoperability through the use of the Decentralized View Storage. In order to promote robustness, we will model the system as a game with regards to its participants.

We discuss the requirements that our solution must comply with, followed by the assumptions made in the system. Then, the proposed architecture and design is introduced and discussed, followed by the protocol flow and the threat model.

3.1 Requirements

T-ODAP must provide a secure and trustless protocol that enables asset transfers via gateways.

Several non-functional requirements are desired:

- *Security* - The protocol should ensure that the processing and delivery of the assets is secure, namely assuring that gateways follow the protocol and do not tamper, drop or re-direct the assets to wrong gateways. In order to do this, the protocol enforces the use of the decentralized view storage, where proofs about both gateways' states are frequently published.

- *Compatibility* - T-ODAP should be compatible with several permissioned (and permissionless) blockchains that support smart contracts with functionality for locking and unlocking assets.

- *Trustless* - The protocol should provide a trustless solution, i.e. a solution in which a participant does not need to trust any other participants in the system in order to maintain security of its assets and other expectations of functionality, only needing to trust the protocol, mathematics, cryptography, code and economics [63]. In our case, this means that the two gateways in any protocol instance do not need to trust each other in order to perform an asset transfer;

- *Availability* - T-ODAP should be working in proper conditions, with a minimal downtime;

- *Testability* - It needs to be possible to test the system in a safe environment (e.g. a non-production environment emulating a high workload);

- *Privacy* - The protocol should only be able to provide state proofs about the internal state of each participating gateway, and not any other that is not involved.

- *Efficiency* - The solution should be efficient, i.e. despite encompassing additional steps in relation to ODAP, these additional steps should not affect performance severely.

T-ODAP must be instantiated by gateways belonging to blockchains that support smart contracts with functionality for locking and unlocking assets.

We also present a set of functional requirements for our solution. Those include:

- Store proofs of a permissioned blockchain's internal state;
- Verify those proofs of state;
- Deploy a smart contract to Polkadot;
- Connect blockchains to Polkadot;
- Perform cross-chain asset transfers;
- Rollback an asset transfer.

The use cases that the solution should tackle are illustrated in Figure 3.1, which also includes the system actors described in Section 3.3.3.A.

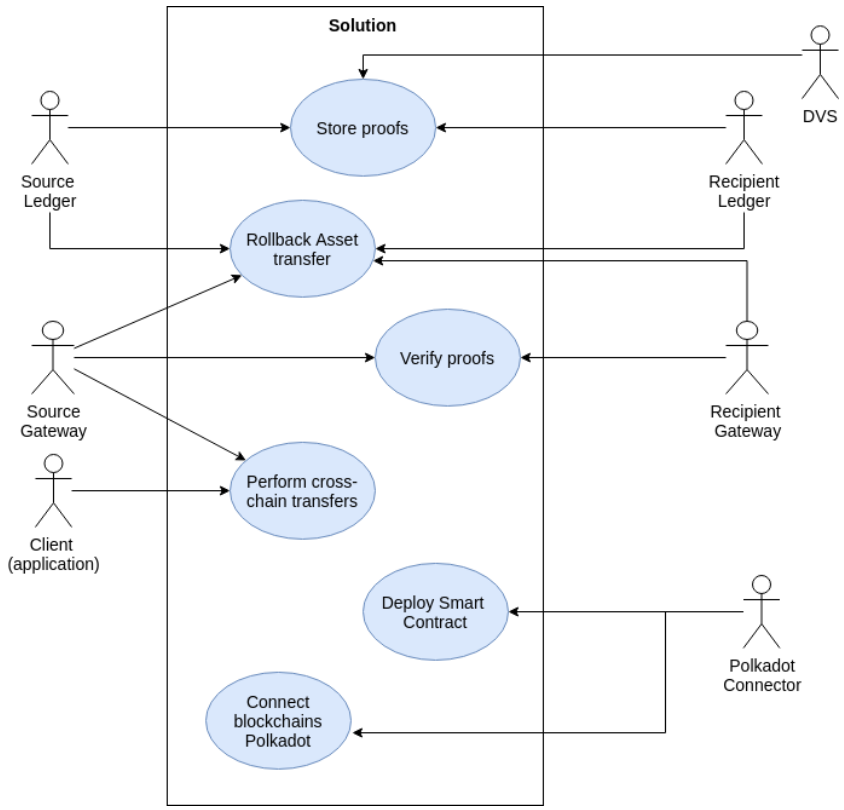

Figure 3.1: Use Cases Requirements T-ODAP

3.2 Assumptions

Similarly to related work [14], we assume that adversaries (in this case, the gateways) are computationally bounded and rational agents, motivated by actions that increase their utility and avoiding actions that decrease their utility. As such, the latter can attempt to perform any attack that potentially maximizes their utility, such as not completing an asset transfer. In our context, we assume that a malicious node is any node which deviates from the established protocol T-ODAP. In terms of the network, we assume that honest nodes are well-connected and there is a maximum delay in which they receive transaction broadcasts from users. When it comes to the DVS, we assume that each permissioned network comprises at least one auditor node (a member of that network) which solves conflicts between views, deciding which views are valid and which are not. Thus, we assume all views published in the DVS are valid (i.e. that they correspond to the correct internal state of the corresponding permissioned blockchain).

3.3 System Overview

Our solution is composed of several layers that stack on each other - the DVS, the Polkadot Connector and T-ODAP. This scheme is depicted in Figure 3.2.

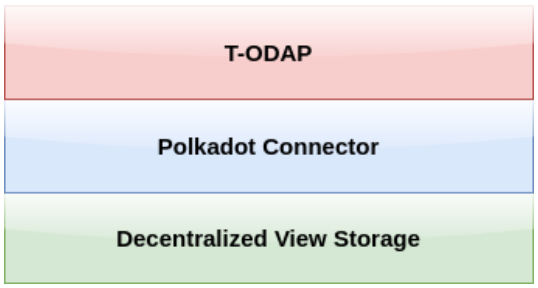

Figure 3.2: Layers that build T-ODAP

3.3.1 Decentralized View Storage

In the bottom layer of Figure 3.2, we have the DVS. The latter is based on the Public Bulletin for permissioned ledgers presented in Section 2.8, a bulletin where internal state proofs of a permissioned blockchain can be published by the corresponding blockchain's committee members. These state proofs (views) are publicly available for external clients to observe and verify facts against. In practice, they correspond to a digest of a permissioned blockchain's internal state.

The necessity for a Public Bulletin or a DVS stems from the fact that permissioned blockchains are closed systems. In order to allow for a truly secure and trustless interoperability between permissioned

blockchains (or between a permissioned and a permissionless chain), there needs to exist a system which securely shares the state of the latter for external observers.

Similarly to the Public Bulletin presented by Yu et. al. [5], the DVS (Decentralized View Storage) is an immutable public bulletin where state proofs of a permissioned blockchain are regularly (i.e. every *k* blocks) published by its corresponding committee members.

It also considers a malicious but cautious committee, along with at least one honest member in the latter, which reports a conflict if it witnesses malicious behavior. However, there are some key differences.

In the context of the Public Bulletin, as stated by the authors, it is possible that distinct valid external views exist for the same state. However, for a given height, this bulletin board's algorithm only publishes a new view in the board in case it is equal to the other views published for that same height. Otherwise, it reports a view conflict that has to be solved externally.

The DVS, however, provides an improvement to the aforementioned method used by the Public Bulletin - instead of reporting a conflict immediately once a new view is different from other views seen in a given height, our algorithm encompasses a voting mechanism. A quorum of members vote on the view and the collective decision determines if that view is either valid or inconclusive, case in which a view conflict is reported which must be solved externally, by an auditor node.

This auditor node exists in each permissioned network, and corresponds to a node which function is to decide if a given view is valid or not. We understand that the existence of this validator node can be seen as contradictory having our goal of trustlessness in mind, however it was the solution we found within our limited time. This is an aspect that can be improved in future work.

Since there is at least one honest member in the committee, even in the worst possible scenario in which all committee members but one collude and try to publish an invalid view, the honest member will vote negatively on that view. For this reason, in our algorithm, one negative vote is sufficient to raise a conflict on the view. Please note that even if the honest member does not belong to the voting quorum, it can still report the view when it detects it is invalid. On another hand, this means that if a single malicious committee member votes negatively on a valid view, even if the other members vote positively on that view, a view conflict will be reported.

The aforementioned conflict resolution is not resolved by the DVS, it is resolved externally. Once it is resolved, the view is deemed either valid or invalid - and the member or members which voted contrarily are recognized as being malicious. In future work, we can implement a mechanism to punish these members in order to discourage them from performing malicious actions again, but for now this mechanism does not exist in the DVS.

It is a trade-off between security and convenience - since permissioned blockchains are opaque, there is no way to know for sure if a member is malicious or not. For this reason, the Decentralized View

Storage always assumes the worst case scenario.

In terms of the DVS's publishing frequency, we take into consideration that the number k should be adjusted depending on the blockchain leveraging it; for a blockchain in which blocks are frequently added to the chain, the number should be higher. But a blockchain with a smaller transaction frequency should work with a lower k value (otherwise, in this case the DVS would publish views with too long a time gap).

This mechanism is implemented in the form of a smart contract and deployed in the Polkadot network (see Section 4.2 for more details on the implementation decisions).

3.3.2 Polkadot Connector

The Polkadot connector emerges on top of the DVS layer, as a bridge for permissioned blockchains to be able to access Polkadot.

This is possible since the connector is part of Hyperledger Cactus (see Section 2.3), an interoperability mechanism which is, at the moment, compatible with several permissioned blockchains including Quorum [19], Corda [20] and Hyperledger Fabric [2]. Thus, a permissioned blockchain supported by Hyperledger Cactus can use the latter to access Polkadot through this connector.

The connector also implements mechanisms to deploy smart contracts to the Polkadot network and to interact with them, by being able to call read and write function from those contracts.

Since the DVS is implemented in the form of a smart contract and deployed in Polkadot, the Polkadot Connector is able to interact with its functionality - given that it receives the correct parameters, it can retrieve or publish a specific view in the DVS.

Beyond providing permissioned blockchains with the ability to connect the DVS - allowing them to perform a secure internal state sharing with external clients - we contributed to an open source project and created a connector with a more general impact, allowing several blokchains to connect to Polkadot, to deploy smart contracts of their choice in the latter and to interact with them.

3.3.3 T-ODAP

As mentioned in Section 2.10, T-ODAP arises as a trustless version of the existing protocol ODAP, leveraging the use of a DVS for permissioned blockchains' internal state sharing. This way, T-ODAP does not require that gateways trust each other since they can verify each other's state in the DVS, prior to any asset transfer occurring.

Note that T-ODAP is compatible with both permissionless and permissioned blockchains, however it is focused in the latter. This is because the DVS is necessary for proving the internal state of permissioned blockchains, but not needed for permissionless ones given that these are publicly verifiable.

3.3.3.A System Actors

The following actors exist in T-ODAP:

- *Source Ledger* B_S - The ledger that desires to transfer an asset to the recipient ledger, by locking *x* units from asset type *a* to be created in the latter;
- *Source Gateway* G_S - The gateway that transfers the locked *x* units from asset type *a* to the recipient gateway;
- *Decentralized View Storage (DVS)* - The immutable bulletin where a permissioned blockchain's internal state proofs are published regularly;
- *Recipient Gateway* G_R - The gateway that responds to G_S and is the target of the transfer;
- *Recipient Ledger* B_R - The ledger that receives the asset transfer, by creating the corresponding tokens in its ledger and making them available.

The existence of the DVS allows G_R to verify B_S's internal state before an asset transfer; more specifically, verifying if it is indeed true that *x* units of the asset of type *a* are locked in the source ledger B_S. If that is the case (meaning that G_S was honest), the algorithm proceeds normally and the transfer completes after the necessary steps (see Section 3.4.2). Otherwise, we are in the presence of malicious behavior and the asset transfer is rolled back, thus not being finalized in G_R and corresponding tokens not being created in B_R.

These actors are also illustrated in the aforementioned Figure 3.1.

3.3.3.B System Model

As mentioned in Section 3.2, we consider gateways as economically rational agents, playing to receive the maximum possible utility.

Similarly to [64], we define two possible action choices for each rational player:

- *Desired action* - An action α_A performed by agent A, where $\alpha_A \in \sigma_d$ and σ_d describes the set of actions that are in accordance with the specification of the T-ODAP protocol;
- *Undesired action* - An action α_A performed by an agent A, where $\alpha_A \in \sigma_u$ and σ_u describes the set of actions that are not in accordance with the specification of the T-ODAP protocol. No action at all ($\alpha_A =$) is also considered an undesired action.

In each step of our protocol, each gateway must decide if it will either choose an action $\alpha_A \in \sigma_d$ or an action $\alpha_A \in \sigma_u$. If both parties choose desired actions in every step of the protocol, the asset transfer is completed and the protocol ends with success. Otherwise (see Section 3.5 for specific threats), the

asset transfer is rolled back.

We previously saw that rational agents are motivated by actions that increase their utility and unmotivated by the actions that decrease it.

In order to provide a secure protocol and motivate the players to choose desired actions instead of the contrary, T-ODAP punishes a gateway each time it chooses an undesired action. This punishment consists of decreasing that gateway's public reputation, making it less likely that it is chosen in the next T-ODAP instance. The latter has a negative value associated to it, which will decrease the player's overall utility.

Please note that would not make sense to offer the agents a reward or incentive to choose desired actions, since following this protocol is their only purpose.

There are several factors (based on [64]) that contribute to the utility attributed to an action $u(\alpha_A)$ by an agent A:

- *Cost (c)* - The cost associated with the action. This includes costs derived from transactions or the opportunity cost for locking *x* units of an asset of type *a* that could be leveraged in a different protocol to earn an interest;

- *Valuation (v)* - A value that encodes a specific, private preference of agent A for an outcome depending on the action.

- *Punishment (p)* - A value that encodes the punishment associated with a given undesired action.

Note that the valuation is related with the preference (or not) for a certain malicious action and can either be influenced by external actors (e.g. the agent can be bribed to perform an undesired action) or intrinsic to the agent (it might have a specific motivation for malicious behavior).

Having this in mind, we calculate the utilities as following:

$$\begin{cases} -c & if\ \alpha_A\ in\ \sigma_d, \\ v-c-p & if\ \alpha_A\ in\ \sigma_u \end{cases} \tag{3.1}$$

Where c corresponds to the cost (including opportunity costs, if the action comprises locking assets), v to the private valuation (i.e. the preference for an outcome) and p to the punishment for choosing an undesired action.

In terms of utility value, a desired action has to discount the cost and has no incentive as a positive utility. In turn, an undesired action has the valuation with a positive value, however has to discount not only the cost, but also the punishment value.

In this context, to guarantee that a rational agent will choose a desired action, we have to make sure that the utility value of that action is higher than the one of a undesired action, i.e., the following must be true:

$$-c > v - c - p \implies p > v, \tag{3.2}$$

As there is no way to know the actual value of the valuation (since it is private) and given that it might be challenging to predict it, we consider two types of rational agents and their relationship with the valuation and punishment values (based on [64]):

- Type A_d - An agent that consistently follows desired actions, since the utility from a desired action is always higher than the utility for an undesired one, i.e. $p > v$ (the utility lost by the agent in case it does not follow the protocol is higher than in case it does).
- Type A_r - An agent that is undecided in its decision between a desired and an undesired action, leading to $p \approx v$.

The other possible type would be an agent that consistently follows undesired actions, given that its valuation for an undesired action is always greater than the punishment for choosing the latter. This type of player is not relevant to our system since a fully byzantine player can never be incentivized to follow the protocol.

However, T-ODAP can guarantee that both types A_d and A_r will choose a desired action every time the latter has a greater utility than an alternate undesired one.

3.4 Protocol

We now discuss the design and architecture of T-ODAP protocol. Figure 3.3, built with the Archimate language [66], illustrates the latter. In this figure, we can observe the several components forming T-ODAP's architecture, which are divided in four different groups for a better understanding.

The first group (on top) comprises the source ledger B_s, as well as an asset of type A and the source gateway G_s, which executes the asset transfer. The source gateway is a specialized type of Cactus Node (a node belonging to Hyperledger Cactus - see Section 2.3), and it can be defined as "a computer system in a blockchain network for the purpose of assisting in the movement of virtual assets into (out of) the blockchain network" [67]. The end-user is connected to G_s. The latter is the component which triggers the whole protocol, by issuing a CC-Tx asset transfer request. This request is associated with the transfer of *x* units of an asset of a given type *A* from B_s, which (if the protocol is successful) will be created as *y* units of an asset of given type *B* in the recipient ledger B_r.

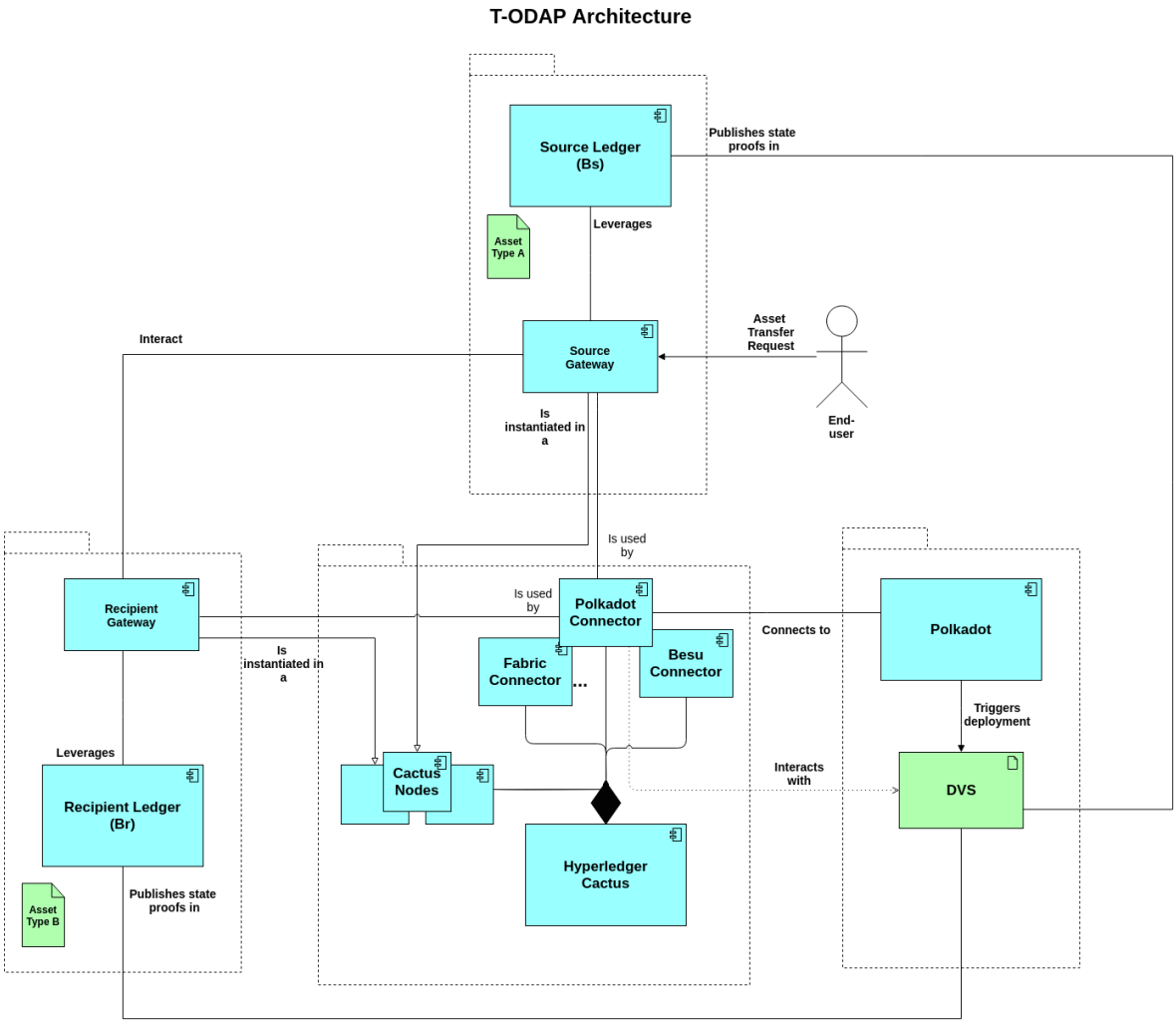

Figure 3.3: Archimate T-ODAP Protocol Architecture

The second group is similar to the first one, however this one comprises, instead, a recipient ledger B_r, the corresponding recipient gateway G_r which interacts with G_s and which is the target of the transfer, and the resulting created *y* units of asset of type *B* in B_r. This group is not directly connected to an end-user.

Both gateways interact with each other, being that G_s is the one that initiates the connection.

Then, we can observe the third and fourth groups. The third group encompasses Hyperledger Cactus and its several connectors to blockchains/interoperability mechanisms (not all are represented), as well as its several Cactus Nodes. The fourth group comprises the Polkadot network and the DVS smart contract, deployed in it.

In order to (indirectly) access the DVS, the gateways have to leverage the Polkadot Connector. Before the protocol instance begins, the connector connects to Polkadot and deploys the contract code containing the logic of the DVS. Then, G_s and G_r can use the connector to retrieve and read views from

it, analyzing the state of B_r and B_s, respectively.

B_s and B_r are also connected to the DVS since, in order to guarantee the integrity of the views, the latter must be published by members of the blockchain itself and not by the gateways. If the gateways were able to publish views, since the latter can be malicious, we would not be able to be certain that the published views were always correct.

3.4.1 Cross-chain digital assets

Let us dive into the life cycle of the cross-chain digital assets transferred from one blockchain to another.

As previously discussed, in a simplified manner, an asset transfer between two gateways entails, on one hand, B_s locking *x* units of an asset of type *a* and, on another hand, B_r creating a representation of that asset corresponding to *y* units of an asset of type *b*.

This means that an asset goes through a transformation from the moment it is locked in the source ledger, to the moment its representation is finally created in the recipient ledger. In order to facilitate the understanding of the cross-chain digital asset's life cycle, we consider that in a given asset transfer, there is only one asset, which can have one of three states:

- *Pure asset* - The asset's initial state. It is still unlocked in the source ledger B_s and thus accessible;
- *Blocked asset* - The asset's intermediate state. It is locked in the source ledger B_s and thus inaccessible. A blocked asset can be (based on [6]):
  - *Pre-locked* - The asset will be transferred, and it becomes temporarily inaccessible. In case the protocol fails before the transfer (i.e. in case G_r rejects the asset transfer) the operation is rolled back and the asset goes back to the pure asset state;
  - *Locked* - The asset was transferred and thus it is not accessible. This lock holds until there is an unlock (controlled by smart contract logic), in order to protect the protocol from malicious parties using the asset while it is locked;
  - *Burnt* - The asset was destroyed or permanently locked.
- Digital twin asset - The asset's final state. It corresponds to the representation of the original asset, created in B_r. After being created, it becomes accessible.

As in ODAP (see Section 2.9), all asset related information is encapsulated in an asset profile. The latter includes information such as an asset code (a unique code that allows for easy identification) and a digital signature emitted by the asset's issuer.

3.4.2 Flow

We will now discuss the protocol flow of T-ODAP, illustrated in figure Figure 3.4.

Before diving into the protocol steps, we will present the types of actions that a gateway can perform (based in [6]).

In terms of off-chain actions, we have:

- *Commit* - To commit on a given asset transfer. This action takes place in case the protocol was successful, and in case both gateways followed desired actions;
- *Rollback* - To go back to the state prior to the operation (i.e. the asset goes back to pure state). A rollback can occur if G_r rejects the asset transfer, or in case G_s does not lock the asset when it should (an attack);
- *Complete* - To finish the asset transfer, once the digital twin asset is successfully created on B_r.

When it comes to on-chain actions, a gateway can alter an asset's state through the actions pre-lock, lock, unlock and create asset, as seen in Section 3.4.1.

Figure 3.4 illustrates an example of T-ODAP's protocol flow, having Fabric as the source ledger and Quorum as its recipient counterpart. Here, we can observe the main differences in relation to ODAP:

- DVS is a participant;
- Phase 3 - View Publication Flow - is introduced.

We can also observe that an if condition is introduced at the bottom, which depends on the outcome of the last step of Phase 3.

First, an end-user (i.e. an application) issues a CC-Tx asset transfer request through Hyperledger Cactus (see Section 3.4), which triggers the beginning of the protocol. Then, Phase 1 (Transfer Initiation Flow) and 2 (Lock-Evidence Verification Flow) take place; these remain unchanged from what was presented in Section 2.9 - the first phase leverages initiation processes, necessary for connection between the gateways and the second phase takes care of taking the asset from a pure state to a blocked state (more specifically, pre-locked), along with verifying that the recipient ledger is indeed interested in receiving the asset transfer.

Then, Phase 3 begins. Here, we begin with Hyperledger Fabric (B_s) publishing a view at a given time t (note that views are frequently published, with the value *k* depending on the source blockchain). This step is particularly important since it shares the internal state of Fabric at that moment in time, and since this view contains information about the state of the asset to be transferred.

The protocol proceeds with G_r retrieving Fabric's most recent published view, in order to be able to analyze its contents and confirm that the asset is indeed in a blocked state. Note that, in this stage, G_r

only retrieves the view after a given time *t* has passed. This amount of time depends on the blockchain B_s. This is due to the fact that even if the asset is locked, the view containing this information might only be published after some time, or the network can have some delay. To guarantee that the retrieved view contains the correct and most recent information about the lock, we wait *t* units of time.

The outcome of this verification triggers one of two options within the protocol:

- If the asset is indeed locked, Phase 4 (Commitment Establishment Flow) takes place. This phase comprises a preparation commit, a final lock (by B_s), which sets the asset's state from pre-locked to locked (both in blocked state) and a final commit of the transfer, containing all the information necessary for B_r to create the asset.

  After G_r claims that the asset was created in Quorum, Phase 5 (Asset Creation Verification Flow) starts. Here, G_s will retrieve Quorum's most recent published view (again, waiting *t* units of time before doing so) and verify if the information provided by G_r is correct. In case it is, the transfer process finishes with success, having the asset in its final state - digital twin asset. Otherwise, G_r attempted to execute an attack by not creating the asset in the Quorum blockchain. The transfer is rolled back and the blocked asset in B_s is set to a pure state again, so that it is not lost.

- If the asset is unlocked, this means that G_s opted for an undesired action. The transfer has to be rolled back; otherwise, by creating a representation of the asset in B_s, double spend would occur.

3.5 Threat Model and Security Analysis

As we previously saw, gateways have the choice to perform either desired or undesired actions. The choice of an undesired action entails a punishment, which varies according to the action taken. Both gateways may have an incentive to deviate from the protocol (thus, execute an attack).

We will now present the threats that compose the threat model and, for each threat, perform an informal security analysis of our protocol's behavior towards it. For each threat presented, we explain its context, associate it with a level of severity from 1 to 5 (where 1 corresponds to very low and 5 corresponds to very high severity) and, finally, delve into how T-ODAP offers protection from that threat. Note that each threat corresponds to an action that can be performed by a malicious node.

Threat 1 - *The source gateway G_s steals the asset to be transferred (it does not lock the asset before transferring it to G_r).*

Let us imagine G_s is meant to transfer an asset to G_r, so that the latter creates the asset's representation in B_r. G_s can try to steal the asset by not providing instructions for B_s to lock the asset, while

lying to G_r about locking it. This way, B_r will still end up creating the asset's representation (the digital asset twin Section 3.4.1) although the asset has not been locked in the source ledger.

This threat has a level 5 of severity (very high) since it allows for double spending (i.e. the same digital asset being used more than once).

T-ODAP mitigates this attack through the use of the Decentralized View Storage. As we have previously seen in Section 3.4.2, the latter allows for the removal of trust between the gateways, since the recipient gateway G_r can observe the source ledger's internal state (including the asset's state) prior to the transfer, so that it can stop the latter in case the asset's is incorrect.

Threat 2 - *The source gateway G_s steals part of the asset to be transferred but transfers the remaining portion.*

This threat is a slight variation of the previous. In this context, imagine the asset transfer comprises transferring 5 units of token of type *A* to be created as *x* units of token of type *B* in G_r. The source gateway can try to lock only 3 of those units and steal the remaining 2. The transfer will still take place, since G_r believes that B_s locked the entire asset.

This threat has a level 4 of severity (high) since it leads to a similar result as threat 2 - double spending still occurs with a portion of the asset, even though it is not the whole asset.

T-ODAP mitigates this threat as it mitigates threat 1 - the recipient gateway can verify B_s the exact amount of token units that must be locked. If this number does not match what is expected, the transfer is rolled back.

Threat 3 - *The recipient gateway G_r does not create the assets in the recipient ledger.*

Here, the threat is focused on the recipient gateway, which performs a denial-of-service attack by not creating the assets in B_r. The latter can be executed by a malicious G_r that desires to harm the users of the source gateway, the source gateway or both by causing them to lock funds that will never be created in B_r. Despite not having a monetary incentive (given that the assets are not created), the malicious intent towards the participants can suffice as an incentive for the attack (i.e. the valuation value is high for this attacker).

This threat has a level 5 of severity (very high), considering that the initiating user will lose access to the funds and the recipient user in the corresponding B_r will not receive them, which is even more serious when handling a great number of assets.

T-ODAP mitigates threat 3 through the fifth phase of the protocol (see Figure 3.4), in which the internal state of the recipient ledger is verified after G_r claims that the assets were created. In case the

gateway is malicious and the assets are not created, the transfer suffers a rollback and the asset's state in B_s goes back to pure state.

Besides the aforementioned threats being taken into account and mitigated by T-ODAP, the attacks described in threat 1, 2 and 3 can still be successfully executed during the attack windows - i.e. during the intervals between view publications, since during the latter the attack is not registered and thus can not be proven to have happened. As an example, in the second phase of a T-ODAP protocol instance, an asset is blocked (in a pre-locked state). The G_s communicates that information to G_r and waits for the source ledger's next view publication. Immediately after this, the G_s unlocks the asset, but the View Publication Flow will still occur successfully given that the view contained the asset in its blocked state.

In order to diminish the attack window as much as possible, the view publishing frequency should be high (i.e. k should be low). However, this is a trade-off - highly frequent publications incur high costs for the solution. In future work Section 6.2, other solutions may be employed in order to maintain an equilibrium between the cost and the security of the solution (e.g. device attestation to verify the gateway's code logic [68] or checking the asset's state for a limited period of time, to guarantee that it remains locked).

As mentioned in Section 3.3.3.B, in order to motivate the agents to follow desired actions instead of undesired ones, we attribute punishments to the latter.

The punishment value corresponds to the severity value of the attack attempted multiplied by the quantity of units the gateway tried to steal, i.e.:

$$p = s \times u, \tag{3.3}$$

where *p* corresponds to the punishment, *s* to the attack severity and *u* to the amount of units attempted to be stolen.

With this formula, we can guarantee that the punishment value is always greater or equal (in case the severity value equals 1) than the valuation's value (i.e. what the attacker would gain from stealing). Thus, as seen in Section 3.3.3.B, the gateways are always incentivized to choose the desired actions that benefit the protocol.

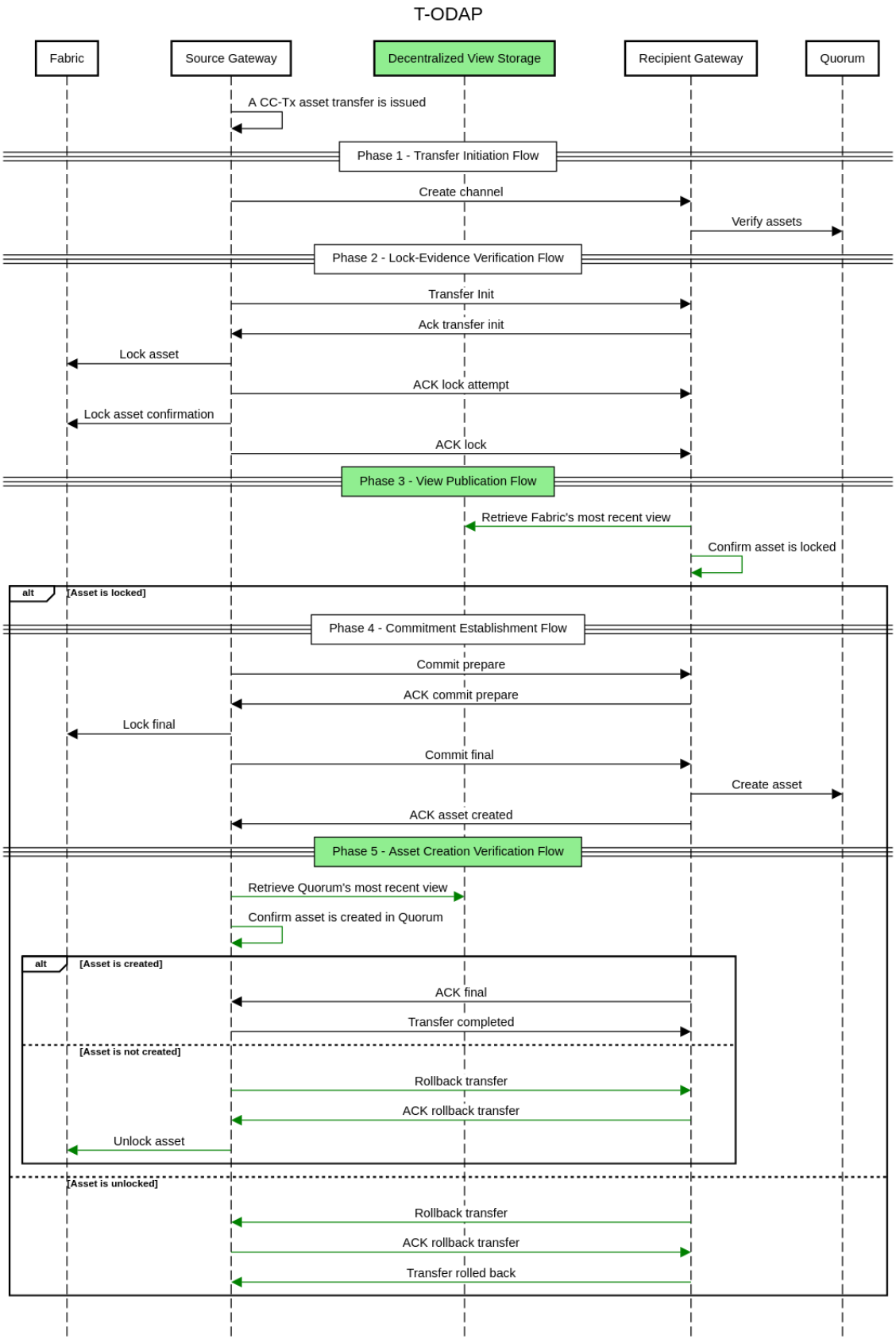

Figure 3.4: T-ODAP Protocol Flow example

4

T-ODAP Implementation

Contents

In this chapter, we present the decisions made in the implementation process of T-ODAP in terms of technology, along with technical details.

4.1 Technologies

This section introduces the decisions made in terms of the technology used for T-ODAP.

Recalling Figure 3.2, there are three main layers that compose our system, starting from the bottom one - the DVS, the Polkadot Connector and the T-ODAP protocol itself.

4.1.1 DVS

As we mentioned prior, the Decentralized View Storage is an immutable bulletin, and we implemented it in the form of a smart contract. We came to this decision by basing our work on [5] and given the fact that, nowadays, most blockchains support smart contracts. Having made the decision of DVS being implemented this way and available as a live, distributed smart contract in a given blockchain network, the next step was deciding the language in which the contract would be implemented, along with deciding the underlying infrastructure where it would be deployed and live.

As the smart contract needed to be accessible externally (i.e. by permissioned blockchains), at first we analyzed several public blockchains as potential underlying infrastructures. In particular, we considered Ethereum. However, we concluded that this solution would have drawbacks since it suffers from scalability issues (Ethereum only supports about 200 transactions per second [3]), consumes a vast amount of energy and has no provable transaction finality - it only guarantees that eventually, at some point in the future, all nodes will agree on the truthfulness of a given set of data (probabilistic finality) [69].

Recently, Ethereum 2.0 (an important update to the current Ethereum) also emerged. This new version of the blockchain will use sharding techniques to improve its throughput, and will also present far lower transaction fees. Its finalization expected latency is around 6-12 minutes [3]. While this is an exciting prospect, Ethereum 2.0 is still in an immature phase (the second phase of the roll-out will happen near the end of 2021).

After analyzing other possible solutions as underlying infrastructures for the DVS, we opted for Polkadot, a scalable and flexible interoperability mechanism - more specifically, a Blockchain of Blockchains (see Section 2.3.2).

Polkadot shares many functionalities with the aforementioned Ethereum 2.0, including the interoperability approach of sharding, which grants both a high scalability. However, Polkadot has an expected latency of 6-60 seconds [3] which is significantly better than the latter. Besides, the BoB provides a hybrid type of consensus (including two protocols, BABE and GRANDPA [69]) which leverages both

provable and probabilistic finality. This way, the system is able to get the benefit of both provable (blocks can never be reversed after some process takes place) and probabilistic finality (ability to always produce new blocks), and also avoid the drawbacks of having just one of the two protocols.

Having decided on Polkadot, we then needed to choose the smart contract language in which we were going to implement the DVS's functionality. Two distinct pallets are currently offered by Polkadot for smart contract development - the Frontier pallet, for the development of EVM-based contracts, and the Contracts pallet, for the development of WebAssembly (WASM) based contracts [70]. The last pallet made some improvements in regards to the first (such as introducing a rent for each contract to justify its existence on-chain) which is why we chose to use it.

This pallet supports any traditional programming language that compiles to WASM code, along with the Domain Specific Language (DSL) Parity's ink! based in Rust [71]. However, as ink! is domain specific, leveraging it increased the efficiency of the software development process (despite the time invested in learning the language), which is why we chose it for the development of our smart contract.

4.1.2 Polkadot Connector

The next challenge was finding a method for connecting the permissioned blockchains to the DVS (i.e. connecting the blockchains to Polkadot).

Our first idea was to build a bridge to Polkadot directly, but this would require manually building several bridges from several permissioned blockchains since, at the time, Polkadot only provided bridges to Ethereum and Bitcoin. Currently, the BoB offers some bridge options in terms of connecting to permissioned environments (such as Tindermint) [72], however they are still limited and building even one bridge manually would not be feasible due to time constraints.

After analyzing other options, we encountered Hyperledger Cactus - a hybrid interoperability mechanism, which is currently live and is already compatible with several permissioned blockchains such as Quorum, Corda, Fabric and Besu [73].

This was ideal since, by using Hyperledger Cactus, we would not have to worry about manually building several bridges for connecting permissioned blockchains to Polkadot. Instead, we just needed to build one connector that would connect them to Polkadot (which is what we did).

4.1.3 T-ODAP

On top of the Polkadot Connector, we had to decide the technology for our trustless version of the Open Asset Digital Protocol - T-ODAP.

This decision was rather easy since the ODAP protocol is currently implemented in Hyperledger Cactus. Therefore, besides implementing the Polkadot connector, we would only have to adapt ODAP's

existing version to have the interaction with the DVS's smart contract in Polkadot (reading and writing from the latter).

4.2 Implementation

This work's implementation comprises Hyperledger Cactus for cross-blockchain transaction logic as well as Polkadot as an underlying infrastructure. It also encompasses one smart contract, with the decentralized view storage's logic, written in Parity's rust-based ink! language [71].

Please note that smart contracts are only now stabilizing on Polkadot. The latter does not natively support them, however some parachains will (see Section 2.3.2).

It is important to note that, although parachains are not yet launched in Polkadot, there is a project called Edgeware [74] which implements the Contracts pallet and is currently in production. Edgeware is essentially a smart-contract blockchain that allows developers to deploy their ink! written smart contracts. It will be integrated as a parachain in Polkadot, so every smart contract deployed in Edgeware will become a part of the Polkadot blockchain.

It is also possible to create a simple Substrate Blockchain, in which a smart contract can be deployed and interact with the Substrate's test network.

4.2.1 Decentralized View Storage Components

As mentioned in Section 4.1, the decentralized view storage (i.e. the smart contract containing its logic) is implemented in ink!.

The storage of the contract contains (see Figure 4.1):

```
#[ink(storage)]
/// Contains the storage of the DVS
pub struct DVS {
    /// Each member is associated with several ids, and each id is associated with a commitment
    commitments_per_member: HashMap<InkAccountId, StorageBox<HashMap<i32, Commitment>>>,
    /// Each member is associated with several ids, and each id is associated with the evaluations of other members for that commitment
    replies_per_member: HashMap<InkAccountId, StorageBox<HashMap<i32, StorageBox<Vec<String>>>>>,
    /// Account ID's which correspond to the committee members of the blockchain
    whitelist: Vec<InkAccountId>,
    /// The current block height of the blockchain
    current_height: u32,
    /// While waiting for a quorum of replies, the timeout is the number of new blocks we can wait for before cancelling and rolling back
    timeout: u32,
}
```

Figure 4.1: Snippet of DVS Storage

- *commitments_per_member* - A HashMap containing account IDs (corresponding to committee members' accounts) as keys and HashMaps as values. The innermost HashMaps comprise integers as keys (each corresponding to a height) and commitments as values. In essence, each committee

member is associated with several heights and each of those heights is associated with a single commitment (or none).

We chose a collection to save this information since we do not *a priori* know the number of heights that will be associated with that particular committee, or how many commitments will exist. The number of committee members might also change. In particular, we chose a HashMap to save this information due to the latter's complexity and due to HashMaps' fast look-ups and insertions.

- *replies_per_member* - As the previous, it is a HashMap containing account IDs (corresponding to committee members' accounts) as keys and HashMaps as values. Here, the innermost HashMaps comprise integers as keys (each corresponding to a height) and Strings as values. In short, each committee member is associated with several heights and, in turn, each of those heights is associated with a String which corresponds to a reply (or none). The latter is the approval or disapproval of a given view.

  As before, a collection was chosen to save this information since we do not *a priori* know the number of heights that will be associated with that particular committee, how many replies will exist or how many committee members will exist (this number might change as time passes). We chose a HashMap to save this information due to the same reasons stated for *commitments_per_member*.

- *whitelist* - A vector containing the committee members of the destination permissioned blockchain. This is necessary in order to verify that certain actions can only be executed by authorized members. A vector was chosen for the whitelist since it is a growable array type, which is necessary because new members can be added to (and removed from) the committee.

- *current_height* - An unsigned integer that corresponds to the permissioned blockchain's current block height. It is unsigned given that the latter can never be negative. The `current_height` is used by the smart contract when checking if the *timeout* has expired.

- *timeout* - An unsigned integer that corresponds to the maximum number of new blocks in the permissioned blockchain we can wait for before an operation is rolled back.

4.2.2 Polkadot Connector Components

The Polkadot Connector provides functionality that enables any permissioned blockchain (as long as it is supported by Hyperledger Cactus) to connect to the Polkadot network and perform monetary transactions to the latter. Besides this, the connector provides methods for these blockchains to deploy and interact with smart contracts in the network.

It is important to note that the aforementioned methods are not specific to T-ODAP or our use case. The Polkadot Connector is used in our work, but is, in its core, a contribution to the development of

Hyperledger Cactus and blockchain interoperability and can be leveraged to many other use cases.

Our connector in Hyperledger Cactus uses three different API's to communicate with the Polkadot network: "@polkadot/api" (which has the base API functionality), "@polkadot/api-contracts" (which contains the API specific for smart contract interaction) and "@polkadot/types" (which encompasses specific Polkadot types).

Prior to diving into the main functions, we present the interfaces that are either fed as input (requests) or returned as output (responses) in those functions:

- *DeployContractInkBytecodeRequest* - A request that encompasses the attributes required to deploy a specific ink! smart contract in Polkadot. These are:
  - *wasm* - A Uint8Array object (a typed array of 8-bit unsigned integer values) which corresponds to a WASM binary file, generated after building the smart contract code;
  - *abi* - An AnyJson object (can be a string, number, boolean or any type of json object) corresponding to the smart contract's Application Binary Interface (ABI), which describes the interfaces that can be used to interact with the contract;
  - *endowment* - A positive number which corresponds to the balance to transfer to the newly created smart contract;
  - *gasLimit* - A positive number which corresponds to the maximum gas the caller is willing to spend when executing the smart contract's constructor;
  - *params* - An optional parameter of type Array $<$Unknown$>$, corresponding to any parameters that possibly need to be supplied to the smart contract's constructor.
- *DeployContractInkBytecodeResponse* - A response returned by the connector which provides information on the success of the smart contract's deployment. It contains only one attribute, *success*, a boolean value with the value "true" in case the smart contract was deployed, and "false" otherwise.
- *ReadStorageRequest* - A request that contains attributes necessary for reading the storage of a smart contract. The latter are:
  - *account* - A string that corresponds to the Polkadot account which wants to perform the read request. This account signs the transaction encoding the read operation;
  - *gasLimit* - A positive number which corresponds to the maximum gas the caller is willing to spend when executing the smart contract's constructor;
  - *read_function* - A string corresponding to the name of the read function for the specific smart contract;

  - *params* - An optional Array <Unknown>, corresponding to any parameters that possibly need to be supplied to the smart contract's read function.

- *ReadStorageResponse* - A response returned by the connector which provides information on the success of the read operation. It encompasses the following attributes:
  - *success* - A boolean with the value "true" in case the smart contract was deployed, and "false" otherwise;
  - *output* - An optional parameter of type AnyJson, which corresponds to the output of the read operation (only existing in case the operation was successful).

- *WriteStorageRequest* - A request that contains attributes necessary for writing in the storage of a smart contract. These are similar to the attributes in a *ReadStorageRequest*:
  - *account* - A string that corresponds to the Polkadot account which wants to perform the write request. This account signs the transaction encoding the read operation;
  - *gasLimit* - A positive number which corresponds to the maximum gas the caller is willing to spend when executing the smart contract's constructor;
  - *write_function* - A string corresponding to the name of the write function for the specific smart contract;
  - *params* - An optional Array <Unknown>, corresponding to any parameters that possibly need to be supplied to the smart contract's write function.

- *WriteStorageResponse* - A response returned by the connector which provides information on the success of the write operation. It contains only one attribute, *success*, a boolean value with the value "true" in case the smart contract was deployed, and "false" otherwise.

Leveraging the aforementioned interfaces, we present the most relevant connector functions for our work:

- *deployContract* - A function that receives a DeployContractInkBytecodeRequest, returns a DeployContractInkBytecodeResponse and whose purpose is to deploy a smart contract given the above-mentioned parameters. The function throws an error in case the smart contract deployment fails. Otherwise, it stores the contract's ABI and code and sets *success* = true.

- *readStorage* - A function that receives a ReadStorageRequest, returns a ReadStorageResponse and whose purpose is to perform a read operation on a smart contract given the attributes of the request. The function throws an error in case the operation fails or, in case the result is a falsy value, returns *success* with the value "false". Otherwise, it returns the attribute *success* with the value "true", along with the retrieved output of the read function.

- *writeStorage* - A function that receives a WriteStorageRequest, returns a WriteStorageResponse and whose purpose is to perform a write operation on a smart contract given the attributes of the request. The function throws an error in case the operation fails or, in case the result is a falsy value, returns *success* with the value ”false”. Otherwise, it returns the attribute *success* with the value ”true”.

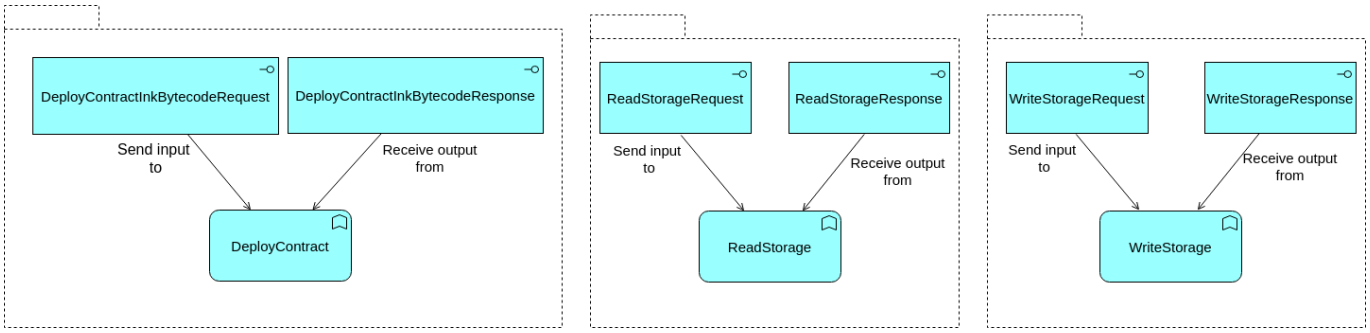

Figure 4.2: T-ODAP’s Functions and Interfaces

Besides these functions, as presented before, the connector presents functionality to send monetary transactions to Polkadot. It also leverages a metrics collection and alerting tool, Prometheus [75] that stores the number of transaction performed.

5

Evaluation

Contents

In this chapter, we evaluate T-ODAP. We begin by discussing the evaluation methodology, the metrics evaluated and the test environment. The evaluation is then performed, being divided into two sections: theoretical and practical evaluation. The first comprises a game theoretical evaluation based on a framework that evaluates the solution's robustness in terms of resilience to attacks. The practical section could not be complete due to the fact that, as mentioned before, the third layer of T-ODAP was not implemented. However, we explain the metrics we planned to evaluate and discuss our expected results. We then present a use case for our solution. Finally, we discuss the results obtained and reinforce the distinguishing aspects of our solution.

5.1 Evaluation Methodology

Our evaluation of T-ODAP is mostly focused on the solution's robustness in face of attacks (i.e. its security).

Please note that, as stated before, T-ODAP was not fully implemented due to circumstances beyond our control which delayed the solution's implementation. More specifically we depended on having a complete implementation of the ODAP protocol in order start our implementation of the third layer of T-ODAP, however ODAP was not fully implemented on the date we thought it would be finished which caused our work to be incomplete.

On this note, the theoretical evaluation (focused on robustness in face of attacks) could proceed with no alterations, tests were performed to the two first layers of the protocol, however the practical evaluation could not be done.

In this context, the main question our evaluation tackles is: How is T-ODAP's robustness characterized in terms of resilience to attacks?

Additionally, although we could not implement the practical evaluation, the questions we had planned to tackle with the latter in order to draw conclusions on the suitability of T-ODAP were the following:

- What is the minimum possible attack window for this solution, i.e. how frequent should the publishing of views in the DVS be?

- What is the latency for each view publication at the maximum throughput, i.e. what is the minimum amount of time required to perform a view publication?

- What is the monetary cost of the DVS solution (cost/byte) i.e. what is the efficiency of the solution?

5.2 Test Environment

Given that the T-ODAP system is composed by several layers (as seen in Figure 3.2), we tested our solution after the implementation of each.

In order to test the DVS, which is implemented in the form of a smart contract (see Section 4.2.1), we wrote tests in the ink! language and ran these tests using rust's toolchain "nightly" [76], version 1.55.0.

The latter included 12 general cases and also edge cases. The tests included the following cases:

- A view being published since it met all the criteria;
- A view not having sufficient approvals and thus not being published;
- A view not being published due to the publishing entity not being authorized (i.e. not being a committee member);
- A view not being published due to an incorrect rolling hash;
- A view publishing that exceeds the timeout and thus is rolled back (and not published).

Next, following the implementation of the Polkadot connector, we tested it. The latter was done on top of the hybrid interoperability solution Hyperledger Cactus. As mentioned before, Cactus achieves interoperability between several blockchains, with a focus on permissioned ones.

In order to be able to properly test our Polkadot connector, we needed to create a Polkadot test environment containing a working node for the connector to connect to. To do this, we leveraged Docker version 20.10.2 to create a container in which we installed the necessary software - some Ubuntu packages, Rust and the nightly toolchain, WASM-related packages and, finally, the Polkadot node itself.

In order to facilitate the use of the container in the tests, we also created a class that programmatically obtained information about the docker container running the Polkadot Node, as well as starting and stopping the latter. The aforementioned class, as well as the tests, were written using Typescript.

We have five main test files in which we performed distinct tests to the connector:

- *constructor-instantiation* - We tested the good functioning of our connector and its connection to the Polkadot test ledger (i.e. the Polkadot node);
- *deploy-ink-contract* - We tested the deployment of a smart contract to the Polkadot node. To test this, we used the smart contract containing the DVS's logic;
- *read-ink-contract* - We tested a read operation from the previously deployed DVS smart contract;
- *write-ink-contract* - We tested a write operation to the previously deployed DVS smart contract.

The tests passed successfully, indicating the correct behavior of our work.

In the next section we will focus on the evaluation of the T-ODAP system as a whole.

5.3 Theoretical Evaluation

As mentioned before, this component of the evaluation of T-ODAP comprises a theoretical game-theory based approach to evaluate the solution in terms of robustness to attacks.

In order to do this, we leverage the game theoretical framework studied in Section 2.5.

The latter evaluates a blockchain protocol's robustness by first identifying the players involved, the actions they can perform (tied with specific utilities) and the game or games that better represent that protocol.

In T-ODAP, we have two players - the source gateway G_s, and the recipient gateway G_r. These are considered to be rational players, meaning that they both always desire to maximize their own utility (see Section 2.4).

Based on the protocol flow described in Section 3.4.2, we divided the protocol into three different games, the first (A) corresponding to Phases 1, 2 and 3, the second (B) corresponding to the scenario where the asset is locked and the third (C) to the remaining scenario.

In each game, the order of the actions performed matters. In this context, if any player deviates the protocol, the game goes back to the initial state, with a null outcome (0) for each (i.e. (0,0), where the first position corresponds to G_s and the second one to G_r). The initial state corresponds to the state before the asset transfer. If they do follow each step correctly, they receive a positive utility of (1,1). If a player is harmed by another player's action, the harmed player receives a negative utility of -1, similarly to authors in [4]. The values of 1 and -1 were chosen by convention.

In game A, represented in Figure 5.1, the following actions can be executed:

- G_s can either create a channel (Y), which leads to node 2, or not do it (N) with outcome (0,0);
- G_r can either verify the assets (Y) which leads to node 3, or not do it (N) with outcome (0,0);
- G_s can either initiate the transfer (Y) which leads to node 4, or not do it (N) with outcome (0,0);
- G_r can either respond to the transfer initiation with an ACK (Y) which leads to node 5, or not do it (N) with outcome (0,0);
- G_s can either lock the asset and inform G_s of this with an ACK (Y) which leads to node 7, or not do it (N) which leads to node 6;

- From node 7:
  - G_s can either confirm the asset is indeed locked and confirm this to G_r with an ACK (Y) which leads to node 7.1, or not do it (N) with outcome (0,0);
  - G_r can either retrieve B_s's most recent view and observe that the asset is locked (Y) which leads to outcome (1,1), or not retrieve the view and verify the asset's state (N) with outcome (0,0);
- From node 6:
  - G_s can either maliciously inform G_r that the asset is locked and confirm this with an ACK (Y) which leads to node 6.1, or not do it (N) with outcome (0,0);
  - G_r can either retrieve B_s's most recent view and observe that the asset is unlocked (Y) which leads to outcome (0,0), or not retrieve the view and verify the asset's state (N) with the same outcome (0,0);

When it comes to the last action described for game A, in case the asset is unlocked the outcome is 0 for both players since G_s deviated from the protocol by not locking the asset. In this context, we do not consider G_r to be harmed since the protocol contains methods to avoid the successful execution of attacks.

In game B, we have the following possible actions, as illustrated in Figure 5.2:

- G_s can either prepare the commit (Y) which leads to node 2, or not do it (N) with outcome (0,0);
- G_r can either inform G_s that it received the information with an ACK (Y) which leads to node 3, or not do it (N) with outcome (0,0);
- G_s can either perform the final asset lock (Y) which leads to node 4, or not do it (N) with outcome (0,0);
- G_s can either commit the asset (Y) which leads to node 5, or not do it (N) with outcome (0,0);
- G_r can either create the corresponding asset and inform G_s of this (Y) which leads to node 7, or not do it (N) which leads to node 6;
- From node 7:
  - G_s can either retrieve B_r's most recent view and observe that the asset is created (Y) which leads to node 7.1, or not retrieve the view and verify the asset's state (N) with outcome (0,0);
  - G_r can acknowledge to G_s that it created the asset on its side (Y) which leads to node 7.2, or not do it (N) with outcome (0,0);

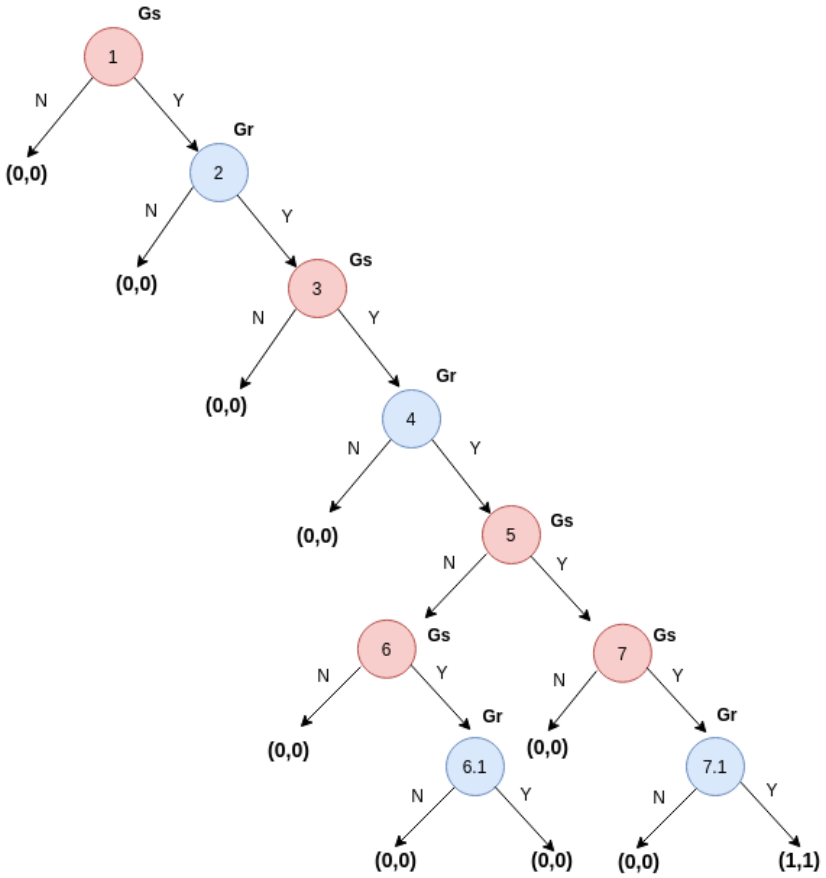

Figure 5.1: Game A - Diagram

  - G_s can inform G_r that the transfer is complete (Y), which leads to outcome (1,1), or not do it (N) with outcome (0,0);

- From node 6:

  - G_s can either retrieve B_r's most recent view and observe that the asset is not created (Y) which leads to node 6.1, or not retrieve the view and verify the asset's state (N) with outcome (0,0);
  - G_s can inform G_r that it will rollback the asset transfer (Y) which leads to node 6.2, or not do it (N) with outcome (0,0);
  - G_r can acknowledge to G_s that it received information about the rollback (Y), which leads to node 6.3, or not do it (N) with outcome (0,0);
  - G_s can instruct B_s to unlock the asset (Y), which leads to outcome (0,-1), or not do it (N) which leads to outcome (0,0);

Finally we have game C, illustrated in Figure 5.3, with the following possible actions:

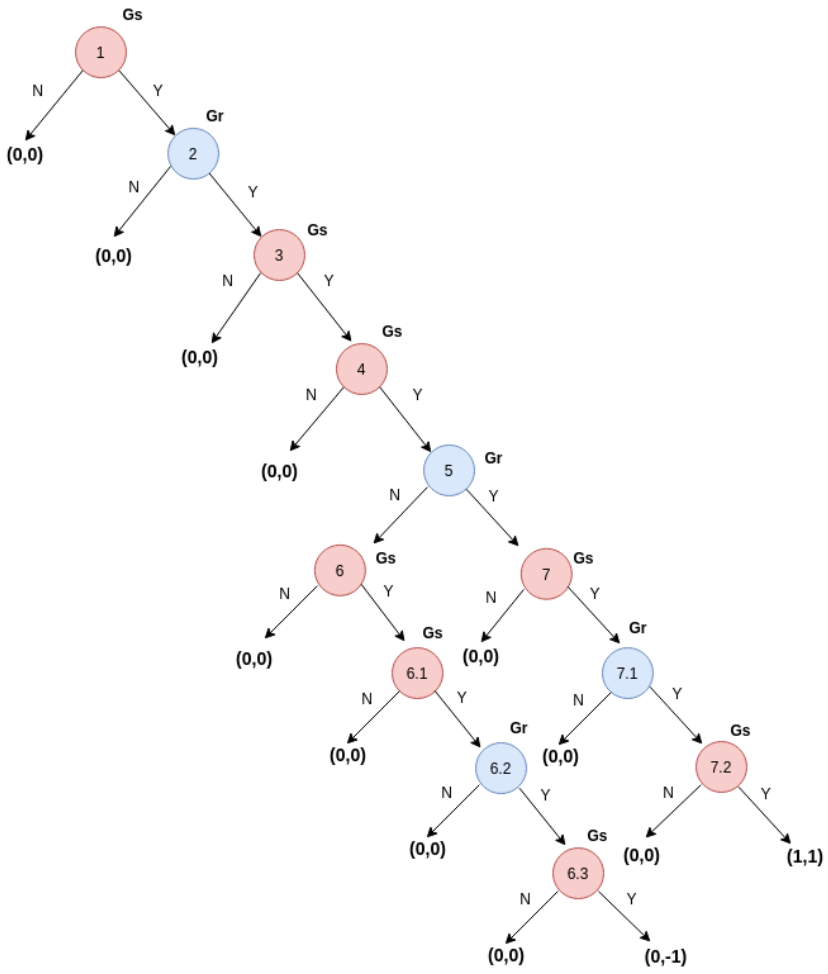

Figure 5.2: Game B - Diagram

- G_r can inform G_r that it will rollback the asset transfer (Y), which leads to node 2, or not do it (N) with outcome (0,0);
- G_s can acknowledge to G_s that it received information about the rollback (Y), which leads to node 3, or not do it (N) with outcome (0,0);
- G_r can rollback the transfer (Y) with outcome (-1,0), or not do it (N) with outcome (0,0);

Note that both cases of rolling back the transfer to avoid the successful execution of an attack - either a double spend attack or a denial-of-service attack (see Section 3.5) - provide a negative outcome for the malicious gateway - (0,-1) in Game B and (-1,0) in Game C - given that their attempted attack has failed. For the honest gateway, the outcome is 0 since the asset transfer goes back to the initial state.

As described in Section 2.5, it is easier to analyze a protocol by dividing it into several independent games. In our case, each game is independent. However, it is important to note that a given instance of T-ODAP will only encompass two of the three games described - depending on the outcome of Game A,

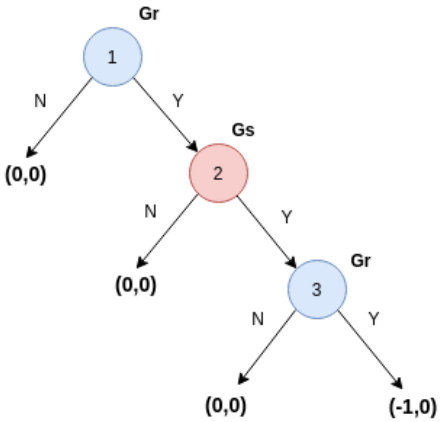

Figure 5.3: Game C - Diagram

either Game B **or** Game C will be triggered. More specifically, Game B will be triggered with an outcome of (1,1) in Game A, and Game B will be triggered with an outcome of (0,0) in Game A.

For Game A, with our two players, we have that the strategy profile that corresponds to following the protocol is σ_{A1} = ({Y,Y,Y,Y}, {Y,Y,Y}), i.e. in case G_s correctly locks the asset, while σ_{A2} = ({Y,Y,N,Y}, {Y,Y,Y}) is followed when G_s deviates the protocol by not locking the asset, leading to outcome (0,0).

In case of Game B with the same players, the strategy profile that corresponds to following the protocol is σ_{B1} = ({Y,Y,Y,Y,Y}, {Y,Y,Y}), i.e. in case G_r correctly creates the corresponding asset, while σ_{B2} = ({Y,Y,Y,Y,Y,Y}, {Y,N,Y}) corresponds to G_r not creating the asset, leading to outcome (0,-1).

Finally, in case of Game C (which is only triggered if the asset is not locked in B_s), the strategy profile where both players follow the protocol corresponds to σ_C = ({Y}, {Y,Y}), with outcome (-1,0).

As mentioned before, an instance of T-ODAP can be composed by Game A and Game B forming mechanism AB or by Game A and Game C, forming mechanism AC. In the latter case, the outcome will never be the best outcome possible for any of the players - it will either be 0 or -1.

AB is formed by composing both A and B games. The latter is represented as

$$(A \circ B, \{\sigma_A, \sigma_B\}) \tag{5.1}$$

where σ_A and σ_B comprise all of the strategy profiles in games A an B, respectively.

Let us analyze the mechanism AB.

As mentioned in Section 2.5, a system is t-immune if the utility of players is not inferior if at most *t* players deviate the protocol. This is not the case for the aforementioned mechanism, since players go back to the initial state with a null outcome if any player deviates the protocol. This means that AB is not t-immune, which means it is not (k,t)-robust.

In order to explore the properties of the mechanism, we need to explore the properties of each game. In Game A, the strategy profile σ_{A1} is the only one with a maximal outcome of (1,1); all the other strategy profiles lead to a null outcome. This means that the mechanism (A, σ_{A1}) is strongly resilient.

It also means that (A, σ_{A1}) is practical given that all strategies except for σ_{A1} are weakly-dominated, since they bring the outcome to 0 which is less than the maximal outcome. Having this in mind, we have that σ_{A1} is a Nash equilibrium of the game A after deletion of weakly-dominated strategies, making (A, σ_{A1}) practical.

Being strongly resilient and practical, (A, σ_{A1}) is considered to be optimal resilient.

This mechanism also presents an interesting property - for any honest player, the worst possible payoff it receives is the initial state. This is true since an honest player can either receive outcome 1, if the other player is honest as well, or receive 0 (corresponding to the initial state) if the other player deviates the protocol, meaning that the mechanism is also weak-immune.

Now looking at Game B, we have that the strategy profile σ_{B1} provides a maximal outcome of (1,1). Similarly to Game A, all other strategy profiles lead to an outcome of 0, which means that the mechanism (B, σ_{B1}) is strongly resilient.

As all strategies except σ_{B1} lead to an outcome that is less than the maximal outcome, this again means they are weakly dominated. This way, σ_{B1} is a Nash equilibrium after deletion of the other strategies, making (B, σ_{B1}) practical as (A, σ_{A1}).

(B, σ_{B1}) is also optimal resilient due to being strongly resilient and practical.

Game B also presents the property of weak-immunity for the same reasons as Game A. Similarly, in the worst case an honest player receives an outcome of 0 if the other player deviates from the protocol, and in the best case both receive 1 if they follow the latter.

Since both mechanisms are optimal resilient and weak immune, by applying the Theorems 2, 3 and 4 of [4] which ensure the invariance of properties once the composition operator is applied, we get that the resulting mechanism AB is both optimal resilient and weak immune, which makes it (k,t)-weak-robust.

Similarly to AB, AC is formed by composing both A and C games. The latter is represented as

$$(A \circ C, \{\sigma_A, \sigma_C\}) \tag{5.2}$$

where σ_A and σ_C comprise all of the strategy profiles in games A an C, respectively.

Let us analyze this mechanism, again by exploring the properties of each of the games. In this case, we already analyzed Game A, leaving us with game C to explore.

In the latter, the strategy profile is σ_C. Contrarily to before, this strategy leads to an outcome of (-1,0), which is not the maximal outcome. In fact, the null outcome (0,0) is higher. This means (C, σ_C) is not

strongly resilient.

The mechanism is also not practical, given that σ_C is not a Nash Equilibrium.

On another hand, (C, σ_C) is weakly dominated, since an honest player never receives less than the outcome of the initial state. If the players reached Game C, it is because G_s is malicious and did not lock the asset it was supposed to transfer to G_r. Therefore, if G_r chooses to rollback the transfer (the last action in the game), G_s gains a negative utility since its attempt of a double spend attack was unsuccessful. G_r on another hand can either be honest or malicious. If it is honest and follows σ_C, the outcome is equal to the outcome of the initial state (0).

Since Game C is weak-immune but not optimal resilient, the latter is not (k,t)-weak-robust. This means that the composition mechanism AC is also not (k,t)-weak-robust [4].

Game C is only triggered in case G_s misbehaves in Game A. If we were to analyze (B, σ_{B2}), we would obtain the same results due to the strategy leading to an outcome of (0,-1). The T-ODAP instance represented by mechanism AC does not correspond to a scenario in which both players follow the protocol, hence it makes sense that this specific mechanism is not (k,t)-weak-robust.

Indeed, out of both mechanisms AB and AC, AB is the one relevant to decide if T-ODAP is (k,t)-weak-robust given the fact that, as mentioned before, out of the two it is the only one that presents a possibility of players following the protocol from the beginning to the end.

Given that AB is (k,t)-weak-robust and given that it is the only relevant mechanism for the characterization of T-ODAP's robustness due to the aforementioned reasons, we can conclude that T-ODAP is (k,t)-weak-robust.

5.4 Practical Evaluation

As mentioned before, the practical evaluation could not be completed due to our implementation of the third layer having been delayed due to external circumstances.

Nonetheless, we present the metrics which we were going to evaluate in the practical evaluation if we had had the time - throughput, latency, cost and the publishing frequency of T-ODAP.

5.4.1 Throughput

This metric is important to our work, given that the solution must be able to deal with several cross-chain transaction requests for asset transfers without causing great harm in latency and performance. We would evaluate T-ODAP's scalability by testing it against:

1. Throughput - T-ODAP's transaction throughput would be evaluated. We would measure the system's response against different asset transfer request rates, to study what is its average and maximum throughput.

2. T-ODAP vs. ODAP - We would also evaluate our baseline's (ODAP) current throughput (specifically, its average and maximum throughput) against T-ODAP's, in order to determine how much slower is T-ODAP in relation to ODAP. As T-ODAP is built on top of ODAP and adds extra steps (and thus, more complexity), it is certain that our solution will be somewhat slower than the baseline. As mentioned before, we understand that there is a trade-off between performance and security yet we choose to give more weight the trustlessness and security of our solution.

5.4.2 Latency

The evaluation of the solution's latency reflects how responsive it can be. This metric is also extremely relevant to our work given that we want it to be as responsive as possible, without compromising the security of our solution.

In this context, we would test T-ODAP against distinct workloads of asset transfer request rates. The latency of ODAP would also be evaluated against our solution's latency.

Similarly to before, T-ODAP is expected to have higher latency when compare to its baseline, ODAP, due to the reasons mentioned in the previous subsection. This is not desirable, however it is something we are willing to accept - we believe that the benefits of the trustlessness of our solution outweigh the cons of having a higher latency and lower average and maximum throughput.

5.4.3 Cost

In this context, the cost of the solution corresponds to the monetary cost of running an instance of the protocol, in terms of transaction fees - T-ODAP encompasses several transactions, some of which entail monetary cost.

Similarly to previous solutions [14] we would measure these costs in EUR as the amount of transaction fees required to the correct functioning of the protocol. The costs would be calculated using the conversion rates at the time of the experimental evaluation.

In order to evaluate this parameter, we could run several instances of the protocol and save the costs entailed in each instance. Then, we could perform an average and observe which were the minimum and maximum values. We would also evaluate the cost of ODAP against the cost of T-ODAP.

Although we can not predict the specific values, similarly to the throughput and latency, it is expected that the cost of T-ODAP would be slightly higher given that there are extra transactions involved (such as reading or publishing a view in the DVS).

5.4.4 DVS Publishing Frequency

This is a key metric for our work, given that the DVS Publishing Frequency directly affects the security of our solution. If a given blockchain publishes views with large time intervals between them, it is possible that, when running an instance of the protocol, a certain action that has already taken place doesn't show up in the retrieved view (for example, G_s locking the asset). This can cause the asset transfer to rollback when that did not need to happen. The opposite can also occur - an attacker may unlock the asset between the publication of views, which causes the retrieved view to contain the state of the asset as locked when it is not indeed locked, which can lead to a double spend attack (see section 3.5).

On another hand, publications with very short time intervals are much better in terms of security however cause a great overhead in the DVS and thus add more latency to the solution, which is not ideal.

In this context, we would test the solution against different publishing frequencies and observe its behavior in terms of robustness to attacks, trying to understand what is the ideal value or range of values for *k* of publishing frequency in terms of a having a small attack window, as well as not having a great loss in performance at the same time.

5.5 Use Case - Trustless Cross-Border Transfers between Banks

It is now a well-known fact that blockchains can have a much larger applicability than just cryptocurrency - they can be leveraged for applications that include securing medical data, NFT marketplaces, music royalties tracking or cross-border payments, the focus of our use case.

Nowadays, cross-border payments are classified by many as the most inconvenient form of payment [77], mostly due to the extremely high transaction fees - in the first quarter of 2021, this value was at 6.38% [78] - but also due to the worrisome lack of transparency and the high latency (a cross-border transfer takes, on average, around 2-3 days, while a domestic transfer usually takes seconds).

This is a real problem, especially since the majority of these cross-border transactions correspond to migrants sending money to their families back at home, most of which have a high need for the latter [77].

Many of these issues derive from the fact that a cross-border transfer involves a large number of intermediaries (in the form of banks that process the transaction) until it reaches its destination bank. This makes the whole process less secure; the more intermediaries there are, the a higher the risk of fraud and more points of failure exist. Additionally, the more intermediary banks, the higher the fees.

Blockchain-powered solutions have been appearing as a way to mitigate this problem - the technology offers decentralization which results in better security, much faster transactions and lower fees [77]. Ripple is an example of a blockchain-based solution that states to offer cross-border asset transfers in real-time (the CEO of the company claims that a transaction takes on average 4-7 seconds [79]).

These solutions are not yet being used worldwide given that regulations revolving blockchain-powered cross-border transactions are not yet clear, and that companies still have some hesitancy around the subject - more banks have to join the network first in order to be seen as a viable option.

T-ODAP can be leveraged to perform trustless cross-border asset transfers between banks.

With the use of T-ODAP's Decentralized View Storage, besides being able to remove all intermediaries that exist in a regular, non-blockchain based transfer (which causes the process to be much faster and cheaper), the protocol also eliminates the necessity of banks trusting each other (which leads to a more secure mechanism, given that they are much less susceptible to attacks).

In this context, each bank is associated with a T-ODAP gateway, which can perform this type of asset transfers in a trustless manner. This can be seen in T-ODAP's architecture Figure 3.3 - in this context, each bank would save their data in each of the ledgers (source or recipient), being that one bank would be the asset sender and the other one the asset receiver. Here, the end-user corresponds for example to a person which desires to transfer his/her money from their account to a relative's account, which resides in a different country (and whose account is registered in a different bank). This entity is the one that initiates the protocol.

Although we did not implement the use case, it might be relevant to note that we planned to implement it having Hyperledger Fabric and Hyperledger Besu as the blockchains chosen for this task, since they are well-known and documented enterprise-friendly chains which support smart contracts and since they are already compatible with Hyperledger Cactus. We also planned to implement the use case in the form of smart contracts, using NodeJS to program Fabric's smart contract and Solidity to implement Besu's smart contract given that the latter is the most active and maintained language for smart contracts development in Ethereum.

5.6 Discussion

We conclude that our system has an adequate robustness level in terms of rational deviations and immunity to byzantine behavior - more specifically, it is (k,t)-weak-robust, which is indeed a less strong definition than a (k,t)-robust system, yet it is still a great achievement given that, as mentioned before, it presents the robustness level presented by very popular protocols in the field including HTLC-based payment schemes, side-chain protocols and cross-chain swap protocols [4]. Additionally, the tests performed to our solution's implementation (both to the smart contract and to the Polkadot connector) demonstrate that the solution (only including two of T-ODAP's layers) is functioning as intended.

In terms of the practical evaluation, although the latter could not be finished, we presented the metrics we planned on evaluating the system with - throughput, latency, cost and publication frequency - and

also presented our predictions in terms of the evaluation's results for each metric. These results show that, given the complexity of T-ODAP, in relation to ODAP, the three more performance-related metrics (throughput, latency and cost) are expected to suffer.

The latter makes sense given that, in general, a more secure mechanism which does not require trust in its participants tends to have a higher complexity when comparing with other systems which are centralized. These added steps are necessary for the correct functioning of the system, and usually cause the system to be less performant. We understand that this is a limitation of our system, yet our focus is in the trustlessness of T-ODAP.

Another limitation of our work is the fact that it is not fully implemented. The implementation of the third layer of T-ODAP, as well as a complete practical evaluation, is intended as future work. More limitations to our system are described in section 6.2.

Despite the incomplete implementation and its performance limitations, among the related work, to the best of our knowledge T-ODAP is one of the first blockchain interoperability solutions with focus on permissioned blockchains to be completely trustless. Indeed, the related work studied in this thesis contained several interesting solutions - most focused on permissionless chains but some focused on permissioned chains (such as ODAP). However, none tackled all the points that T-ODAP does (for more information on the related work, refer to section 2.10).

6

Conclusions

Contents

Blockchain interoperability is essential for mass adoption. This applies to both permissioned and permissionless blockchains, however we focused on the first due to the small amount of solutions that currently exists. Some projects have been studying and presenting ways to achieve this, however most require a centralized trusted third party, which may not be completely secure. By trusting individual entities, these systems are vulnerable to attacks such as incorrect state sharing by malicious members of permissioned chains or by a malicious centralized third-party. Therefore, there is an opportunity for a new, trustless solution which makes the process more secure. The latter ensures that trust is moved from one entity to a protocol; this provides a higher level of security given that the protocol itself is secure.

We explored and analyzed several state-of-the-art solutions, including XCLAIM, CrowdPatching, ODAP and Public Bulletin. The aforementioned studies comprise highly valuable aspects which can be applied to a solution for permissioned blockchain interoperability. This includes mechanisms based on game theory that incentivize the participants to follow the established protocol, a mechanism to publish internal state proofs and a mechanism to perform unilateral asset transfers between gateways. All were extremely valuable for the construction of the solution, however none of them encompasses all the desired characteristics for our solution - e.g. XCLAIM uses possibly insecure chain relays and ODAP requires gateways to trust each other which can be insecure.

We presented T-ODAP, a multi-layered secure and trustless system leveraging a DVS to publish internal state proofs, a Polkadot Connector to interact with the latter and a trustless adaptation of the ODAP protocol. T-ODAP has the goal to arise as an alternative to the centralized interoperability solutions currently offered to permissioned blockchains, providing stronger levels of security in relation to other protocols such as ODAP due to being trustless.

Theoretically, we evaluated the full system's robustness in face of attacks and concluded that the system is (k,t)-weak-robust, similarly to popular mechanisms such as HTLC-based payment schemes. We performed tests to the implemented layers of our solution, which were successful. The latter were realized through Hyperledger Cactus, which enables blockchain and smart contract testing. Unfortunately, we were not able to perform an experimental evaluation given the fact that the third layer of T-ODAP was not implemented. However, we presented information about the tests we did to the first two layers of our solution, which indicated the correct functioning of the system. Finally, we presented the metrics we would have evaluated if we had had the opportunity, as well as expressing our predictions for the results to expect, relatively to ODAP, which is our work's baseline.

Our solution contributes to the development of permissioned blockchain interoperability, which in turn will hopefully contribute to the widespread adoption of permissioned blockchains in enterprises, affecting the latter and society as a whole.

6.1 Contributions

We were able to deliver several contributions to the scientific community and the open-source community, that have value not only together but also individually.

As primary contributions, we delivered:

1. A Decentralized View Storage built as a Polkadot smart contract, which allows for trustless state sharing between opaque blockchains, that can be leveraged for multiple use cases;
2. A public Polkadot Connector implemented in Cactus, capable of connecting several permissioned blockchains to Polkadot. Besides implementing the connector, we contributed to the open-source community of Hyperledger;
3. A theoretical model of a trustless adaptation of the ODAP protocol, T-ODAP, that leverages the DVS (and the connector in order to interact with the latter);
4. A game theory based analysis that demonstrates that T-ODAP is (k,t)-weak-robust.

Note that the implementation of the Polkadot connector and the DVS smart contract can be found in https://github.com/hyperledger/cactus/pull/1490.

6.2 Limitations and Future Work

T-ODAP has some limitations, the first being that it is more costly than ODAP. This makes sense since our solution adds steps and more complexity to the latter, as well as several transactions in blockchains, which causes this higher cost. However, as stated before, the biggest focus of T-ODAP is in a trustless and secure solution, so the higher cost comes as a trade-off.

As mentioned before, the third layer of T-ODAP and, consequently, the use case mentioned in Section 5.5, were not implemented. The implementation was not possible on time due to circumstances outside of our control; however, the layer and corresponding use case are ready for implementation in future work. This work can be done through Hyperledger Cactus, which has open-source code, where ODAP is implemented as well. The adaptation consists on connecting ODAP to the DVS through the Polkadot Connector, and then adjusting the protocol to verify the asset's state before and after an asset transfer (as described in Section 3.4.2). The use case can be built on top of T-ODAP, leveraging the Hyperledger Fabric and Hyperledger Besu blockchains as Source and Recipient ledgers, such as described in Section 5.5.

Additionally, in future work, other features may be added to T-ODAP such as the support of slashing to punish participants in case they deviate the protocol. This can correspond, for example, to the use of a collateral (as in XCLAIM [14]), which is removed in case the participants misbehave. This mechanism

involves, however, some challenges: how to assure that an internal state proof is actually invalid, in order to punish a participant fairly? The latter is hard to achieve due to the opaqueness of permissioned blockchains.

The Decentralized View Storage may also support more gateways simultaneously, instead of only two. This leads to much more possible synergies between different blockchains, yet it also entails much more complex logic and a broader array of attacks. Moreover, the assumption described in Section 3.2 which states that our solution assumes all views published in the DVS are valid can possibly be removed in future work. This means accepting the fact that auditor nodes can be malicious or assume that there are no auditor nodes at all, and present a solution for this challenge.

It is also interesting to leverage a crash-recovery mechanism for the T-ODAP gateways, given that one of them can crash in the middle of an asset transfer and it is not desirable to have to rollback that transfer every time this happens. [6] presents a first approach to this problem, however it is not implemented yet.

An additional feature for future work can be the attestation of the smart contract code running on each gateway, or having nodes checking the smart contracts and guaranteeing they do not change. This is due to the fact that the contracts may contain malicious code that tries to unlock an asset right after locking it, for example. Finally, it can be interesting to see study if it is possible to decentralize the solution further, and see the limitations of the latter.

Bibliography

[1] Z. Liu, N. C. Luong, W. Wang, D. Niyato, P. Wang, S. Member, Y.-C. Liang, and D. I. Kim, "A survey on applications of game theory in blockchain," *arXiv*, 2019.

[2] E. Androulaki, A. Barger, V. Bortnikov, S. Muralidharan, C. Cachin, K. Christidis, A. D. Caro, D. Enyeart, C. Murthy, C. Ferris, G. Laventman, Y. Manevich, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolić, S. W. Cocco, and J. Yellick, "Hyperledger fabric: A distributed operating system for permissioned blockchains," *Proceedings of the 13th EuroSys Conference, EuroSys 2018*, vol. 2018-January, 4 2018. [Online]. Available: https://doi.org/10.1145/3190508.3190538

[3] R. Belchior, A. Vasconcelos, S. Guerreiro, and M. Correia, "A survey on blockchain interoperability: Past, present, and future trends," *ACM Comput. Surv.*, vol. 54, no. 8, Oct. 2021. [Online]. Available: https://doi.org/10.1145/3471140

[4] P. Zappalà, M. Belotti, M. Potop-Butucaru, and S. Secci, "Game Theoretical Framework for Analyzing Blockchains Robustness," in *35th International Symposium on Distributed Computing (DISC 2021)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), S. Gilbert, Ed., vol. 209. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, pp. 42:1–42:18. [Online]. Available: https://drops.dagstuhl.de/opus/volltexte/2021/14844

[5] E. Abebe, Y. Hu, A. Irvin, D. Karunamoorthy, V. Pandit, V. Ramakrishna, and J. Yu, "Verifiable observation of permissioned ledgers," *2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pp. 1–9, 2021.

[6] R. Belchior, A. Vasconcelos, M. Correia, and T. Hardjono, "Hermes: Fault-tolerant middleware for blockchain interoperability," Mar 2021. [Online]. Available: https://www.techrxiv.org/articles/preprint/HERMES_Fault-Tolerant_Middleware_for_Blockchain_Interoperability/14120291/1

[7] S. Underwood, "Blockchain beyond bitcoin," *Communications of the ACM*, vol. 59, pp. 15–17, 10 2016. [Online]. Available: https://dl.acm.org/doi/10.1145/2994581

[8] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” *Cryptography Mailing list at https://metzdowd.com*, 03 2009.

[9] M. Mettler, “Blockchain technology in healthcare: The revolution starts here.” Institute of Electrical and Electronics Engineers Inc., 11 2016.

[10] “On public and private blockchains — ethereum foundation blog.” [Online]. Available: https://blog.ethereum.org/2015/08/07/on-public-and-private-blockchains/

[11] R. Belchior, B. Putz, G. Pernul, M. Correia, A. Vasconcelos, and S. Guerreiro, “Ssibac: Self-sovereign identity based access control,” in *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. IEEE, pp. 1935–1943.

[12] “Cryptokitties — collect and breed digital cats!” [Online]. Available: https://www.cryptokitties.co/

[13] T. Hardjono, A. Lipton, and A. Pentland, “Toward an interoperability architecture for blockchain autonomous systems,” 2019. [Online]. Available: http://www.ieee.org/publications_standards/publications/rights/index.html

[14] A. Zamyatin, D. Harz, J. Lind, P. Panayiotou, A. Gervais, and W. Knottenbelt, “Xclaim: Trustless, interoperable, cryptocurrency-backed assets,” in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019, pp. 193–210.

[15] “Polkadot: Vision for a heterogeneous multi-chain framework.” [Online]. Available: https://github.com/ethereum/wiki/wiki/Chain-Fibers-Redux

[16] “hyperledger/cactus: Hyperledger cactus is a new approach to the blockchain interoperability problem.” [Online]. Available: https://github.com/hyperledger/cactus

[17] P. Taylor, T. Dargahi, A. Dehghantanha, R. Parizi, and K.-K. R. Choo, “A systematic literature review of blockchain cyber security,” *Digital Communications and Networks*, vol. 6, 02 2019.

[18] R. Belchior, S. Guerreiro, A. Vasconcelos, and M. P. Correia, “A survey on business process view integration,” *ArXiv*, vol. abs/2011.14465, 2020.

[19] “Quorum whitepaper.” [Online]. Available: https://www.blocksg.com/single-post/2017/12/27/Quorum-Whitepaper

[20] “R3 — dlt blockchain software development company.” [Online]. Available: https://www.r3.com/

[21] “Tendermint.” [Online]. Available: https://tendermint.com/

[22] V. Buterin, “Ethereum whitepaper — ethereum.org.” [Online]. Available: https://ethereum.org/en/whitepaper/

[23] J. R. Douceur, “The sybil attack,” in *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, ser. IPTPS ’01. Berlin, Heidelberg: Springer-Verlag, 2002, p. 251–260.

[24] S. Li, M. Yu, C.-S. Yang, S. Avestimehr, S. Kannan, and P. Viswanath, “Polyshard: Coded sharding achieves linearly scaling efficiency and security simultaneously,” 06 2020, pp. 203–208. [Online]. Available: http://arxiv.org/abs/1809.10361

[25] W. Xin, T. Zhang, C. Hu, C. Tang, C. Liu, and Z. Chen, “On scaling and accelerating decentralized private blockchains.” Institute of Electrical and Electronics Engineers Inc., 7 2017, pp. 267–271.

[26] “Small business retail — visa.” [Online]. Available: https://usa.visa.com/run-your-business/small-business-tools/retail.html

[27] J. Teutsch, S. Jain, and P. Saxena, “When cryptocurrencies mine their own business,” in *Financial Cryptography*, 2016.

[28] J. A. Kroll, I. C. Davey, and E. Felten, “The economics of bitcoin mining, or bitcoin in the presence of adversaries,” 2013. [Online]. Available: https://asset-pdf.scinapse.io/prod/2188530018/2188530018.pdf

[29] “Blockchain solutions implementation: How much does it cost in 2020?” [Online]. Available: https://www.tpptechnology.com/blog/blockchain-solutions-implementation-how-much-does-it-cost-in-2020/

[30] P. Frauenthaler, M. Borkowski, and S. Schulte, “A framework for blockchain interoperability and runtime selection,” *ArXiv*, vol. abs/1905.07014, 2019.

[31] “Bitcoin mining now consuming more electricity than 159 countries including ireland most countries in africa.” [Online]. Available: https://powercompare.co.uk/bitcoin/

[32] P. Wegner, “Interoperability,” *ACM Computing Surveys*, vol. 28, pp. 285–287, 3 1996.

[33] “National interoperability framework observatory.” [Online]. Available: https://joinup.ec.europa.eu/collection/nifo-national-interoperability-framework-observatory

[34] H. Vo, Z. Wang, D. Karunamoorthy, J. Wagner, E. Abebe, and M. Mohania, “Internet of blockchains: Techniques and challenges ahead,” 07 2018, pp. 1574–1581.

[35] P. Frauenthaler, M. Borkowski, and S. Schulte, “A framework for blockchain interoperability and runtime selection,” *ArXiv*, 2019.

[36] A. Zamyatin, M. Al-Bassam, D. Zindros, E. Kokoris-Kogias, P. A. Moreno-Sanchez, A. Kiayias, and W. J. Knottenbelt, “Sok: Communication across distributed ledgers,” *IACR Cryptol. ePrint Arch.*, vol. 2019, p. 1128, 2019.

[37] M. Borkowski, C. Ritzer, D. Mcdonald, and S. Schulte, “Caught in chains: Claim-first transactions for cross-blockchain asset transfers — whitepaper,” 2018. [Online]. Available: https://dsg.tuwien.ac.at/projects/tast/pub/tast-white-paper-2.pdf

[38] P. Lafourcade and M. Lombard-Platet, “About blockchain interoperability,” *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 643, 2020.

[39] A. Garoffolo, D. Kaidalov, and R. Oliynykov, “Zendoo: a zk-snark verifiable cross-chain transfer protocol enabling decoupled and decentralized sidechains,” *arXiv*, 2020.

[40] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, “Succinct non-interactive zero knowledge for a von neumann architecture,” in *Proceedings of the 23rd USENIX Conference on Security Symposium*, ser. SEC’14. USA: USENIX Association, 2014, p. 781–796.

[41] “Hashlock - bitcoin wiki.” [Online]. Available: https://en.bitcoin.it/wiki/Hashlock

[42] “Timelock - bitcoin wiki.” [Online]. Available: https://en.bitcoin.it/wiki/Timelock

[43] L. Deng, H. Chen, J. Zeng, and L. J. Zhang, “Research on cross-chain technology based on sidechain and hash-locking,” vol. 10973 LNCS. Springer Verlag, 6 2018, pp. 144–151.

[44] “Architecture · polkadot wiki.” [Online]. Available: https://wiki.polkadot.network/docs/en/learn-architecture

[45] “Substrate — bison trails.” [Online]. Available: https://bisontrails.co/substrate/

[46] M. Borkowski, M. Sigwart, P. Frauenthaler, T. Hukkinen, and S. Schulte, “Dextt: Deterministic cross-blockchain token transfers,” *IEEE Access*, vol. 7, pp. 111 030–111 042, 2019.

[47] G. Verdian, P. Tasca, C. Paterson, and G. Mondelli, “Quant overledger ® whitepaper release v0.1 (alpha),” 2018. [Online]. Available: https://uploads-ssl.webflow.com/6006946fee85fda61f666256/60211c93f1cc59419c779c42_Quant_Overledger_Whitepaper_Sep_2019.pdf

[48] E. Fynn, A. Bessani, and F. Pedone, “Smart contracts on the move,” *arXiv*, 2020.

[49] “hyperledger/burrow: https://wiki.hyperledger.org/display/burrow.” [Online]. Available: https://github.com/hyperledger/burrow

[50] P. P. Ippolito, “Game theory in artificial intelligence — towards data science.” [Online]. Available: https://towardsdatascience.com/game-theory-in-artificial-intelligence-57a7937e1b88

[51] K. Chatterjee, A. K. Goharshady, R. Ibsen-Jensen, and Y. Velner, “Ergodic mean-payoff games for the analysis of attacks in crypto-currencies,” in *CONCUR*, 2018.

[52] R. Laraki, “The splitting game and applications,” *International Journal of Game Theory*, vol. 30, pp. 359–376, 2002. [Online]. Available: https://link.springer.com/article/10.1007/s001820100085

[53] I. Eyal, “The miner’s dilemma,” *arXiv*, 2014.

[54] Y. Zhen, M. Yue, C. Zhong-Yu, T. Chang-Bing, and C. Xin, “Zero-determinant strategy for the algorithm optimize of blockchain pow consensus.” IEEE Computer Society, 9 2017, pp. 1441–1446.

[55] B. Johnson, A. Laszka, J. Grossklags, M. Vasek, and T. Moore, “Game-theoretic analysis of ddos attacks against big and small mining pools,” vol. 8438, 03 2014.

[56] M. Mirkin, Y. Ji, J. Pang, A. Klages-Mundt, I. Eyal, and A. Juels, “Bdos: Blockchain denial-of-service,” in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS ’20. Association for Computing Machinery, 2020, p. 601–619. [Online]. Available: https://doi.org/10.1145/3372297.3417247

[57] M. Herlihy, “Atomic cross-chain swaps,” *Proceedings of the Annual ACM Symposium on Principles of Distributed Computing*, pp. 245–254, 7 2018.

[58] “Alt chains and atomic transfers.” [Online]. Available: https://bitcointalk.org/index.php?topic=193281.msg2224949#msg2224949

[59] E. Puggioni, A. Shaghaghi, R. Doss, and S. S. Kanhere, “Towards decentralized iot updates delivery leveraging blockchain and zero-knowledge proofs,” *arXiv*, 2020.

[60] S. Goldwasser, S. Micali, and C. Rackoff, “Knowledge complexity of interactive proof-systems.” ACM (Order n 508850), 1985, pp. 291–304. [Online]. Available: http://portal.acm.org/citation.cfm?doid=22145.22178

[61] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, “Bulletproofs: Short proofs for confidential transactions and more,” in *2018 IEEE Symposium on Security and Privacy (SP)*, 2018, pp. 315–334.

[62] “Odap - open digital asset protocol draft.” [Online]. Available: https://datatracker.ietf.org/doc/draft-hargreaves-odap/

[63] “A trustless, general-purpose polkadot — ethereum bridge — by snowfork - polkadot — ethereum bridge.” [Online]. Available: https://snowfork.substack.com/p/a-trustless-general-purpose-polkadot

[64] D. Harz, L. Gudgeon, A. Gervais, and W. J. Knottenbelt, “Balance: Dynamic adjustment of cryptocurrency deposits,” in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS ’19. Association for Computing Machinery, 2019, p. 1485–1502. [Online]. Available: https://doi.org/10.1145/3319535.3354221

[65] H. A. Kalodner, S. Goldfeder, X. Chen, S. M. Weinberg, and E. W. Felten, “Arbitrum: Scalable, private smart contracts,” in *USENIX Security Symposium*, 2018.

[66] “Archi – open source archimate modelling.” [Online]. Available: https://www.archimatetool.com/

[67] T. Hardjono, “Blockchain gateways, bridges and delegated hash-locks,” *ArXiv*, vol. abs/2102.03933, 2021.

[68] ——, “Attestation infrastructures for private wallets,” *ArXiv*, vol. abs/2102.12473, 2021.

[69] “Polkadot consensus · polkadot wiki.” [Online]. Available: https://wiki.polkadot.network/docs/learn-consensus

[70] “Polkadot builders starter’s guide · polkadot wiki.” [Online]. Available: https://wiki.polkadot.network/docs/build-build-with-polkadot/

[71] “Parity’s ink! to write smart contracts.” [Online]. Available: https://github.com/paritytech/ink

[72] “Bridges · polkadot wiki.” [Online]. Available: https://wiki.polkadot.network/docs/learn-bridges

[73] “Hyperledger besu.” [Online]. Available: https://besu.hyperledger.org/en/stable/

[74] C. Labs, “Edgeware: An adaptive smart-contract blockchain commonwealth — whitepaper,” 2019. [Online]. Available: https://static.coinpaprika.com/storage/cdn/whitepapers/10576761.pdf

[75] “An introduction to prometheus metrics and performance monitoring — enable sysadmin.” [Online]. Available: https://www.redhat.com/sysadmin/introduction-prometheus-metrics-and-performance-monitoring

[76] “Toolchains - the rustup book.” [Online]. Available: https://rust-lang.github.io/rustup/concepts/toolchains.html

[77] “A look at blockchain in cross-border payments — paymentsjournal.” [Online]. Available: https://www.paymentsjournal.com/a-look-at-blockchain-in-cross-border-payments/

[78] “Remittance prices worldwide quarterly, march 2021 — the world bank.” [Online]. Available: https://remittanceprices.worldbank.org/sites/default/files/rpw_main_report_and_annex_q121_final.pdf

[79] “How long does a ripple (xrp) transaction take? - quora.” [Online]. Available: https://www.quora.com/How-long-does-a-Ripple-XRP-transaction-take