b
UNIVERSITÄT
BERN

Analysing Inter-Blockchain Communication

Bachelor Thesis

Raphael Markus Jean-Maxim Fehr
from
Bern, Switzerland

Faculty of Science, University of Bern

15.Dezember.2023

Prof. Christian Cachin
David Lehnherr
Cryptology and Data Security Group
Institute of Computer Science
University of Bern, Switzerland

Abstract

Cosmos is an ecosystem that enables blockchains to connect by providing a common interface. This interface has an inter-blockchain communication (IBC) protocol that allows users to transfer assets across different blockchains. This protocol also enables an atomic swap of assets from different blockchains. In this Bachelor thesis, we first isolate the theoretical aspect of IBC in a modular way, and then we analyse the atomic swap, which allows users to swap their assets with other users.

Contents

Chapter 1

Introduction

Motivation As children, we often traded cards on the playground. In this activity, we face challenges of trust and fairness. When someone breaks their promise, we learned the hard way that we could not always trust the other children, and we also learned that a third party would not always be on our side. Similar problems exist in the field of blockchain technology, where trustless interactions and secure exchanges are crucial. This anecdote about trading cards on the playground serves as an analogy for the challenges we face in the blockchain world today. Just as trading cards as a child taught us not to trust other children, blockchain users face similar dilemmas when transacting across different chains. There is also some risk in using a third party, as that party might be corrupt. So not all problems can be solved with a third party, we need a reliable protocol to ensure fairness and security in these exchanges. Cosmos addresses this problem by developing an inter blockchain communication which also allows swapping items between blockchains. This bachelor thesis deals with the Inter Blockchain Communication (IBC) protocol developed by Cosmos, with the aim of analysing its theoretical aspects and describing it with modules. [4]

Cosmos Within a blockchain, three core components collaborate to guarantee its proper operation and security: the network-, consensus-, and application layers, each with distinct responsibilities. The network layer plays a critical role in enabling communication between diverse nodes within the blockchain network. The consensus layer ensures all nodes on the network are informed of the latest transactions and updates, facilitating efficient dissemination throughout the system. This plays a crucial role in maintaining a synchronized and up-to-date blockchain ledger, underpinning its integrity and immutability. Its primary function is to enable nodes on the network to reach an agreement on the current state of the system. Through a range of consensus mechanisms, nodes collaboratively validate and confirm transactional validity and reach a mutual agreement on a singular version of the truth. This agreement safeguards the blockchain from any tampering or malicious attacks. The application layer serves as the core of the blockchain system, responsible for the handling and execution of transactions. When transactions are started by users, the application layer is tasked with updating the state of the blockchain based on them. By collaborating, the networking, consensus, and application layers create a strong and decentralized blockchain ecosystem. The networking layer facilitates the smooth transmission of data, while the consensus layer establishes trust and mutual understanding among nodes. Working in tandem, the application layer validates and updates the blockchain's transactional state. [6]

Tendermint Stack One way to contribute to the Cosmos universe is by using the provided Tendermint stack. The stack contains three things: the Tendermint Core, the Cosmos SDK, and the IBC protocol. The Tendermint Core provides a Byzantine Fault Tolerant consensus algorithm. The Cosmos SDK allows us to build our application framework on top of this consensus algorithm. Therefore, the Cosmos SDK allows us to create modules for our Blockchains. For example, the Authentication and Bank module, that create and authenticate new addresses, or the IBC Module which enables communication with other

Blockchains. In order for two IBC modules that belong to different chains to communicate with each other, they need to follow the IBC protocol. Further, if two clients on different Blockchains wish to exchange assets, they must follow the IBC protocol as well. [18] [26] [27]

Related Work Most academic papers and documentation focus on the practical implementation of blockchains within the Cosmos system. The topics covered include guidance on setting up blockchains, the key requirements for a successful implementation, and the strategies for achieving this. The same is with the documentation from Cosmos, which sets the focus on the implementation part. In this bachelor work, we will set our focus on the theoretical foundation.

In this work We will set our focus on the theoretical aspect from IBC which allows us to communicate over the boarder of blockchains and the Atomic Swap, which enables to trade assets on two different Blockchains. For this, we will first start in the with the preliminaries section, which gives us the tool that we need for our intent. In Section 3, we will dive deeper into the Cosmos. We will break the Inter Blockchain Communication down to its modules. With this break down, we intend to analyse the Atomic swap in Section 4, which allows two users to exchange their asset. We will draw our conclusion in Section 5.

Chapter 2

Preliminaries

To analyse IBC, we need to have concepts of Hash Functions (c.f. Rosulek [23]), Merkle Trees (c.f. Merkle [20]) and Vector Commitment (Catalano et al.[5] and Rosulek [23]) in mind. We will define these concepts in the following section.

2.1 Merkle Trees

For future reference we need to formally define what it means to be negligible.

Definition 2.1. A function $f : \mathbb{R} \mapsto \mathbb{R}$ is negligible if, for every polynomial p, we have

$$\lim_{\lambda \to \infty} p(\lambda) f(\lambda) = 0.$$

Hash Function A cryptographic hash function, denoted as H, where $H : \{0,1\}^* \mapsto \{0,1\}^\lambda$ maps a binary string of arbitrary length into a unique representation. Given a bit string $x \in \mathbb{R}$ as input, it produces a fixed length binary value $h \in \{0,1\}^1$ as output. The key property of a cryptographic hash function is its collision resistance, which means that it should be impossible for any process, even subject to random errors or malicious actions, to find two different input values x and x' such that $H(x) = H(x')$, meaning that the case that we can find an x that satisfy $H(x) = H(x')$ is negligible. [4]

Merkle Trees A hash tree or *Merkle Tree* is a tree in which each "leaf" (node) is labelled with the cryptographic hash of a data block, and each node that is not a leaf is labelled with the cryptographic hash of the labels of its child nodes. Figure 2.1 shows a *Merkle Tree* with the values v_1, v_2, v_3 and v_4. We hash the values for the parent node, we concatenate the child nodes and hash it. When we know the root, the value of $h((hv_3)||h(v_4))$ and $h(v_2)$ we can validate that the items v_1 with its value is on the first place without knowing the value of v_2, v_3 or v_4.

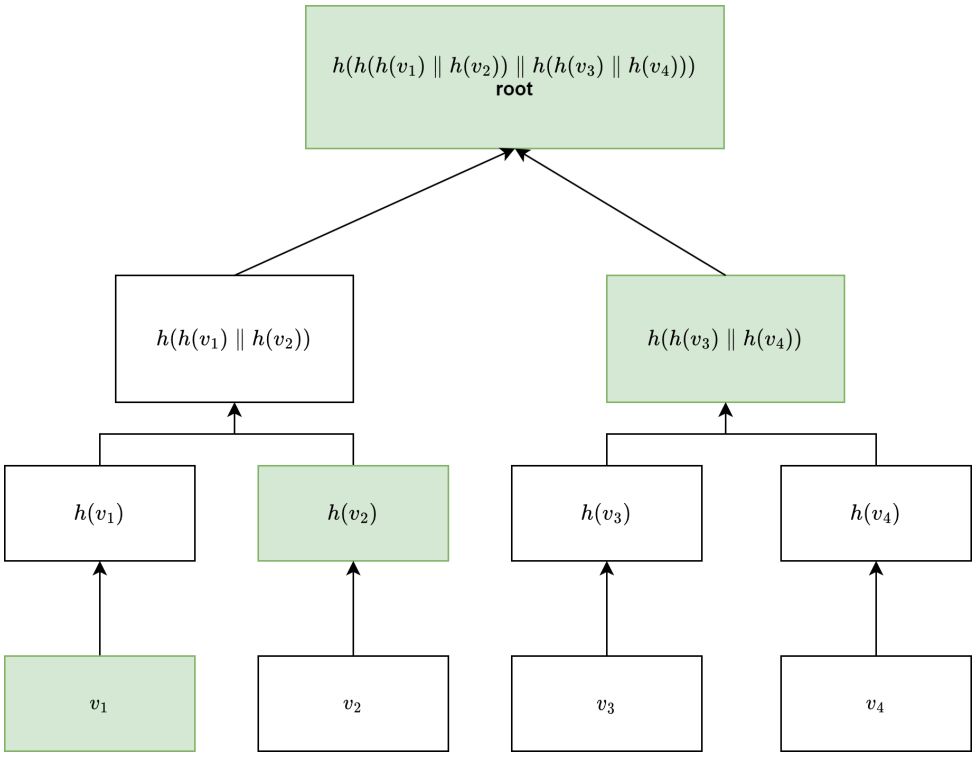


Figure 2.1. An illustration of a Merkle Tree of the values v_1, v_2, v_3 and v_4. The green fields illustrate, which values are necessary for a commitment for v_1.

2.2 Commitment Scheme

A *Commitment Scheme* allows us to commit to the values in the Mekrle Tree. We follow the paper Vector Commitments and their Applications by Catalano et al. [5]

Parameter	**Description**
Security Parameter (λ)	A value that sets the security parameter.
Message Space (M)	The set of all possible values that elements in the vector can assume.
Vector Length (q)	The total number of elements which is polynomial.
Public Parameters (pp)	These are public parameters used in the system.
Vector (v)	A sequence of values or elements from the message space M with a length n. The notation $v[i]$ means the value at place i from the vector v.
Auxiliary Information (aux)	Additional data required for creating openings.
Index (i)	Represents a specific position in the vector, ranging from 1 to n.
Value (y)	A specific element from the message space M.

Table 2.1. These are the parameters for the Commitment Scheme

Commitment Scheme A *Commitment Scheme* consist of the followings four algorithms for the parameters consider the Table 2.1

- KeyGen$(1^\lambda, q) \rightarrow pp$: The key generation algorithm produces public parameters pp.

- *Com*$(pp, v) \rightarrow (C, aux)$: For input pp and a message vector $v \in M^n$, the commit algorithm returns a commitment C and auxiliary information *aux* needed to create openings.
- *Open*$(pp, i, aux) \rightarrow \pi_i$: Given the public parameters pp, an index $i \in \{1, ..., n\}$ and auxiliary information aux, the opening algorithm outputs an opening π_i.
- *Ver*$(pp, C, i, y, \pi_i) \rightarrow b \in \{false, true\}$: Given the public parameters pp, a commitment C, an index $i \in [n]$, a value $y \in M$, and an opening π_i, the verification will return either *true* or *false*.

In the *Commitment Scheme* we have the *KeyGen* algorithm which is used to generate the keys. The *Com* creates a commitment which is used to verify the opening. The *Open* algorithm generate a proof which can prove that our value is in the commitment. The *Ver* algorithm which allows us to verify the opening. A commitment scheme needs to satisfy *completeness*, *soundness*, and *position binding*.

Completeness For any given public parameters pp, commitment C, and auxiliary information *aux*, the following holds:

$$\text{If } C = \textit{Com}(pp, v) \text{ and } \pi_i = \textit{Open}(pp, i, aux).$$

Then the probability that

$$\textit{Ver}(pp, C, i, v[i], \pi_i) = \textit{false}$$

is negligible in λ.

Soundness Consider any given public parameters pp, commitment C, and for every possible opening π_i and every possible value y not at the position [i] in the committed vector v: The probability that

$$\text{If } \textit{Ver}(pp, C, i, y, \pi_i) = \textit{true}$$

is negligible in λ.

Position Binding For a given set of public parameters pp, commitment C, and auxiliary information *aux*, there exists a unique value $v[i]$, meaning a value at place i, for which:

$$C = \textit{Com}(pp, v[i]) \text{ and } \pi_i = \textit{Open}(pp, i, aux).$$

The probability that

$$\textit{Ver}(pp, C, i, v[i], \pi_i) = \textit{false}$$

is negligible in λ. Considering all other values y where $v[i] \neq y$, and for each possible opening π_i: The probability that

$$\textit{Ver}(pp, C, i, y, \pi_i) = \textit{true}$$

is negligible in λ.

Chapter 3

Inter Blockchain Communication

In this section, we will describe Inter Blockchain Communication. As described in the introduction, IBC allows us, to communicate between Blockchains. We focused our work on the Cosmos Developer Portal[18] and the original Source code on Github [11]. We will first have a look at the actors, as they play a fundamental role in the Cosmos universe. This will include the *Relayer*, *Light Client* and the *Freezing Wallet*. We will break these actors down into modules, in the matier of CGR11/[4]. We also only have an interest in the theoretical concept, therefore we will not dive into the practical part on how to implement it. Once we understand the actors, we can move on to the transfer protocol. This will use the modules we defined earlier to illustrate and show how the packet flow and the proof system in IBC work in the IBC. Finally, we will break down the IBC concept into a module. A more documentation orientated overview can be found in the Appendix A.1.

3.1 Actors

Relayer In the context of blockchain interoperability, a *Relayer* serves as a central actor that facilitates communication between different blockchains. Much like a truck driver transporting goods between two locations, a *Relayer* listens for packets that need to be transferred to another blockchain. As an essential component of the IBC protocol, the *Relayer* acts as a client responsible for orchestrating the seamless transfer of data packets between interconnected blockchains. *Relayers* have access to full nodes of both the source and destination blockchains, giving them comprehensive knowledge of the entire blockchain's data, including all its blocks. A full node is like the blockchain's watchdog. It runs the blockchain software, thoroughly checks every transaction, and keeps a complete history. But anyone can run a full node to bolster the network's strength. This access enables them to efficiently query and send messages. Upon detecting events that require an IBC packet transfer, the *Relayer* delivers the packet then securely transmitted to the target chain for processing. The role of *Relayers* in the blockchain ecosystem is paramount. They play a critical role in ensuring the reliability and security. The *Relayers* can be operated by any participant in the network, reinforcing the decentralized nature of the entire blockchain system. In this paper, we will see the *Relayer* as a reliable broadcast. We assume that at least one *Relayer* is live and trusted. The *Relayer* makes a commitment to the message it will deliver. It can only accept a new message if it can prove that it delivered the old message. If there is a pool of *Relayers* where only one is trustworthy, the others will be determined after a certain time. The *Relayer* cannot stop the process, it can only slow it down because the *Relayer* knows only the destination of the packet, not what it delivers and to whom. The message the *Relayer* delivers can also be encrypted. We can break the *Relayer* down to the following, Module 3.1. In this module the *Relayer* event is triggered by a Relayer and the *Receive* event is handled and consumed by the receiving blockchain.

Module 3.1 Interface and properties of Relayer

Module

Name: Relayer , **instance** *rl*.

Events:

Request: $\langle$ *rl*, *Relay* | *m* $\rangle$: A Relayer broadcasts the Message m to the other Chain.

Indication: $\langle$ *rl*, *Receive* | m $\rangle$: A user receives a Message m.

Properties:

Rel1: *Reliable delivery:* If a correct process p sends a message m to a correct process q, then q eventually delivers m.

Rel2: *No duplication:* No message is delivered by a process more than once.

Rel3: *No creation:* If some process q delivers a message m with sender p, then m was previously sent to q by process p.

Light Client The *Light Client* holds the main header information of a full node, but it is compressed to save space. This information is needed to validate commitment proofs. *Light Clients* are provided by Cosmos. A *Light Client* is an algorithm that allows an actor to interact with a blockchain system. The client abstraction provides a formalized interface and set of requirements that allow the IBC protocol to seamlessly integrate with new ledgers running different consensus algorithms. As long as *Light Client* algorithms that meet the specified requirements are available, the IBC protocol can be used effectively with these new ledgers. This flexibility ensures compatibility and adaptability for the integration of different Blockchains into the IBC ecosystem. [16]
The light client has two main components, there are used to verify commitment. We will set there our focus, they are also used to detect attacking actions, which means it should detect conflicts and verify commits across multiple nodes. So we can break the *Light Client* down to the Module 3.2, with the *Check* Event and the properties *LC1*, which says that it terminates, and *LC2*, which says that it gives the correct response.

Module 3.2 Interface and properties of Light Clients

Module

Name: Light Client , **instance** *lc*.

Events:

Request: $\langle$ *lc*, *Check Proof* | π_i $\rangle$: A Client can query the Light Client with a Proof π_i, and checks if the proof is valid.

Indicator: $\langle$ *lc*, *Return* | π_i, b $\rangle$: Returns a boolean b that indicates whether π_i is a valid proof.

Properties:

LC1: *Termination:* If a correct process invokes an operation, then the operation completes.

LC2: *Correctness:* If π_i is valid $b = True$, if not $b = False$.

Trust assumption Only a trusted source can be a *Light Client*. Hence, we can assume their trustworthiness. It is assumed that no more than two-thirds of the *Light Clients* can be malicious. However, there is also the possibility of the trusted source being compromised. Therefore, it is assumed that no more than two-thirds of the *Light Clients* can be malicious. Therefore, it is assumed that no more than two-thirds of the *Light Clients* are malicious. This means that when querying the *Light Clients*, we require $2f+1$ *Light Clients*, where f is the amount of malicious actors. We already have this information. However, when

we receive two-thirds of the responses and assume that one-third of them are malicious while one-third are trustworthy, it becomes difficult to determine which responses are accurate. Therefore, to determine which responses we can trust, we require at least one additional Light Client. This means that we need 2f + 1 Light Clients.(c.f. Cosmos [25])
We can formally proof this:
Let us consider $N = 3f + 1$ we take a set $M \subseteq N$, and a set $M_2 \subseteq N$, where $|M_1| = 2f + 1$ and $|M_2| = 2f + 1$ so that $|M_1 \cap M_2| \leq f + 1$ otherwise it would be $|M - 1| + |M_2| > 3f + 1 = N$ where f nodes are Byzantine. We can conclude that $|(M_1 \cap M_2)| \geq 1$.
We can also illustrate this with the following Example.

Example Let us consider that we have ten *Light Clients* were one-third is malicious, and the rest is trustful. When we have only one response, we can not determine if it is malicious or not. When we take six responses of three times yes and three times no, we do not know which side is malicious. But when we have four times yes and three times no, we now can determine that the answer need to be yes and the three times no are malicious.

Freezing Wallet The freezing wallet is part of the blockchain. We created a module for this to have a better abstract look to it. The module has two events. First we have the freeze event. This is used to freeze an asset. Secondly, we have the unfreeze event, which is used to unfreeze an asset on a wallet on a blockchain.

Module 3.3 Interface and properties of the freezing Wallet

Module

Name: Freezing Wallet , **instance** *fw*.

Events:

Request: ⟨ *fw*, *Freeze* | *c* ⟩: Freeze an Asset.
Request: ⟨ *fw*, *Unfreeze* | *c*, *a* ⟩: Release an asset *c* on the address *a*.

Properties:

FW: *Spending:* An asset can only be spent when it is not frozen.

3.2 Transfer

To understand the packet flow, we first need to have a look at the channels and the connections. In order for a blockchain to interact with another blockchain, they need to create channels. This is to ensure that packets are delivered between certain modules on separate ledgers via this pipeline. A channel is a pipeline that ensures packets are delivered exactly once between specific modules on separate ledgers. It has at least one end capable of sending packets and one end capable of receiving packets. A channel serves as a pathway for packets to travel between a module on one blockchain and a module on another blockchain. Each channel is associated with a particular link, and a link can have any number of associated channels. Channels are payload independent. This means that the modules sending and receiving IBC packets decide how to construct the packet data to act on the incoming packet data. Use their own application logic to determine which state transactions to state transactions to apply according to what data the packet contains. [19]. Now that we understand a channel, we can begin to describe a connection. The connection abstraction encapsulates two stateful objects (connection ends) on two separate ledgers, each associated with a *Light Client* of the other blockchain, which together facilitate cross-blockchain

substate verification and packet forwarding through the channels. Connections are initiated by a typical *handshake protocol*.

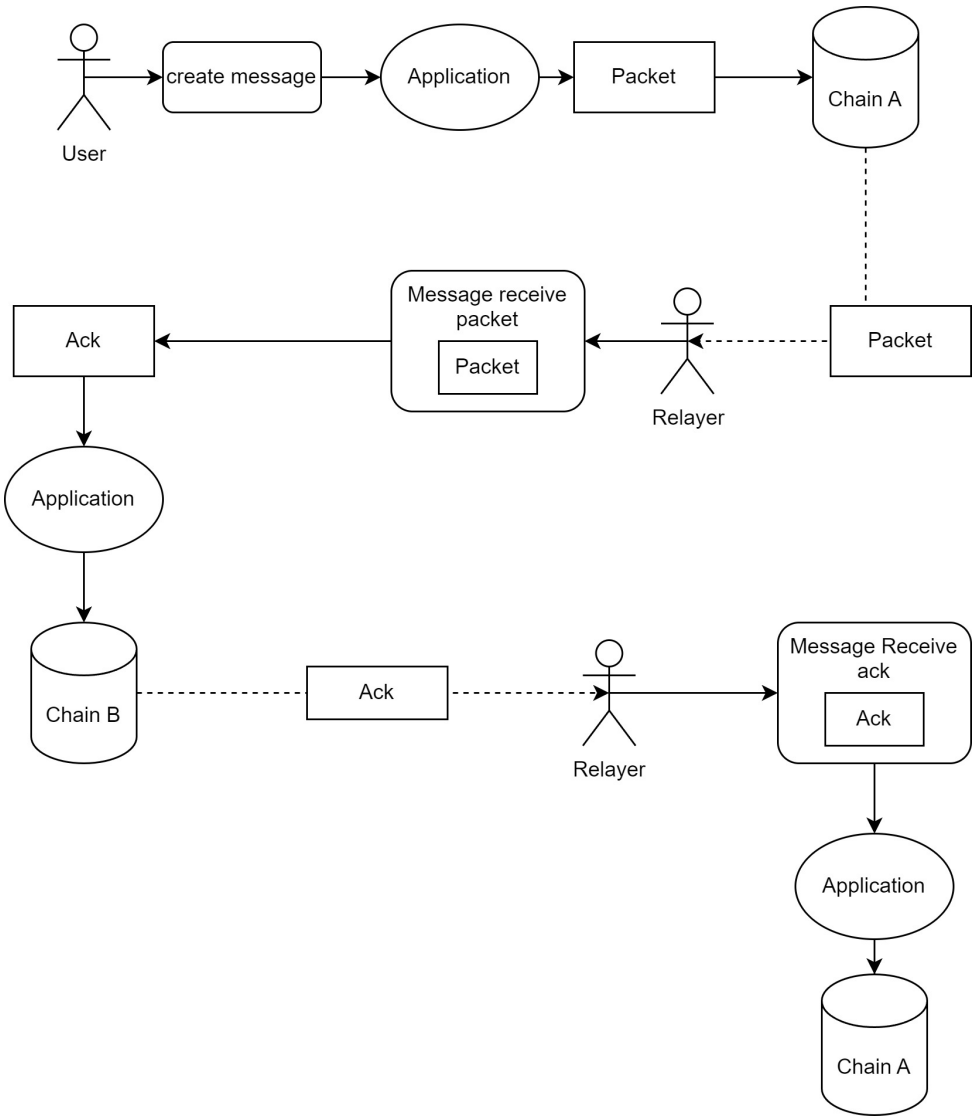


Figure 3.1. An overview for the flow of an packet.

Packet flow from a message to another blockchain For this part also see figure 3.1. In the process of interchain communication, a user A initiates the transfer by creating a message that needs to be sent to another blockchain B. This message is then passed to the application, also known as the handler, which is responsible for processing such messages. The handler takes the message and places it on the source Blockchain A, indicating the intention to transfer the data. At this stage, first a *Relayer* comes into play. At least one *Relayer* continuously monitors Blockchain A (Doted line) and detects the presence of the message. Once the *Relayer* identifies the message, it captures it and encapsulates it in a Message Receive Packet (MRP). The *Relayer* then transmits this MRP to the Handler on the destination blockchain (Chain B). This one checks the validity of the MRP. Upon receipt of the MRP, the handler on Chain B processes the data contained in the packet, meaning where it checks the authentication and accuracy of the packets and the commitments in it. After successful processing, the handler issues an acknowledgement (Ack) to signal that the message has been received and processed. Another or the same *Relayer* then actively monitors Chain B (Doted line), looking for new ACKs. When the *Relayer*

detects the Ack, it captures it and includes it in a Message Receive Acknowledgment (MRA). Finally, this *Relayer* contributes this MRA back to Blockchain A, completing the interchain communication process. Through the involvement of *Relayers*, data packets are reliably and securely transferred between different blockchains, facilitating seamless interoperability and maintaining the decentralized nature of the blockchain ecosystem.

Proof System In IBC we use a commitment scheme similar to the scheme we have shown in the preliminaries. The value which is committed from a user A on the blockchain A is stored in a Merkle Tree. This allows to create and validate proofs if the value a user A says, is actually stored in the tree. (see Figure 2.1) Similar to the vector commitment, we have the following function.

- *calculateRoot*$(commitmentState) \rightarrow commitmentRoot$: This creates the root of the Merkle Tree. This is the commitment and used to verify proofs.
- *createProof*$(state, commitmentPath, value) \rightarrow \pi_i$: The createsProof is used to make a proof that the value is in the storage (Merkle Tree).
- *VerifyProof*$(root, \pi_i, path) \rightarrow b \in$*false*, *true*$\}$: Is used to verify the validity of the proof.

Where *calculateRoot* is similarly to *Com* from the Vector Commitment, the *createProof* is the *Open* and the *VerifyProof* is the *Ver* function. The commitment path is the place from the value. We notice that we don't have a Key Generator, this will be handled from the blockchain itself. We can see this in the Figure 3.2.

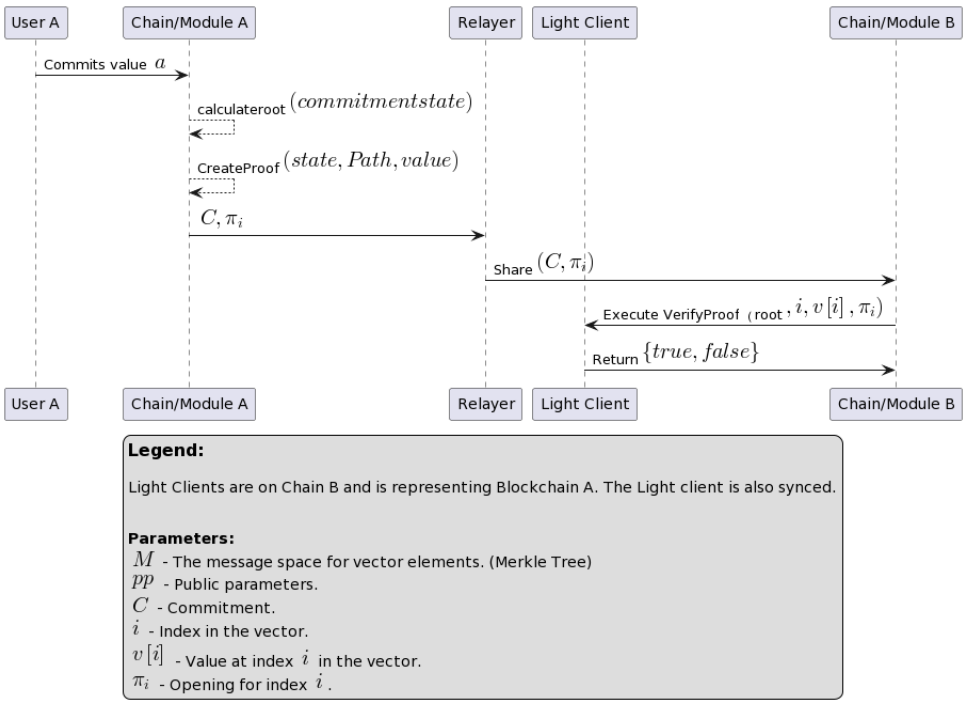


Figure 3.2. An UML Diagram for the commitment scheme.

Consider Figure 3.3. A correct actor A starts the proof machinery by sending a message m (1) to the application layer, that is blockchain A. Upon recording m A updates the Merkle Tree (2) and generates a proof π_i. Observe that the $\bar{lc}_a$ on blockchain B is assumed to be synchronized with A and thus both have the same *root*. Eventually, the *Relayer* detects (m, π_i) and delivers it by using the IBC protocol (3). The blockchain B then checks the proof with help of the *Light Clients* (4). The Blockchain B will then deliver the message if the proof is valid (5).

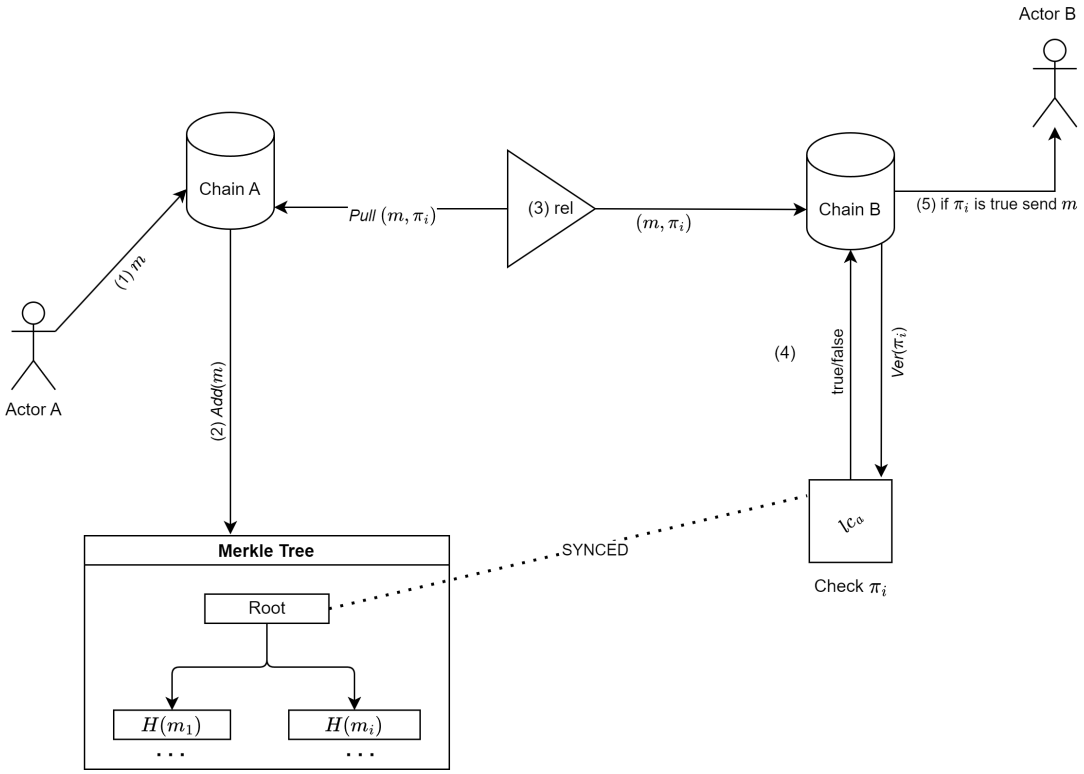


Figure 3.3. An UML Diagram that illustrate the system of the Merkle Tree.

3.3 IBC

In this section, the IBC module will be developed and a description of the module will be provided.

IBC The IBC Module 3.4 has two events. The send and the receive events. These are used to send a message or to receive a message. This is described in detail in the algorithms in the next part. We have three Properties that are similar to the reliable broadcast (c.f. Cachin et al. [4]). The IBC Module need to satisfy the following. Validity, meaning that the process function properly. No duplication, meaning that every message is delivered once, so that the receiving chain does not receive two and process two messages. No creation, meaning that the message which is processed is created by a user. Additive, it has the property correctness, which means that the msg only go through, if π_i is a valid proof. We also need to notice that the packets are sent to Blockchain.

Send and receive a message So there we have Algorithm 1 IBC send and receive in the IBC module. We have seen the trust assumption in Section 3.1. Therefore, we will use this again to ensure the process holds under this assumption. To send a message m to the destination blockchain B, the user A gives the message to the IBC handler module from his blockchain. This module sets this message in a chain of messages with need to be delivered by a *Relayer*. This triggers a *Relayer* to take this message, read the destination address, and deliver this message to the other blockchain where it is processed and the other user Bob receives the message *msg* if it had a valid proof π_i. Blockchain B receives the message per a *Relayer*. The module queries the *Light Clients* if the packet contains a proof. It will wait until it has $N - f$ as response and validates the proof. If the proof is valid, the message will be processed.

Module 3.4 Interface and properties of IBC

Module

Name: Inter Blockchain Communication , **instance** ibc.

Events:

Request: $\langle$ *ibc*, *Send* | *msg*, B $\rangle$: A process sends a IBC packet *msg* to a Blockchain B.
Request: $\langle$ *ibc*, *receive* | *msg*, B $\rangle$: A Blockchain B receives an IBC packet msg.

Properties:

IBC1: *Validity:* If a correct process p sends a message m, then a Relayer eventually delivers m.
IBC2: *No duplication*: No message is delivered and processed more than once.
IBC3: *No creation*: If a process delivers a message m with sender s, then m was previously created by an prozess a.
IBC4: *Correctness*: Msg will only delivered if it contains a valid proof.

Algorithm 1 IBC send and receive

Implements:
IBC module, **instance** *ibc* .
Uses:
Relayer, **instance** *rel*,
Light Client,**instance** *lc*

upon event $\langle$ *ibc*, *init* $\rangle$
TrueCount $= 0$
FalseCount $= 0$
upon event $\langle$ *send*, *ibc* | *msg*, B $\rangle$ **do**
root = calculateRoot(State)
π_i = createProof(State,i , m)
trigger $\langle$ *relay*, *Relayer* | *msg*$||\pi_i$ $\rangle$

upon event $\langle$ *receive*, *Relayer* | *msg*$||\pi_i$ $\rangle$ **do**
for each *Light Client* *lc*
wait until *TrueCount* $= 2f + 1$
trigger $\langle$ *receive*, *ibc* | *msg*, B $\rangle$

uppon event $\langle$ *return*, *lc* | π_i, b $\rangle$
if b **then**
TrueCount $=$ *TrueCount* $+ 1$

Chapter 4

Atomic Swap

Now we can dive in the *Atomic Swap* which allows two users to exchange their asset over different blockchains. We will first have a look at what is an *Atomic Swap* and what properties it has. Then we will build the module.

4.1 Atomic Swap

An *Atomic Swap* is a mechanism for exchanging fungible tokens, a and b, between two parties A and B on separate blockchain networks. Which satisfies the properties, guarantee of exchange, refundable, order cancelling and atomicity. We have the following properties from the IBC documentation [9].

- **GE:** *Guarantee of Exchange*
- **RF:** *Refundable*
- **OC:** *Order Cancellation*
- **AT:** *Atomicity*

Atomic Swap This Module describes the *Atomic Swap*. We have three Events and four desire properties. *Guarantee of Exchange:* This principle ensures that there is no situation where a user receives tokens without receiving the promised exchange. In other words, it guarantees that token exchanges are fair and that users get what they expect in return for their tokens. *Refundable:* This means that tokens are refundable under certain circumstances. Tokens can be refunded by freezing them when a timeout occurs (when a transaction takes too long) or when the user cancels an order. This feature provides users with a safety net and ensures that they can get their tokens back if necessary. *Order Cancellation:* This feature allows users to cancel. In other words, if a user has placed an order to exchange tokens, they have the flexibility to cancel that order as long as no one has accepted it. This feature adds a level of control and convenience for the user. *Atomicity:* Atomicity refers to the property that an exchange of one token for another is either a complete success or a complete failure. There are no partial exchanges or incomplete transactions. When an exchange is initiated, it is executed as a single, indivisible unit, ensuring that it either completes or fails completely.
The Atomic Swap Module 4.1 has three events: *start*, *take* and *cancel*. The start event allows a user on a blockchain to initiate the atomic swap. This is where they freeze their assets. Then the user creates the proof for their order. Which shows their commitment, meaning which asset they froze and what they want. This is delivered by a *Relayer* and taken to the taker's blockchain. There, the proof is verified by a *Light Client* . A user can then take this order. This user freezes their assets on blockchain B. They send a packet to the taker blockchain along with a proof, which proofs that the tokens are locked and their commitment. This happens through a *Relayer*. Here, the *Light Clients* will be queried and validates the proof. If the proof is valid, the assets are transferred from the freezing wallet to the taker's wallet on

the maker's blockchain. A proof is generated and sent to the taker's blockchain. A *Light Client* on the taker blockchain verifies the proof. The asset will be unfrozen and sent to the maker's wallet on the taker blockchain.

Module 4.1 Interface and properties of Atomic Swap (AS)

Module

Name: Atomic Swap , **instance** as.

Events:

Request: ⟨ *Start*, *as* | *Asset A*, *Asset B*, *Receive Address* ⟩: An user starts a swap.

Request: ⟨ *Take*, *as* | *order*, *Token B*, *Receive address* ⟩: An user takes a swap .

Request: ⟨ *Cancel*, *as* | m ⟩: Cancel the Swap.

Properties:

GE: *Guarantee of Exchange:* No occurrence of a user receiving tokens without the equivalent promised exchange.

RF: *Refundable:* Tokens are refunded by freezing when a timeout occurs, or when an order is cancelled.

OC: *Order Cancellation:* Orders without takers can be cancelled.

AT: *Atomicity:* An exchange of one token for another where it is either a total success or a total failure.

Flow The flow of the *Atomic Swap* is illustrated on the Figure 4.1.

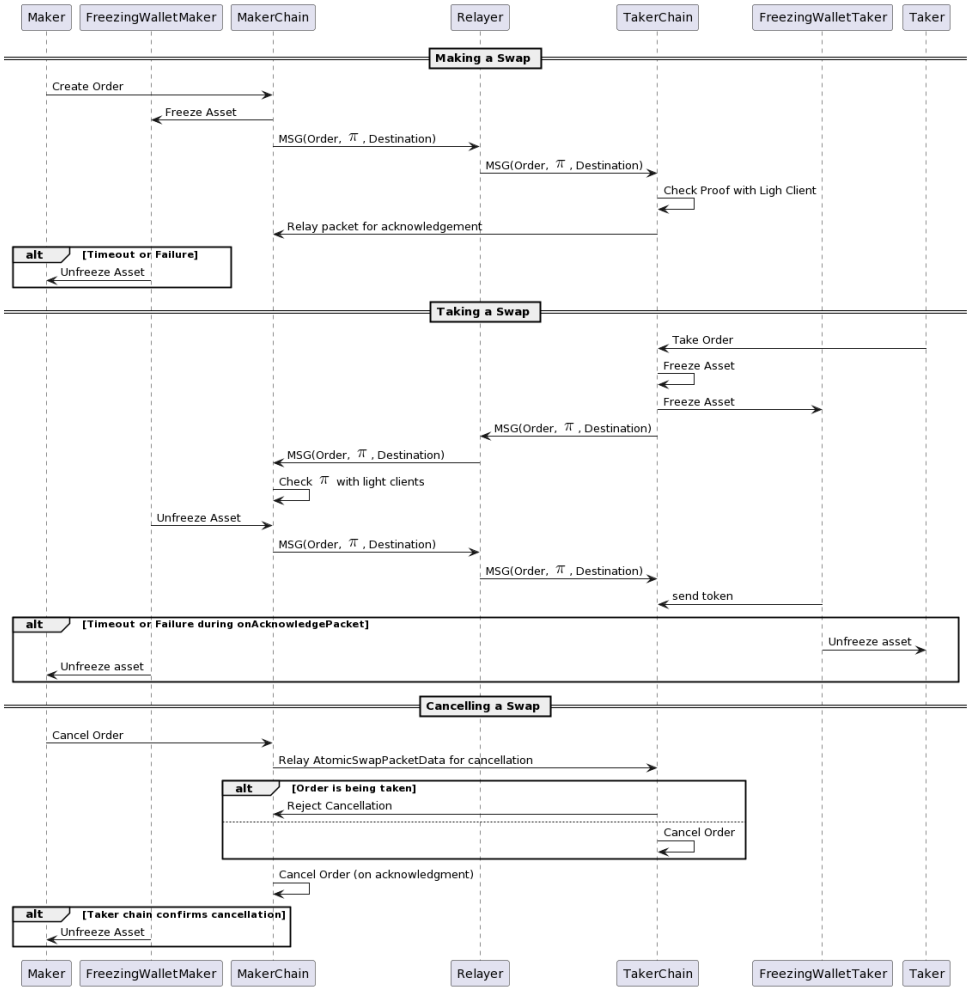


Figure 4.1. An UML Diagram of the flow of Atomic Swap.

Start a swap To start a swap between two users, Alice and Bob, who agreed on a side channel to the trade, Alice starts her order and locks her asset on the maker blockchain. She does this with freezing her asset on the maker blockchain. She creates a message which contains the amount she offers, the amount she wants, a proof of her commitment and a destination address of her wallet in the taker blockchain. A *Relayer* delivers this message to the taker blockchain, where a *Light Client* checks the proof. When the proof is valid. An order is created on the Taker blockchain. A *Relayer* relays then an acknowledgement back to the maker blockchain.

Algorithm 2 Atomic Swap Start

Implements:
Atomic Swap module, **instance** *as* .
Uses:
IBC, **instance** *ibc*,
Freezing Wallet, **instance** *fw*.

upon event $\langle$ *as*, *init* $\rangle$
$order = \perp$

upon event $\langle$ *as*, *Start* | *Asset A*, *Asset B*, *Receive Address* $\rangle$ **do**
trigger$\langle$ *fw*, *Freeze* | Asset A $\rangle$
$msg =$ *Asset A* || *Asset B* || *Receive address*
$order = msg$
trigger $\langle$ *send*, *ibc* | msg, B $\rangle$

Take a Swap To take a swap (c.f Algorithms 2) Bob, takes the order. This locks the order, and the asset will then be frozen on the taker blockchain. A message is created which contains: A proof of the lock and a receiving address. This is relayed by an *Relayer* to the blockchain A. Where it will be checked through the *Light Clients*, if it is valid, the amount on the freezing wallet is sent to Bobs address on the blockchain A. A message with a proof of this transaction is then relayed to the blockchain B. There a *Light Client* checks the proof if it is valid it releases the amount on the freezing wallet from Bob to Alice address on the blockchain B.

Algorithm 3 Atomic Swap Take a Swap

Implements:
Asset Transfer module, **instance** *as* .
Uses:
IBC, **instance** *ibc*,
Freezing Wallet, **instance** *fw*

upon event $\langle$ *as*, *Take* | *order*, *Asset B* $\rangle$ **do**
if $B \in order$ **then**
trigger $\langle$ *fw*, *Freeze* | Asset B $\rangle$
$msg = ACK$
trigger $\langle$ *send*, *ibc* | msg, *Maker Chain* $\rangle$

upon event $\langle$ *ibc*, *receive* | msg, *Maker Chain* $\rangle$
if $msg = ACK$ **then**
trigger$\langle$ *Unfreeze*, *fw* | *Token A*, *ReceivingAdress* $\rangle$
trigger $\langle$ *send*, *ibc* | msg, *Taker Chain* $\rangle$

upon event $\langle$ *ibc*, *receive* | msg, *Taker Chain* $\rangle$
trigger$\langle$ *Unfreeze*, *fw* | *Token B*, *ReceivingAdress* $\rangle$

Cancel In Algorithms 4, if a timeout occurs, the cancel action is executed. Note that the atomic swap either happens completely or not at all. This means that once the asset transfer from the freezing wallet of blockchain A happens, a cancel is no longer possible. We notice that both parties cancel the swap. Because the Taker gets his asset first, he can not get scammed. If the Maker asset is still frozen, the swap can be canceled.

Algorithm 4 Atomic Swap Cancel a Swap

Implements:
Atomic Swap module, **instance** *as* .

Uses:
IBC, **instance** *ibc*,
Freezing Wallet,**instance** *fw*

upon event ⟨ *cancel*, *AS* ⟩ **do**
if Maker asset is in freezing wallet **then**
trigger ⟨ *ibc*, *send* | *Abort*, *Maker Chain* ⟩
trigger⟨ *fw*, *Unfreeze* | Asset A, MakerAdress ⟩
trigger ⟨ *ibc*, *send* | *Abort*, *Taker Chain* ⟩
trigger⟨ *fw*, *Unfreeze* | Token B,TakerAdress ⟩

Flow Finally, we can illustrate the atomic swap with Figure 4.2. Which shows the flow of the protocol and how the Vector Commitment interfere with it.

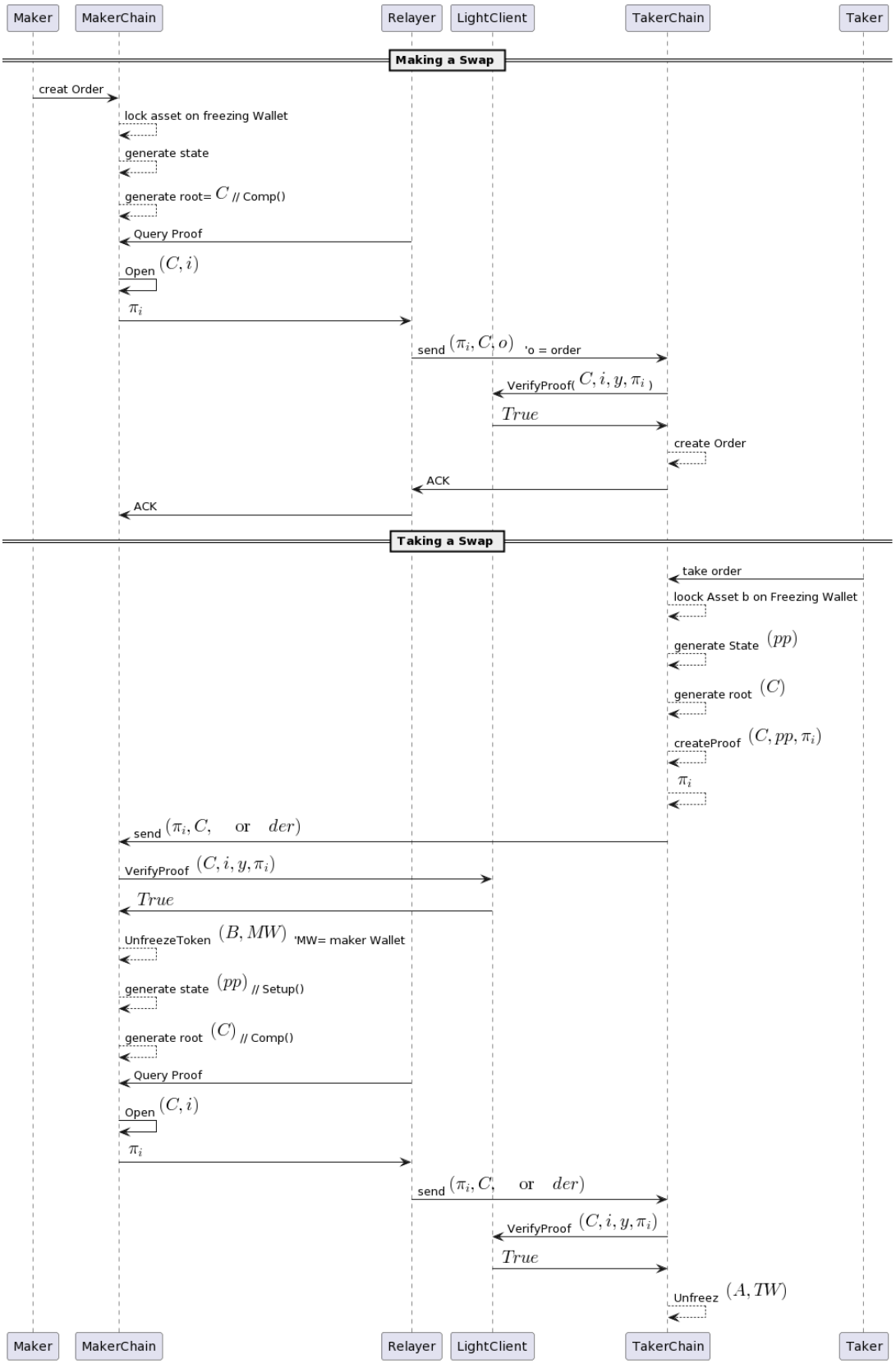


Figure 4.2. An UML Diagram that illustrate the commitment scheme in Atomic Swap.

Chapter 5

Conclusion

In our analysis of the IBC protocol and the atomic swap, we have broken the scheme into modular components, laying the foundation for further in-depth research. It is important to note that our study primarily delves into the theoretical aspects, leaving the practical implementation aspects untouched. The complexities and challenges that might arise during actual implementation, particularly in dealing with actors distributed across different blockchains, are beyond the scope of our work. Our analysis contains a breakdown of key actors within the IBC protocol, including the *Light Client*, *Relayer*, and the *Freezing Wallet*, which have been distilled into particle modules. We have introduced certain assumptions in our study, such as the requirement for $2f + 1$ of the *Light Clients* to be correct, and that at least one *Relayer* is trustful and alive. While this assumption is more difficult to fulfil, the *Reayer* assumption is relatively easy to satisfy. Our use of UML Diagrams has illustrated the concept of vector commitment within IBC and the Atomic Swap. To advance our understanding of the *Freezing Wallet*, *Relayers*, and *Light Clients*, further research is important. In our study, we have treated these elements as black boxes. We relied on assumptions about their behaviour. Similarly, the Commitment Scheme, slated for blockchain implementation, is contingent upon specific conditions, with trust vested in the blockchain itself as the bedrock of reliability. A lack of trust in the underlying blockchain infrastructure would undermine the entire IBC and Atomic Swap. Our modules can serve as a valuable toolkit for analysing additional aspects of the IBC protocol. Exploring various components of IBC and delving into the feasibility of implementing auctions within this framework are areas ripe for further investigation. In conclusion, our work serves as a springboard for deeper exploration, emphasizing the importance of scrutinizing both theoretical and practical aspects to ensure the robustness and trustworthiness of the IBC protocol in real-world applications.

Appendix A

Extra material

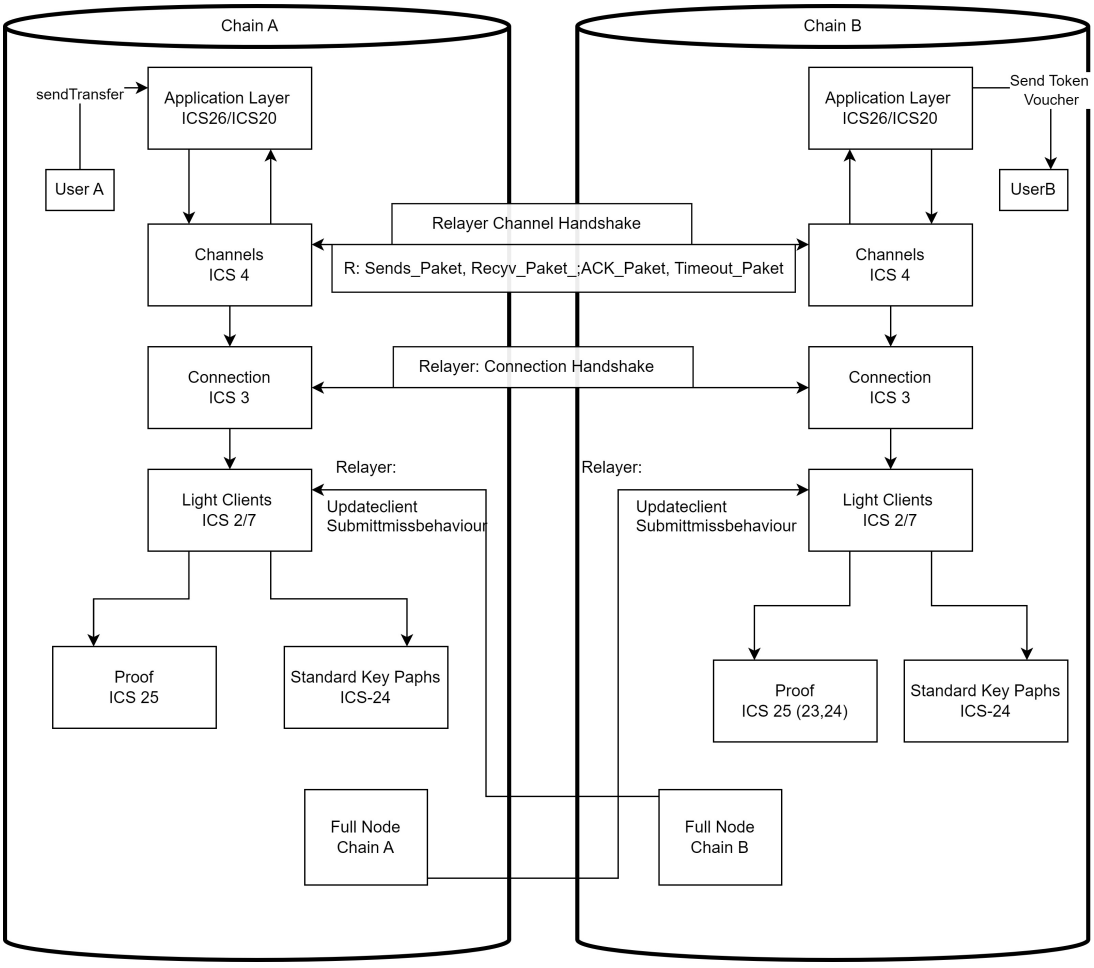


Figure A.1. An overview of the complexity of IBC.

Bibliography

[1] N. Asokan, V. Shoup, and M. Waidner, "Asynchronous protocols for optimistic fair exchange," in *Security and Privacy - 1998 IEEE Symposium on Security and Privacy, Oakland, CA, USA, May 3-6, 1998, Proceedings*, pp. 86–99, IEEE Computer Society, 1998.

[2] N. Asokan, V. Shoup, and M. Waidner, "Optimistic fair exchange of digital signatures," *IEEE J. Sel. Areas Commun.*, vol. 18, no. 4, pp. 593–610, 2000.

[3] Binance Academy, "Tendermint explained." `https://academy.binance.com/en/articles/tendermint-explained`, 2023. Accessed: December 3, 2023.

[4] C. Cachin, R. Guerraoui, and L. E. T. Rodrigues, *Introduction to Reliable and Secure Distributed Programming (2. ed.)*. Springer, 2011.

[5] D. Catalano and D. Fiore, "Vector commitments and their applications," in *Public-Key Cryptography - PKC 2013 - 16th International Conference on Practice and Theory in Public-Key Cryptography, Nara, Japan, February 26 - March 1, 2013. Proceedings* (K. Kurosawa and G. Hanaoka, eds.), vol. 7778 of *Lecture Notes in Computer Science*, pp. 55–72, Springer, 2013.

[6] J. Choi, "Demystifying cosmos: Atomic swaps, ethereum, polkadot, and the path to blockchain interoperability." `https://medium.com/the-spartan-group/demystifying-cosmos-atomic-swaps-ethereum-polkadot-and-the-path-to-blockchain-interoperability-d1a2d75c20d6`, 2019. Accessed: December 01,2023.

[7] CometBFT, "What is cometbft." `https://docs.cometbft.com/v0.34/introduction/what-is-cometbft.html`, 2023. Accessed: December 3, 2023.

[8] Cosmos, "Cosmos." `https://www.Cosmos.com/`. Accessed: July 3, 2023.

[9] Cosmos, "Cosmos ibc specification - ics-100 atomic swap." `https://github.com/cosmos/ibc/tree/main/spec/app/ics-100-atomic-swap`. Accessed: July 3, 2023.

[10] Cosmos, "Cosmos Tutorial." `https://tutorials.cosmos.network/hands-on-exercise/5-ibc-adv/2-relayer-intro.html`. Accessed: July 3, 2023.

[11] Cosmos, "IBC git Repo." `https://github.com/cosmos/ibc`. Accessed: July 3, 2023.

[12] Cosmos, "IBC git Repo ReadMe." `https://github.com/cosmos/ibc/blob/main/spec/core/ics-004-channel-and-packet-semantics/README.md#definitions`. Accessed: July 3, 2023.

[13] Cosmos, "Map of Zones." `https://www.mapofzones.com/`. Accessed: July 3, 2023.

[14] Cosmos, "Tutorial cosmos." `https://tutorials.cosmos.network/academy/2-cosmos-concepts/1-architecture.html#the-cosmos-sdk`, 2023. Accessed: December 3, 2023.

[15] J. Frankenfield, "Merkle tree in blockchain: What it is and how it works." https://www.investopedia.com/terms/m/merkle-tree.asp. Updated July 26, 2021. Reviewed by Erika Rasure and fact-checked by Amanda Jackson, Accessed: July 3, 2023.

[16] C. Goes, "The interblockchain communication protocol: An overview," *CoRR*, vol. abs/2006.15918, 2020.

[17] IBC Protocol, "Eli5: What is IBC?." https://medium.com/the-interchain-foundation/eli5-what-is-ibc-def44d7b5b4c. Published on Nov 15, 2022, Accessed: July 3, 2023.

[18] I. F. (ICF), "Cosmos academy - ibc." https://tutorials.cosmos.network/academy/3-ibc/. Accessed: July 3, 2023.

[19] J. KWon and E. Buchman, "Cosmos, a network of distrubuted ledgers." https://v1.cosmos.network/resources/whitepaper. Accessed: July 3, 2023.

[20] R. C. Merkle, *Secrecy, authentication, and public key systems.* Stanford university, 1979.

[21] A. Narayanan, J. Bonneau, E. W. Felten, A. Miller, and S. Goldfeder, *Bitcoin and Cryptocurrency Technologies - A Comprehensive Introduction*. Princeton University Press, 2016.

[22] A. Nitulescu, "Sok: Vector commitments." https://www.di.ens.fr/~nitulesc/files/vc-sok.pdf. Accessed: December 01,2023.

[23] M. Rosulek, *The Joy of Cryptography*. yofcryptography.com, 2021.

[24] W. Stallings, *Cryptography and network security - principles and practice (3. ed.)*. Prentice Hall, 2003.

[25] Tendermint, "Tendermint light client specification readme." https://github.com/tendermint/tendermint/blob/master/spec/light-client/README.md. Accessed: July 3, 2023.

[26] Tendermint Team, "What is tendermint." https://tendermint.com/, 2023. Accessed: December 3, 2023.

[27] Tendermint Team, "What is tendermint." https://docs.tendermint.com/v0.34/introduction/what-is-tendermint.html, 2023. Accessed: December 3, 2023.

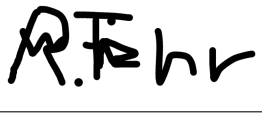
Erklärung

Erklärung gemäss Art. 30 RSL Phil.-nat. 18

Ich erkläre hiermit, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen benutzt habe. Alle Stellen, die wörtlich oder sinngemäss aus Quellen entnommen wurden, habe ich als solche gekennzeichnet. Mir ist bekannt, dass andernfalls der Senat gemäss Artikel 36 Absatz 1 Buchstabe r des Gesetzes vom 5. September 1996 über die Universität zum Entzug des auf Grund dieser Arbeit verliehenen Titels berechtigt ist.

Für die Zwecke der Begutachtung und der Überprüfung der Einhaltung der Selbständigkeitserklärung bzw. der Reglemente betreffend Plagiate erteile ich der Universität Bern das Recht, die dazu erforderlichen Personendaten zu bearbeiten und Nutzungshandlungen vorzunehmen, insbesondere die schriftliche Arbeit zu vervielfältigen und dauerhaft in einer Datenbank zu speichern sowie diese zur Überprüfung von Arbeiten Dritter zu verwenden oder hierzu zur Verfügung zu stellen.

Bern, 15.12.2023



Ort/Datum

Unterschrift