Master's thesis

NTNU
Norwegian University of Science and Technology

Hallvard Echtermeyer

Interoperability between heterogeneous Blockchains for Supply Chain

Master's thesis in Programvareutvikling
Supervisor: Li, Jingyue
May 2022

NTNU
Norwegian University of Science and Technology

Hallvard Echtermeyer

Interoperability between heterogeneous Blockchains for Supply Chain

Master’s thesis in Programvareutvikling
Supervisor: Li, Jingyue
May 2022

Norwegian University of Science and Technology

NTNU

Kunnskap for en bedre verden

Master Thesis

Interoperability between heterogeneous Blockchains for Supply Chains

Spring 2022

By

Hallvard Echtermeyer

Abstract

There is a need for interoperability in the current blockchain environment to shape a less fragmented blockchain ecosystem. In recent years there has come a focus on interoperability between homogenous blockchains. With the unique benefits provided by permissioned and permissionless blockchains, it is unlikely that homogenous interoperability will solve the interoperability issue. Therefore a focus has to be placed on heterogeneous interoperability.

This thesis developed two products following the design science paradigm, solving our three research questions:

- **Is there a way to interoperate data between two heterogeneous blockchains?**
- **What is the performance, and what are the issues of the selected interoperability solution?**
- **Can Self-sovereign identities be used to deanonymize cross-chain data between two heterogeneous blockchains?**

By answering these three questions, the current existing heterogeneous interoperability solutions were found, and a working solution was built based on the superior solution. Experiments were done using the developed products, focusing on the solution's performance and exposing problems it had. A second product was created to solve the anonymization issue discovered, using Self-sovereign identity, yielding the desired result of deanonymizing the data sent over the cross-chain solution.

Sammendrag

Det er et behov for interoperabilitet i det nåværende blokkjeder miljøet for a skape et mindre fragmentert blokkjeder økosystem. I de siste årene har det vært et økt fokus på interoperabilitet mellom homogene blokkjeder. Med de unike fordelene gitt av tillatede og tillatelsesløse blokkjeder, er det usannsynlig at homogen interoperabilitet vil løse interoperabilitetsproblemet. Derfor må det settes fokus på heterogen interoperabilitet.

Denne oppgaven utviklet to produkter etter det designvitenskapelige paradigmet, og løste våre tre forskningsspørsmål:

- **Er det en måte å interoperere data mellom to heterogene blokkjeder?**
- **Hva er ytelsen, og hva er problemene med den valgte interoperabilitetsløsningen?**
- **Kan Self-sovereign identiteter brukes til å deanonymisere krysskjededata mellom to heterogene blokkjeder?**

Ved å svare på disse tre spørsmålene ble dagens eksisterende heterogene interoperabilitetsløsninger funnet, og en fungerende løsning ble bygget basert på den overlegne løsningen. Eksperimenter ble utført ved bruk av de utviklede produktene, med fokus på løsningens ytelse og avdekket problemer den hadde. Et annet produkt ble opprettet for å løse anonymiseringsproblemet som ble oppdaget, ved å bruke Self-sovereign identitet, noe som ga det ønskede resultatet av deanonymisering av dataene sendt over krysskjedeløsningen.

Acknowledgments

I would first like to thank my thesis advisor, Associate Professor Jingyue Li of the Department of Computer Science at the Norwegian University of Science and Technology (NTNU). Assoc. Prof. Li was always available to answer questions, and was very understanding of my unfortunate sickness period, trying his best to accommodate my needs. I am also very grateful, for the input he gave during the writing of the master thesis, his knowledge was invaluable for the end result.

This thesis was in part supported by the Research Council of Norway (No.309494). The project PaaSforChain enabled me to communicate with colleagues from China, and also work on blockchain-related projects, giving valuable feedback and support.

I would also like to thank my colleague HaoMing Li, a student from NanJing University, and part of the PaaSforChain project. His expertise was tremendous in the creation of the BitxHub solution. his willingness and readiness to answer questions was a substantial help throughout the latter half of this thesis.

Last but certainly not least, I would like to thank my family and friends, for their support. My friends provided me with the entertainment and breaks I needed during this master thesis, and my family provided feedback on the thesis, and gave food and shelter for the final part of the thesis writing

Contents

List of Figures

List of Tables

Acronyms

SCIP	Smart Contract Invocation Protocol
SSI	Self-Sovereign Identity
pBTF	Practical byzantine fault tolerance
GO	Golang
CA	Certificate Authority
PoW	Proof of Work
EOA	Externally owned account
FA	Fungible assets
NFA	Non-fungible assets
DID	Decentralized identifier
SCM	Blockchain-based supply chain system
IBTP	Inter-Blockchain Transfer Protocol
SCL	Smart contract locator
RBTF	Redundant Byzantine Fault Tolerant protocol
TCP	Transmission Control Protocol
SDK	Software development toolkit
Dapp	Decentralized application
JSON	JavaScript Object Notation
IPFS	Inter-Planetary File System
API	Application Programming Interface
tps	transactions per second

1 Introduction

Cryptographer Devid Chaum first proposed the concept of a blockchain-like protocol in 1979 [13]. The realization of blockchains was first created by Satoshi Nakamoto in 2008 when he launched Bitcoin [14], a peer-to-peer online payment system that was immutable and trustless. Since the inception of blockchains, multiple new blockchains have come, all with their different niche use cases. Ethereum [15] built on Bitcoin, by introducing smart contracts, a truing-complete programing language, which allowed blockchains greater diversity in what a blockchain could do. Hyperledger Fabric challenged the notion of true decentralization and instead focused on making a blockchain that could benefit businesses. Allowing only a select few users into the blockchain could vastly increase performance and remove the gas cost needed in permissionless blockchains. Since 2008 there have come more and more new blockchains, all aiming to fill a niche, Vechain, IBM Blockchain, and Hyperledger Sawtooth, just to name a few. With the emergence of all the new blockchains, a new challenge came: interoperability. Interoperability can be defined as:

"the semantic dependence between distinct ledgers for the purpose of transferring or exchanging data or value, with assurances of validity or verifiability". [10]

Blockchains have historically been indifferent to the idea of interoperability. This indifference to future interoperability has led to a fragmented blockchain universe, where none can communicate efficiently with each other [16]. This lack of communication has limited users to only one blockchain, having to prioritize desirable attributes.

To tackle the need for interoperability between blockchains, there have come some interoperability solutions, using different methods trying to solve the problem, such as Polkadot [4], Hedera [17], Interledger [18], Smart Contract Invocation Protocol (SCIP) [9], Ermyas Abeb Relay [10] and BitxHub [19] for instance.

The focus of the current interoperability landscape is communication between homogeneous blockchains, that is, two compatible blockchains sharing the same consensus model, smart contracts, and authentication methods [16], like two Hyperledger blockchains or two Ethereum blockchains. Interoperability between heterogeneous blockchains is be an afterthought after the system has been developed, like Polkadot's Bridges [20] or Hederas Hashport [21]. Typically these created solutions only focus on interoperating Fungiable assets, something with value, taking a small fee for interoperating the asset. Businesses, however, place the most value in using blockchains for supply chains [22], where data is the most critical information to interoperate. For this reason, there was a desire to find out if it was possible to interoperate data between two heterogeneous blockchains, Hyperledger and Ethereum.

This thesis addresses three research questions described below:

Research question 1: Is there a way to interoperate data between two heterogeneous blockchains?

An analysis of blockchains found in the pre-study and master thesis found that two candidates could interoperate data between heterogeneous blockchains. These were SCIP and BitxHub. However, of the blockchains, which can interoperate data between heterogeneous blockchains, none have provided any literature about the performance of the solutions. The desire to uncover more about the performance led to the second research question:

Research question 2: What is the performance and what are the issues of the selected interoperability solution?

Heterogeneous blockchains will commonly have different methods for authenticating users [9, 19]. Hyperledger Fabric works with digital certificates to validate the users in their blockchain. These signatures are then used to sign transactions they wish to be placed on the blockchain, which is later validated and signed by the peers before the transaction is placed on the blockchain ledger [23]. Ethereum uses digital wallets linked to a public address. Each transaction set on the ledger by this wallet will display this address as the entity which created the transaction. For heterogeneous cross-chain transactions, problems arise when data is passed through the cross-chain solution. From the discovered heterogeneous interoperability solutions [19, 9], both leave the signing of information in their respective blockchains to the gateways. Leaving the signing to the gateways anonymizes the data sent over the interoperability solution. The information which potentially later would need to be audited will stop at the cross-chain gateway, as all knowledge of who placed information on the other blockchain is erased. A possible solution to this problem is Self-sovereign identities (SSI). Enabled through desirable qualities found in blockchains, SSI empowers users to reliably authenticate themselves digitally, relying on a new trust model between issuer, holder, and verifier. This problem and a potential solution led to the third research question:

Research question 3: Can Self-sovereign identities be used to deanonymize cross-chain data between two heterogeneous blockchains?

This thesis contributes the following findings:

- an overview of which interoperability solutions can interoperate heterogenonus data between each other
- a working interoperability solution made in BitxHub, interoperating between Ethereum and Hyperledger 2.3. Code provided here
- An analisis about the performance of the created interoperability solution

- A Hyperledger-Indy SSI solution used to deanonymize the information sent over the Interoperability solution. Code provided here

This thesis is structured in the following way:

- The background section 2 will list necessary information vital to understanding this thesis better
- The related work section 3 will prove the novelty of the research questions posed
- Research and Design section 4: will provide the motivation for choosing the questions and give an insight into the methods and designs used during this thesis.
- Results section 5 will provide the results found during this thesis
- Discussion section 6 will provide the findings in this thesis and discuss them
- Conclusion section 7 will summarise the findings and provide suggestions for future work.

2 Background

2.1 Definitions of Blockchains

Blockchain is a distributed ledger for recording transactions, maintained by many nodes, without a central authority, through a distributed cryptographic protocol.

This sentence or similar ones are typically used to describe a blockchain. For clarity, each part of the sentence will be explained.

A **Distributed ledger** is essentially a type of database that is shared, replicated, and synchronized among the members of the network. This gives blockchains the decentralization attribute, meaning that there is no single point of failure in blockchains. In this database, there is typically a timestamp for when the transaction happened in this database and a cryptographic signature unique to a user. The ability to have the ledger on all nodes makes blockchains immutable, as the information can't be changed after being put on the blockchain.

A **Node** in a blockchain is simply one entity that is holding onto the information which is sent, when there are made updates on the blockchain. It is the nodes' job to check if new information coming in is valid through some form of consensus algorithm. It stores the transaction history and will update new nodes in the cluster with the existing blockchain. Users can own or connect to existing nodes in public (permissionless) blockchains, like Infura nodes. In Private (permissioned) blockchains, a node might be a company, meaning more users can access that same node.

A **distributed cryptographic protocol** This is simply put, the rules by which the blockchain is governed. This includes how blocks agree on what information should be placed on the blockchain (Consensus Algorithm), how transactions are done on the blockchain (Smart Contracts), and who can see the information on the blockchain (permissioned, permissionless) and any other choice which defines the blockchain.

Blockchains are simply put a cluster of nodes all working together to validate and store data on distributed databases, giving an absolute truth of any situation going on in the blockchain to all nodes.

Blockchains can be divided into two categories, this is **permissioned** and **permissionless**. Permissionless means that anyone can participate in the blockchain. The nature of permissionless blockchains allowing anyone to participate makes security a priority. It should not be possible to propose a transaction that is not true. Permissionless blockchains usually employ strong consensus models like PoW or PoS or some variation to ensure a proposed transaction's validity. Strong consensus models like these hurt performance, usually giving the blockchain low throughput. Permissioned blockchains are strict on who can enter the blockchain, requiring some for om authentication to enter. Partially knowing the identity of all participants in the blockchain allows for a more lenient consensus. Typically a permissioned blockchain will deal in signatures, checking using Practical byzantine fault tolerance (pBTF). The more lenient security allows for faster throughput, as suggested changes to the chain can be validated fast, and gas cost is unnecessary.

2.2 Reasons for industry using Blockchains

In the last decade, Blockchains have started to become more and more interesting for businesses in their supply chain. In 2018 a survey conducted by Deloitte, asking 1000 corporate executives what makes blockchain interesting for them, 53% of the participants identified the supply chain as the use case their companies are exploring [22]. So what makes blockchains attractive to businesses?

A typical supply chain sees thousands of transactions every day. These transactions usually only involve two parties in a huge supply chain. The transactions are then stored on the databases of the two parties involved in the trade; therefore each of these two links in the supply chain holds its own truth about the product's journey. Each new link in the supply chain then increments the amount of ”truths” leading to inefficiency, errors, delays, and in the worst-case, fraud. [24]. This can be solved with blockchains, as here, everyone who should be in the know can have access to a shared database called a Ledger which holds a single version of the truth. This powerful concept is not the only driving force why supply chains are interested in blockchains.

A recent study done in 2020 [12] looked at what drives companies to choose blockchains for their supply chain, and they identified the six factors given in Table 1 that had a driving impact on the choice.

Factors	Factor Characteristics	Description
Accessibility	• Tracability and Visibility • Identification of issues • Integrity	The Blockchain ledgers with their unique identifier for every asset, make it easy to know where the item is or if the asset has any issues.
Laws and policy	• Laws • Goverment policy	Smart contracts halt the need for humans to perform digital actions, and instead these actions will happen when certain conditions are met. Smart contracts also can meet the stricter privacy laws developing around the world..
Quality	• Quality assurance • Quality fairness	Smart contracts execute all inputs onto the ledger, so the bias and error provided by humans is eliminated. This makes investigations to find weak points in the chain more obvious, exposing corruption or fraud
Data Safety and Decentralization	• Hacking of data • change of data • Controlling-authority • Near impossible loss of data	The Immutability in ledgers makes it near impossible to change data, which first has been placed on the ledger, hacking data becomes harder, as anything the company will deem secret will be only stored as a hash. The relevant data will be off-chain. Loss of data becomes impossible, as removing one ledger does nothing because every ledger is a copy of this ledger.
Data Management	• High quality data • Information flow • Data access control	The ledger which is stored on every node of the blockchain is updated in close to real-time, 100% accurate, and because smart contracts contain all data needed. This means all necessary data is stored on the ledger making predictions on what is happening easy.
Documentation	• Auditable • Accounting • Ecosystem-simplification	Blockchains work on smart contracts, and therefore ensure that what is put on the blockchain is correct. Reducing human errors (no wrong inputs), simplifies auditability, as a clean trail from start of production to sale can be followed easily and makes the whole ecosystem easier to follow.

Table 1: Motivating factors driving industry to adapt blockchains based on findings found in [12]

2.3 Blockchains related to the thesis

2.3.1 Hyperledger Fabric

Hyperledger Fabric is an open-source framework for developing permissioned blockchains by the Linux Foundation, supporting the creation of smart contracts in general-purpose programming languages such as Javascript (Node) and Golang (Go) [25]. The ability to write in already familiar languages has made Hyperledger Fabric a popular choice for blockchain development. It is supported in blockchain development by IBM, Microsoft, SAP, and ORACLE [16].

Making Hyperledger Fabric permissioned makes it a popular choice for businesses that want a certain amount of privacy or an assortment of companies monitoring a supply chain. Hyperledger Fabric relies on the fact that the users in the Blockchain will be known or partially known. This assumption allows them to reduce the security, preferring consensus like Practical Byzantine Fault Tolerance (pBTF) [26]. pBFT can handle there being up to $\frac{1}{3}$ malicious nodes in the Blockchain before the system is compromised. This is significantly weaker than permissionless consensus models. However, this vastly increases the performance of the blockchain and makes payments in the form of gas obsolete. Permissioned blockchains suffer from a more substantial possibility of malicious collaboration, especially in smaller channels.

Hyperledger Fabric works with digital X.509 certificates to validate the users in their blockchain [27]. Without valid certificates, they are not allowed to communicate within the network. These certificates provide some privileges and knowledge about the users. To communicate in a Hyperledger blockchain, you need the correct certificates. Without them, the user loses that privilege. Certificates create a knowledge base inside the blockchain. For a transaction to pass, the user signs the transaction which is later verified by endorsing peers, who sign the transaction as valid. Once the transaction is validated, the proposed inputs are used in the chain code, changing the state of the blockchain [23].

Hyperledger Fabric is a blockchain with multiple different ledgers held by nodes [1]. One node can hold onto numerous different ledgers within the same blockchain. This is done by Hyperledger Fabric, allowing for channels within the blockchain network. In a channel, a subgroup of organizations can communicate using their node without the remaining organizations knowing what is going on. Creating channels within the blockchain is realized by creating a configuration block. The configuration block details who can join and interact and the policies, which define the structure of how decisions are made and specific outcomes are reached. Therefore, all nodes within the channel are bound by the policies created by the configuration block. To identify the nodes of an organization, they must have the appropriate Certificate created by a Certificate Authority (CA). These certificates are then used for identification and endorsement. Organizations tend to prefer using their own certificate, and therefore there will be multiple CA. To handle all these CAs, Hyperledger Fabric also has a Membership Service Provider (MSP), which can identify that nodes indeed were created by a valid organization. The channel configuration can now provide the appropriate rights to the nodes based on the channel policy. For example, some organizations might be allowed to add more organizations to the channel, while others might only be allowed to read what is on the ledger.

A channel will have a distinct ordering service meant to create blocks [28]. It is deterministic, meaning any block validated by the peer is guaranteed to be final and correct. For this to work, a proposed update has to be sent. Then, a subset of the channel peers invoke a smart contract and endorse the results, if satisfactory. The approved proposal is then sent to the ordering service (more than one ordering block). The proposal will be ordered in a defined sequence and packaged into blocks. The orderer will distribute the block created to all peers connected to it using the gossip protocol. Each node will then look at the transaction inside the block and validate them, checking that there has been a correct endorsement and that these endorsements are valid. If a transaction is found to be invalid, the block will remain in the blockchain, but the transaction itself will be marked as invalid, and the ledger's state will not be updated based on this transaction in the block. Currently, the two main ordering services used in Hyperledger Fabric are Raft and Kafka.

Hyperledger Fabric has smart contracts, which they call chain code. The business logic that describes how nodes interact with the ledger is contained in the smart contracts in Hyperledger Fabric. Smart contracts are, therefore, channel-specific in Hyperledger Fabric. The Smart contracts are installed on the relevant nodes in the channel, enabling them to do the desired interactions based on the channel configurations. The most important smart contract in each channel is the endorsement policy, used to validate incoming transactions by the nodes [29].

When peers [30] finally come into the channel, they are all given the ledger for this channel. Each node can be part of multiple channels and therefore can also have multiple ledgers and smart contracts attached to it. The end result might look something like Figure 2.

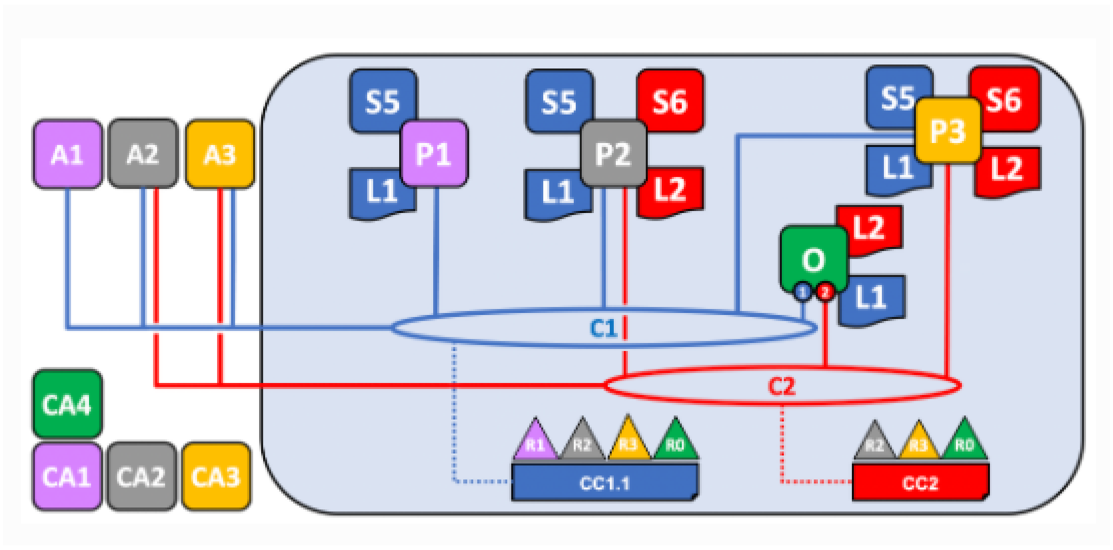


Figure 1: An example structure of how a Hyperledger Fabric implementation might look like. In the picture are two channels C1 and C2 with both their respective configuration blocks CC1.1 and CC2, showing which Organisations can host nodes in the triangles. The nodes themselves have the appropriate ledgers and smart contracts for the different channels, and outside the network square we see the appropriate CAs and applications which can send requests to the blockchain, taken from [1]

2.3.2 Ethereum

Ethereum is a permissionless blockchain allowing anyone to participate, provided they can create an Ethereum wallet. Ethereum was the first blockchain to introduce the concept of smart contracts. Ethereum has a turning complete programing language, allowing anyone who can write in primarily Solidity to create smart contracts on the blockchain.

Ethereum is a popular choice for anything requiering a strong proof While it is still most known for its currency ETH, Ethereum is linked to anything from supply chains to the recent non-fungible tokens trend, anything which requires documentation like audibility is preferred on a permissionless blockchain, as it is harder to tamper with.

Ethereum is a permissionless blockchain and therefore has to assume that there are malicious nodes in the domain. For this reason, Ethereum uses a consensus mechanism called Proof of Work (PoW). A very energy and time-consuming consensus model, which however is very secure. Ethereum sacrifices performance for security.

Ethereum, at its core, is built up of accounts, which can either be an externally owned account (EOA) or a smart contract. Independent of if it's an EOA or smart contract, it has 5 data fields [15].

- Address: a 20-byte address, which is a cryptographic public key, used to identify both Ethereum wallets and Smart contracts uniquely.
- Value: The total sum of Eth held by an account.
- Code: If the account is a smart contract, this will be the code which allows it to complete operations. If this is an EOA, this field is empty.
- Data: If this is a smart contract, this will be the data that is stored on the smart contract (memory). If this is an EOA this is empty.
- Nonce: a counter, which makes sure a transaction can only be processed once, and is incremented for every time the account is used.

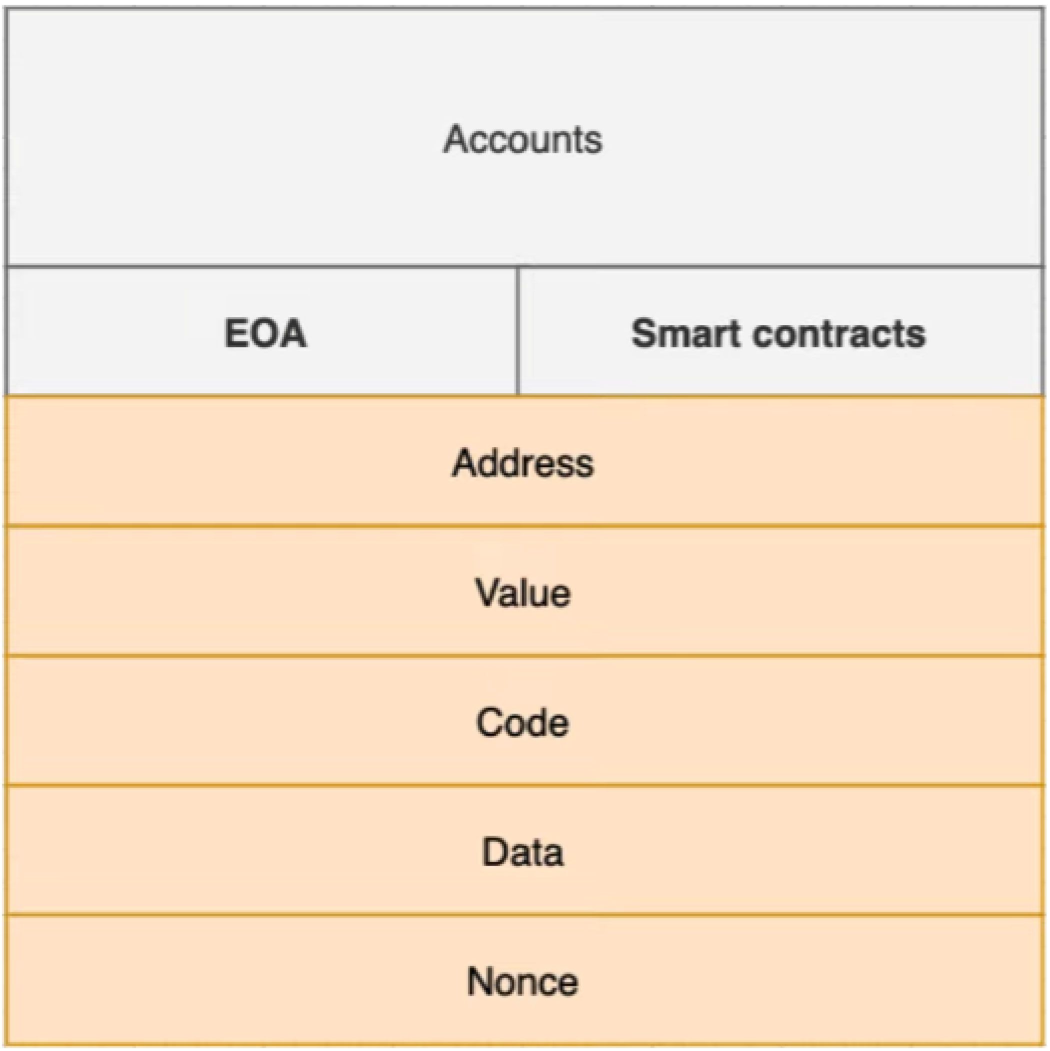


Figure 2: The structure of the ethereum blockchain

Ethereum suffers compared to Hyperledger Fabric in speed, but the permissionless nature of the blockchain makes it ideal for auditing a truth. Hyperledger Fabric, the entity interested in the auditing, must be part of the correct **channel** (not just blockchain) to prove a statement. In Ethereum, a user must only have an account and access to the smart contract in question to prove the statement [15].

2.3.3 Permissionless Vs Permissioned

When choosing between permissioned or permissionless, there are often a few considerations to take into account when selecting which blockchain to use [31], as shown in Table 2

Considerations	Permissionless	Permissioned
Data access	In a permissionless blockchain, anyone can participate, and therefore anyone can, in theory access the information stored	Permissioned blockchains only allow access to those entities who have been accepted into the blockchain
Performance	Permissionless blockchains are generally seen as slow in processing transactions compared to permissioned ones	Permissioned blockchains are considered a lot faster at processing and storing information on the ledger compared to permissionless
Availability	Permissionless blockchains anyone can access and, therefore, are available to everyone in theory.	Permissioned blockchains only allow the entities inside the blockchains to view the information stored
Integrity	Permissionless blockchains are often seen as having better integrity, as their consensus models require a lot more effort to successfully change information	Permissioned blockchains, while secure and immutable, only need a majority of the entities within the significantly smaller blockchain to collude maliciously to change information on the ledger
Cost	Permissionless blockchains work with validators, which must be incentivized, so as not to act maliciously; for this reason, there is a cost associated with placing information on the blockchain	Permissioned blockchains are invite-only, so there is already established a good amount of trust to the users, for this reason, the consensus mechanisms don't require a cost to come to an agreement
Data Protection	Permissionless blockchains need to be very careful about what information is stored on the blockchain. The information is immutable, so mistakingly placing sensitive information on a public blockchain can be detrimental when following laws like GDPR	Permissioned blockchains are invite-only; for this reason, the data set on the blockchain often does not need to consider data protection as much as permissioned
Governance	The blockchain is, in theory, owned by everyone who has a node on the blockchain. In practice, depending on the consensus mechanism, a small to very small number of nodes control the blockchain.	Permissioned blockchains usually make their own consensus rules on how the blockchain should be governed, and an agreement between the parties involved is reached

Table 2: Considerations to make when choosing permissioned vs permissionless blockchains

2.4 Blockchain Interoperability

Interoperability can be classified as: [16].

- The capacity of a computer system to exchange and make use of information
- The capacity to transfer an asset between two or more systems while keeping the state and uniqueness of the asset consistent.

The challenging part is the 2nd part of the definition, as blockchains need to make sure that an asset in one ledger is not duplicated on the other ledger without necessarily being able to check this for themselves.

Typically assets can be placed into three categories. [11, 32]

- Fungible assets (FA): These are assets which can be used interchangeably with another asset of the same type. A typical everyday FA is money. A One Dollar bill can be exchanged with any other One Dollar bill. In the Blockchain world, a FA is typically a cryptocurrency.
- Non-fungible assets (NFA): These are assets that can't be swapped as they are unique, with specific properties. Typical everyday NFA is a concert ticket. They are similar, but one has a completely different seat number than the other, making both unique. In the Blockchain world, this could also be a virtual ticket.
- Data: The final asset type is Data; this can be defined as everything else, anything which can be on two ledgers at once without creating problems. This might, for example, be current electricity prices. Again, having this data in two separate places does not matter.

Challenges arise with FA and NFA. These need to be ensured to only exist in one place at a given time to avoid duplication of assets. The blockchains will either burn or lock this asset on their ledger to solve this issue. Locking means that the blockchain doesn't allow anyone to use this currency until it has been unlocked again. This can happen when a trade is made, and the asset returns in circulation. Burning an asset means that the asset is permanently lost on the one blockchain and can't be retrieved even if the value is traded back.

Typically these assets are swapped through different types of schemes [33]

- **Notary Schemes** are trusted intermediaries; these schemes typically work where two blockchains are connected to a Notary Scheme, which is third-party operated. The Notary usually listens in on events happening on Blockchain A and when an event happens on blockchain B, it claims to blockchain A that this event has taken place. These Schemes rely heavily on trust, where both sides (or only one side) must trust that what the Notary Scheme claims to be true actually is true.

  Notary Schemes can also be used for the transfer of value. Usually FA for other FA [34]. Here a Notary scheme will lock assets on one blockchain and release the assets on the other blockchain. Typically the Notary Scheme will either be centralized or federated, meaning either one party or multiple parties will agree on when to release funds. Both suffer from the trust problem, where a Notary Scheme can run off with all the locked money if they want, less so in a federated on, but if more then 50% are maliciouse the same problem applies. Centralised Schemes also suffer from the single point of failure problem, where if the server running the trusted third party is down the whole system is down.

- **Relays** use smart contracts to read, validate and act upon events on another ledger. This means that instead of trusting what a Notray Scheme says is true, they themselves read what is happening one the other chain. This interoperability requires that

  - 1. Blockchain A has knowledge of what is happening in Blockchain B
  - 2. Smart contracts in blockchain A can somehow read and understand what is happening in Blockchain B, use B's consensus algorithm to verify that this block is valid, and then from this deduce information.

  This is typically done by Both or one chain having some light version of the blockchain they are reading on their blockchain. This is generally achieved by a lightweight node [35] having the other blockchains header downloaded and using Merkel trees to get the desired proof [34, 36]. This also means that one the side feeding information, there must be a node on block B that sends this information to block A. Relays Typically work either one way or two way pegged.

  - **One-Way relays** In a one way realy, Blockchain A can read from ledger B, but not the other way, this is usually because one of the ledgers is not able to understand their consensus algorithm.
  - **Two way relays** Both relays know about each other and can read each other's ledgers

- **Hash-Locking** Hashed Time Locks or Atomic Swaps [37, 38, 18]: Users on different chains agree with each other on soemthing they want to exchange, and create smart contracts with a hashlock and a timelock securing them from maliciouse behaviour. Everyone involved first sends out their contracts, showing a willingness to part from a certain amount of assets/goods on their blockchain. When all parties have done this on their respective blockchains, they then send out their secrets, releasing their ownership from their assets and claiming their new assets in another chain and then release their secrets on their respective blockchains. This means instead of needing a partial copy of the ledger like in Relays, all that is needed between two Blockchains that want to switch assets using Hash Locking is a single hash.

For interoperability between blockchains, there currently exist two types of interoperability defined by the world economic forum [16].

- homogenous (compatible blockchain platform) have the same platform logic, consisting of consensus mechanisms, smart contracts and authentication, and authorization¨
- heterogenous (non-compatible blockchain platforms) have different platform logic, consisting of consensus mechanisms, smart contracts, and authentication and authorization

2.5 Self Soveregin Identities

When the internet was first created, it was built without a layer to authenticate identities. However, people are not endpoints in a network, and therefore, there is no way to identify people uniquely.

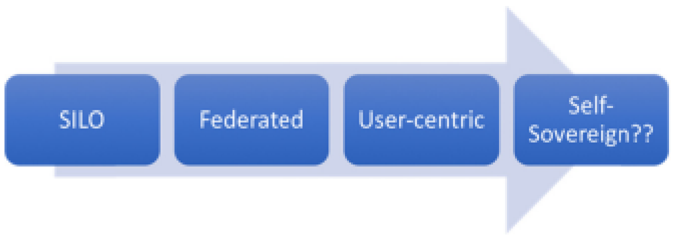


Figure 3: The evolution of identity managment [2]

The simplest form of identity management is Isolated User Identity. There are only two parties involved, a service provider and users. The service provider provides access to their domain, but the credentials are not valid in any other domain providing services.

The next step is a federated model. The Identity provider works together with one or more service providers. The Service provider then relies on the Identity provider to issue the necessary credentials to the Service provider for authentication. Once the Identity Provider registers a user, the user can access all service providers who know the Identity provider. A typical example is universities, which issue an Identity, which works on multiple platforms that they have agreements with.

The user-centric model is similar to the federated model, however here there does not have to be any trust between the identity provider and the service provider. The Identity Provider will simply release the necessary information which the Service Provider requires to authenticate users. Common examples are Facebook and Google. This solution improves usability but gives huge power and centralization to a few select businesses, giving these companies a complete view of the digital footprint of their users. These companies could now also deny access to users they deemed undesirable to a vaste amount of services. [39].

The next step in the user identity is Self-Sovereign Identity. The individual's digital existence is independent of any single organization. Nobody can take your Self-sovereign identity away from you.

Self-sovereign identity (SSI) gives individuals control over their own identity to decide how their personal information is shared and accessed, enabling trusted interactions while preserving privacy. [40].

SSI is a very new concept and therefore goes by many names. For this thesis, It will therefore be set like this. The Self-sovereign identity (also known as decentralized identity, personal identity, or distributed identity) is the total of all the information available to be shared. The sharing of information is accomplished using decentralized identifiers (DIDs).

Decentralized identifiers or certificates can be seen as holding some truth about a person or object, like where that person went to university, and which degree that person achieved; these identities should exist somewhere with the public key of the entity owning the DID so that entities interested in verifying the information can do so out requiring third parties [41] [42].

In an SSI setup, there are three key parties that must work together to enable SSI to work: Issuer, Holder, and Verifier. [43, 44, 3, 45].

- Holders: Are individuals who store data in a digital wallet, typically using blockchain technology, to store the total of all the credentials received creating their SSI. **The Holder is the only entity allowed inside the blockchain to share their credentials**.
- Issuers: Are credential creators. This entity creates credentials for Holders, which are known to them. Usually, these should be big trusted organizations like universities, hospitals, or governments, which know information about a holder, which can be of interest to other entities. It is important to note that although they make the credentials for a holder **the Issuer can't share it with anyone besides the holder** meaning information can't be leaked by an Issuer.
- Verifiers: Are businesses or individuals that need to confirm something about someone. This is achieved by proposing what information is required from the Holder to validate their requirements. Requirement can be anything from age restriction to a valid passport, which then, through the use of one or multiple credentials, can be verified.

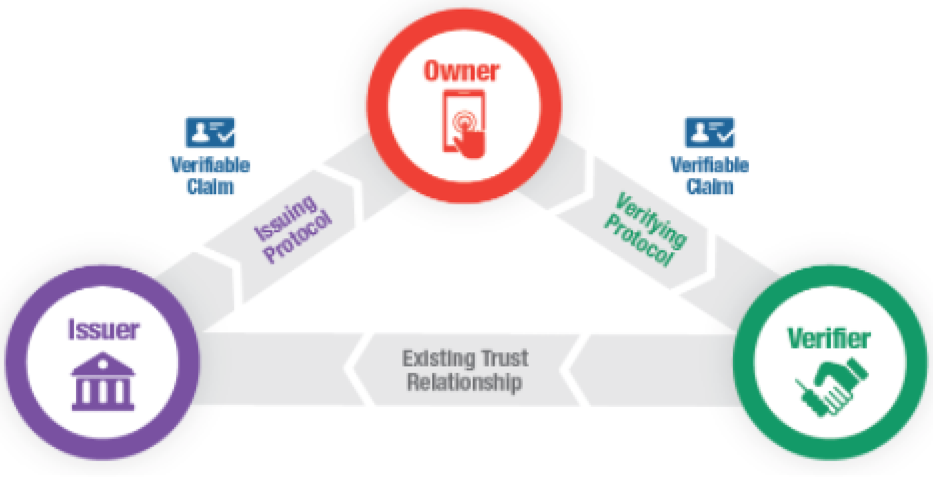


Figure 4: The desired communication between Issuer, Holder (Owner in this image) and Verifiers [3]

There is a fair amount of literature about the requirements needed for an SSI [46, 2, 47]. However, most of them derive or at least use Christopher Allen's Ten Principles of Self-Sovereign Identity [48] as a basis. These ten principles are:

- Existence: Users must have an independent existence. An SSI can't digitalize the whole truth about a person but should make publicly accessible some information about the user.
- Control: Users must control their identities. The user is the ultimate authority on their identity and should be able to do with it as they please.
- Access: Users must have access to their own data. A user must always easily be able to retrieve all the claims made about them.
- Transparency: Systems and algorithms must be transparent. The systems used to administer and operate a network of identities must be open, both in how they function and in how they are managed and updated
- Persistence: Identities must be long-lived. The identities should preferably exist as long as the user needs them. However, the persistence should not conflict with the right to be forgotten; if the user wishes, the information should either be updated or deleted.
- Portability: Information and services about identity must be transportable. A third-party entity should not hold identities. Preferably

the information should be transportable and recoverable.

- Interoperability: Identities should be as widely usable as possible. The goal should be to make identities usable in as many situations as possible.
- Consent: Users must agree to the use of their identity. Sharing of data should only happen at the behest of the user.
- Minimalization: Disclosure of claims must be minimized. When data is disclosed, that disclosure should involve the minimum amount of data necessary to accomplish the task at hand.
- Protection: The rights of users must be protected. When there is a conflict between the needs of the identity network and the rights of individual users, then the network should err on the side of preserving the freedoms and rights of the individuals over the needs of the network.

A blockchain exhibits several properties which can aid in achieving the requirements set by Allen and is therefore also an attractive option for SSI to use as a basis technology for the solution [2].

- **Distributed consensus** Blockchain's ability to gain a distributed consensus on the state enables the verifiability of information by any authorized entities. This enables **transparency**, everyone on the network knows the consensus method and to some extent, can participate in the authorization of the data saved.
- **Immutability** Achieving a distributed consensus with the participation of a large number of nodes ensures that the ledger state becomes practically immutable and irreversible after a certain period, satisfying the **Persistence** and **Existance** requirement
- **Data persistance** Data on a distributed ledger can't be deleted. The distributed fashion ensures that as long as a single node in the blockchain exists, the data persists. This aids in the **Persistance**, **Existance** and **Access** requirements.
- **Data provenance** The data storage process in any distributed ledger is facilitated by a transaction. Every transaction needs to be digitally signed to ensure the authenticity of the source of data, this means anyone participating in the ledger has a key. Every user having a key enables the user to **control** signing their information with their unique public key and the possibility of **existance**, linking a user's information to some cryptographic authentication already used in the blockchain. This gives users **control** of their data, as they are the only ones with the private key to verify their information, giving them the ability to **consent** to what information they want to share, as they need to use their key-pair to verify it, enabling **minimalization** as verifiers can't get too greedy in what information they can ask for.

- **Distributed data control** The ledger ensures that there are always multiple nodes that can provide the information, securing no single point of failure. This secures **access** for users' information on the blockchain.
- **Accountability and transparancy** Every single transaction can be verified by an authorized entity, promoting accountability and transparency. This, to some extent, helps the users with **protection**. However, it is not a solve-all solution to the problem.

3 Related Work

A fair amount of research has been done on blockchain interoperability. The most significant contributor to gathering this research is a systematic literature review, which has done an excellent job surveying all existing methods from 2014 to August 2020 [49]. This paper does present solutions which it claims can interoperate data heterogeneously. Unfortunately, their definition of heterogeneous **is a transaction between different blockchains**, meaning any two blockchains which can communicate data are considered heterogeneous. While our definition is a lot stricter and more in line with the world economic forums definition [16]. They define heterogeneous as a difference in platform layer, meaning consensus algorithm, smart contracts, and authentication method, which essentially means two different types of blockchains. Unfortunately, their suggestion for interoperating this data is using an API. In our pre-study, we looked for interoperability solutions that would satisfy the research question:

What are the current interoperability methods/technologies which might be used to interoperate SCMs.

Five solutions were found which could interoperate either data, FA or NFA between blockchains. These were Polkadot [20], Hedera [17], Interledger [18], Abebe Relay [10] and SCIP [9]

Polkadot [20] only provides interoperability between substrate blockchains [50]. There are mentions of bridges. However, these are still in development [20]. Hedera seems only to be a solution for Hyperledger [17]. It has developed the HashPort [21], which enables interoperability between Hypereldger and Ethereum, however only for FA [51]. Intereldger was developed to interoperate between heterogenous blockchains, however, only for FA [18]. Abebe was developed to interoperate data between two Hyperledger Fabric blockchains, limiting itself by requiring all the certificates on both blockchains for secure validation [10]. SCIP is able to interoperate data betwenn Ethereum and Hypereldger Fabric [9]. This shows that currently, the biggest focus on interoperability is homogenous or focused on FA.

We did not consider data interoperability between heterogeneous blockchains during the pre-study when discussing them. To our knowledge there is no literature review or study conducted to highlight heterogeneous interoperability solutions.

The two solutions found to satisfy the first research question, BitxHub [19] and SCIP [9] both don't provide any information about their solution's performance. Polkadot has been measured to at least 10 000 tps [49], while Hedera talks about 250 000 tps with 100-byte transactions [52]. Both these solutions, however, discuss homogenous performance. There is a need for this information in heterogeneous interoperability, too, in order to better inform users about the capabilities.

Self-sovereign identities are very new, and therefore most of the articles focus on either of the three topics. The formalization of a definition of what Self-sovereign identities require to function as intended [53, 54], justification of why Self-sovereign identities are needed in the current day [47, 55, 53], and what needs to be improved about self-sovereign identities for them to become mainstream [53].

There are very few use cases for Self-sovereign identities in the literature. Most cases use SSI to digitalize existing identities or justify the need for SSI in certain fields [56]. However, there do exist a fair amount of articles on the internet presenting use cases, often by companies developing SSI, like Cheqd [57] and Adnovum [58]. To the knowledge of this reader, there are no articles about leveraging Self-sovereign identities as a method to authenticate information sent over two heterogeneous blockchains.

4 Research Design and Implementation

This chapter will explain our research approach and implementation. In section subsection 4.1 we present the motivation for our project, and in subsection 4.2 we give the three research questions yielded from this motivation. Finally, in subsection 4.3 we present the methods and design we choose to follow for each research question.

4.1 Research Motivation

Current supply chains typically have thousands of transactions happening every day between multiple vendors in the chain, all processing and saving transactions in their blockchain. However, communication between the different blockchains for supply chains remains fragmented. From the pre-study section 8, it was found that a vast majority of SCM adopt either Hyperledger or Ethereum. To enable communication between these fragmented ecosystems, Interoperability is needed. Interoperability can be defined as [10]:

"the semantic dependence between distinct ledgers for the purpose of transferring or exchanging data or value, with assurances of validity or verifiability"

From the results gathered in the pre-study, there was a clear tendency to favor interoperability solutions that only work for a single blockchain (homogeneous). In order to tackle the fragmented blockchain ecosystem, a greater focus must be placed on interoperability between non-compatible blockchains platforms (heterogeneous). Linking blockchains of compatible nature does not fully solve the fragmentation problem unless one blockchain type becomes the standard; however, this is unlikely with the distinct advantages granted by permissioned and permissionless blockchains. Therefore, we decided to determine from the blockchains researched in the pre-study and this thesis which one would best be able to interoperate data between two heterogeneous blockchains, one permissioned and one permissionless.

Further, a greater insight into the problems and performance should be made, as currently, no literature exists on this topic for heterogeneous blockchains. For any supply chain system, the throughput of information is vital. In a typical supply chain, there will be a lot of data processed to update all the moving pieces constantly occurring in the supply chain. Therefore, it is desirable for the information transferred between the supply chains also to have a good performance. However, no information about this issue exists and therefore needs to be highlighted to inform users of current capabilities and further developmental priorities.

From the research done on BitxHub and SCIP, it was discovered that heterogeneous blockchains have a problem with authenticating data when it is transferred from one blockchain to another. Information gets signed by the respective cross-chain getaway, anonymizing the information, as it can't be traced back to the creator of the data on the other blockchains. Self-Sovereign Identities (SSI) might be able to solve some of these issues related to the Identity of who placed the data. If this worked, it would enhance audibility in the blockchains, as information transferred would have a verifiable origin, and might enhance security, as data sent over the cross-chain gateway could be authenticated to see if it comes from a trusted source.

4.2 Research Questions

From the research motivation mentioned in subsection 4.1, we formulated three research questions. The three research questions are as follows:

RQ1: Is there a way to interoperate data between two heterogeneous blockchains?

RQ2: What is the performance, and what are the issues of the selected interoperability solution?

RQ3: Can Self-sovereign identities be used to deanonymize cross-chain data between two heterogeneous blockchains?

4.3 Research Method and Design

This section will summarize the research method and design for our study. A literature review was conducted in our pre-study to find existing interoperability solutions. The thesis will primarily focus on designing artifacts capable of solving our three research questions. This sets us into the design science and behavioral science paradigm, which is argued to go hand in hand by Alan Hevner [59].

Design science is the scientific study and creation of artifacts as they are developed and used by people with the goal of solving practical problems of general interest.[60]

The literature review will, together with the solution found in this thesis, form the basis for which interoperability solution can interoperate data between two heterogeneous blockchains. The research here is the behavioral science paradigm of finding the truth, which motivates the artifact's creation. Once the solution has been found, the artifact's creation can begin. Research question one will mainly focus on design as a process, where the solution is built and evaluated until a satisfactory artifact is created.

Research question two will then analytically evaluate the created artifact, studying the artifact in use for dynamic qualities [59]. In our case, the quality focused on is performance, as research has not been conducted on heterogeneous interoperability performance.

Research question three is motivated by the observation that data sent over the heterogeneous interoperability solution gets anonymized. This question will focus on design as a process, building and evaluating the design until it can fulfill our goal of deanonymizing the data sent over the interoperability solution. Once the artifact is created, an experimental design evaluation method will be used. Finally, a controlled experiment will be conducted studying the artifact for qualities, in our case, usability.

5 Results

This section will present the results found for our three research questions.

For our first research question: **Is there a way to interoperate data between two heterogeneous blockchains?**. We will first, in subsection 5.1 present the interoperability solution found in this thesis, BitxHub. We will then, in subsection 5.2 attempt to find the solutions able to interoperate data between heterogeneous blockchains and justify our interoperability choice. Finally, in subsection 5.3 we will use the found interoperability choice to build a functioning solution capable of interoperating data between Hyperledger 2.3 and BitxHub.

For our second research question: **What is the performance, and what are the issues of the selected interoperability solution?** We will in subsection 5.4 present the experiments and results related to performance, as well as presenting the issues observed when using our developed interoperability solution.

For our third research question: **Can Self-sovereign identities be used to deanonymize cross-chain data between two heterogeneous blockchains?**. First, in subsection 5.5 we will justify our choice to use Hypereldger Indy as our platform for the SSI solution. Finally, in subsection 5.6 we will present the whole creation process from preparation to working solution. After that, we ultimately present the experiment used to prove that SSI can deanonymize cross-chain data between two heterogeneous blockchains.

5.1 Bitxhhub

This section will present another interoperability solution found during the master, claiming it can interoperate heterogeneous data. It will be shown in the same way as the findings of the pre-study in section 8, to maintain the consistency of how interoperability solutions are presented in this thesis. The results shown here will subsequently be used in subsection 5.2 to find the best interoperability solution between heterogeneous blockchains.

5.1.1 Architecture

BitxHub is, as of 2020, an open source solution [61], which aims to interoperate all asset types Data, Fungiable Assets, and Non-fungible assets [19].

BitxHub consists of three core parts, and the Inter-Blockchain Transfer Protocol (IBTP) their standardized communication protocol, which enables the sharing of information over the heterogeneous blockchains [61, 19].

The three core parts of BitxHub are

- **App-chain:** The App-chains are simply existing fully functioning blockchains, which, together with the Cross-chain gateway and Relay chain, can achieve interoperability between each other. Officially BitxHub claims that it can communicate between **Hyperledger Fabric (1.4), Ethereum, BCOS, CITA, and Hyperchain**.

- **Cross-chain gateway:** The Cros-Chain gateway is responsible for collecting and broadcasting IBTP transactions [62]. The validation rules should be set once a majority of peers inside the Relay-Chain have accepted the Cross-chain gateways proposal to join the channel, but before it connects to the Relay-Chain. These Rules describe how the IBTP information should be handled for that particular App-chain [63]. Ideally, the Cross-chain gateway should be managed by the same people managing the App-chain. This ensures that there are no malicious motives when managing the Cross-chain gateway. [61].

- **Relay-chain:** This is a permissioned blockchain responsible for routing IBTP information from one App-chain to another. Like any blockchain, it has its own peers, which are responsible for all the operations, as well as keeping the ledger. For an App-chain to connect to the Relay-chain, the Cross-Chain Gateway connected to the App-chain must receive a majority vote from all the peers operating inside the App-chain. Only once a majority has accepted the proposal to join can the Cross-Chain Gateway (and by proxy the App-Chain) successfully connect to the Relay-Chain [62]. After the successful registration, but before the actual connection, the validation rules must be sent. The validation rules are inside a script, which will be used to validate the Proof of the transaction and ensure validity [19]

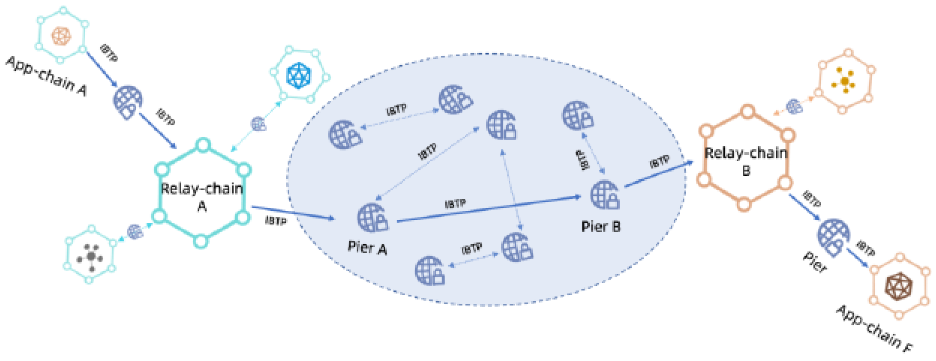
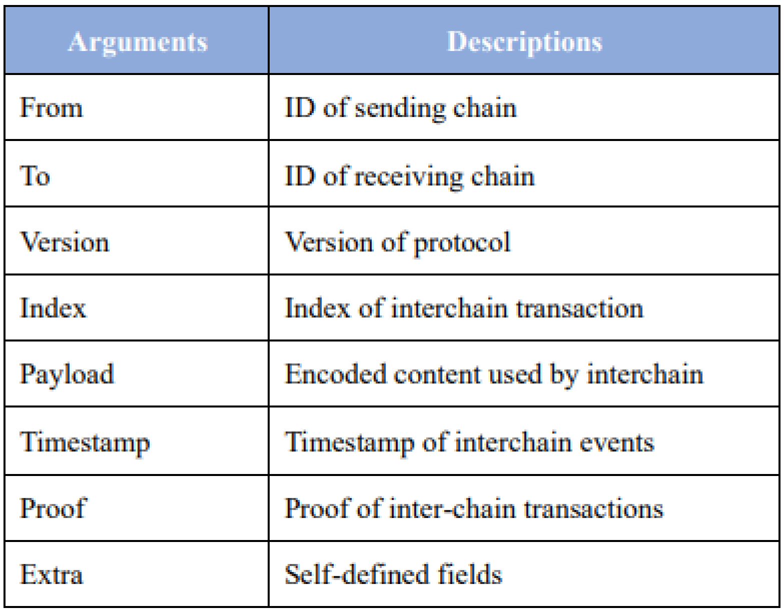


Figure 5: The BitxHub architecture, with different App-Chains (Blockchains), Relay-Chains for intercomunication, and Cross-chain gateways (in image called Peers) for setting up the IBTP between sender and reciever

BitxHub has a goal to interoperate with as many heterogeneous blockchains as possible. All blockchains have to some degree different structures, which makes it impossible to communicate directly between two heterogeneous blockchains. For this reason, BitxHub created their cross-chain transfer protocol, which abstracts the information of the blockchain into something which is readable for the cross-chain gateways, which then can make it readable for their underlying blockchain [19, 61].

The transfer protocol is called Inter-Blockchain Transfer Protocol (IBTP), and consists of the following data fields shown in Figure 6



Arguments	Descriptions
From	ID of sending chain
To	ID of receiving chain
Version	Version of protocol
Index	Index of interchain transaction
Payload	Encoded content used by interchain
Timestamp	Timestamp of interchain events
Proof	Proof of inter-chain transactions
Extra	Self-defined fields

Figure 6: The IBTP Data Structure

The values in the fields are as followes [19, 61]:

- **From:** Is the address of the Cross-chain gateway, which is sending the information
- **To:** Is the address of the Cross-chain gateway, which the information should be sent to
- **Version:** Is the version of the Cross-chain gateway, both Cross-chain gateways need to have the same version to communicate
- **Index:** Is an increasing number used to order interchain transactions.
- **Payload:** Is the encrypted information which should be sent from one App-chain to another.
- **Timestamp:** is a timestamp of the transaction.
- **Proof:** Holds the validation rules, which need to be validated by the relay chain. The validation rules will be different according to the characteristics of the app-chain that the Cross-chain gateway belongs to. A permissioned blockchain can often instantly validate a block, while a

permissionless blockchain needs a certain length before it can be seen as secured on the blockchain. The proof of the protocol is determined by the type of App-chain (Is it fabric or Ethereum). However, the rules for validating the proof can be written by a programmer before it has been accepted into the relay chain [19, 63, 62].

- **Extra:** This field is an additional field which can be used to provide some flexibility inside the protocol.

5.1.2 Consensus model

The IBTP protocol is the consensus model used by BitxHub and enables Cross-chain gateways to translate information to IBTP and blockchain information to IBTP. It holds all the critical information needed for the Relay chain to send it from one Gateway to another and for gateways to validate the data. [19, 61, 63]. Typically the most vital payload is the proof. The proof is signatures from the App-chain/Gateway that the info is indeed on the App-chain and has not been tampered.

For App-chains with probabilistic finality (like Ethereum), the sending gateway itself is responsible for signing that the information sent exists on the blockchain in question [19]

For App-chains with absolute finality (Hyperledger Fabric), the peers inside the blockchain themselves can sign the information as valid and existing and then send it as proof to the Cross-chain gateway. [19].

5.1.3 Constraints

The validity security in a permissionless blockchain appears weak. The fact that a single cross-chain gateway can be responsible for signing the existence as valid means it easily could act maliciously. The Whitepaper mentions incentives, and cross-chain clusters [19]; however there does not seem to be any incentives provided in their solution or a method to connect multiple Cross-chain gateways together. This means that the whole system needs a strong trust in that the cross-chain gateway doesn't act maliciously.

Having a single cross-chain gateway for each app-chain to relay chain connection can provide a single point of failure on the system, or can be a bottleneck in high throughput situations [19, 62]

5.1.4 Interoperability

BitxHub uses the IBTP protocol to ensure a single format for communication through their relay chain, to allow multiple heterogeneous blockchains to communicate with each other. This is enabled through using cross-chain gateways, which take information from the App-chain and translate it into the

IBTP protocol format, making it readable for the Relay-chain. The Relay-chain later sends it to the destination cross-chain gateway to be translated into information readable to that app-chain. [19]. BitxHub claims a single relay chain can hold at most 64 different App-chains, of any of the supported app-chains, making it very scalable [19]

5.1.5 Pros and Cons

Table 3: Pros and Cons BitxHub.

PROS	**CONS**
Security	
• The IBTP consensus ensures that nothing will be tempered with from when it was sent from App-chain A to App-chain B. • Only smart contracts which are validated by the broker contract to interact in the BitxHub system are allowed to communicate over BitxHub. • Validation rules are customisable, meaning an entity can either increase security if they see a need for it. • The Relay-chain is permissioned, meaning the nodes inside the Relay-chain must come to a majority consensus that a new App-chain can join • The validation rules are customizable for each owner of the Cross-chain gateway enabeling every App-chain to specifiy their own requirements for a consensus.	• The cross-chain gateway consists of a single node, this means that it is a single point of failure, if down, communication is impossible and high trust needs to be places in the peer, so no censoring attacks occure. • Getting the Cross-chain gateway to sign information for a probabilistic finality blockchain means that node has 100% controll over the validity of the information. BitxHub mentiones in the whitepaper [19] a cluster of nodes, but at the moment this does noe seem possible to make
Performance	

• BitxHub in their whitepaper mentiones high perform, but does not mention how this is achieved or has any measurable results [19]. For this reason this claim has to proved.	• The cross chain gateway is as mentioned a single node, meaning it needs to translate in and outcoming information from the valid blockchain protocol to the IBTP protocol, making it a bottleneck for performance.
Scalability	
• BitxHub has a highly scalable capability. every relay chain can in theory support 64 app-chains [19] • Thanks to the IBTP protocol, any blockchain cabaple of adapting this protocol can communicate in Bitxhub. • BitxHub is open source, so anyone can expand on and make BitxHub interoperable with more blocckhains.	• Interoperating blockchains must have prior knowledge about the existing smart contracts in the other app-chain, and how to access them. • Interoperating blockcahins must have knowledge about the chain-ID of the other blockcahin
Costs	
• There is no monetary cost associated with using BitxHub	•

5.2 Heterogeneos Interoperability

This part will be based on the literature study presented in 8 and the later found solution BitxHub 5.1 justifying the choice of the interoperability solution used to interoperate data between two heterogeneous blockchains. Subsequently, the solution chosen from this section will motivate the next subsection 5.3 which will describe the development of the chosen solution.

Data is the simplest form of information that can be transferred and needs the least amount of security compared to FA and NFA. However, in industry, this is by far the biggest information flow and needs to be addressed before the other two can be considered. The evaluation is further confined to requiring one blockchain to be permissioned and the other permissionless. This is to understand better how two very different consensus models create challenges for interoperability and to enable an ecosystem where the users can reap the benefits from both permissioned and permissionless blockchains. The selected solution must also have the potential for future development, as interoperability between blockchains is very new, and being able to adapt to state of the art is essential.

The choice was based on some critical considerations created in the pre-study when choosing which interoperability method to pursue. These are: **Interoperability, Consensus Model, Security, Performance, Cost, and Scalability**

5.2.1 Interoperability for heterogenous data transfer

For Data to be transferred from one blockchain to another, there needs to be satisfactory proof that the information being transferred is valid and is on the other blockchain. The following interoperability methods have succeeded in this requirement.

- SCIP uses its SCL to provide an URL identification method to contact gateways, which are responsible for translating the uniform protocol from the sender blockchain to the receiver blockchain. The information will be returned to the sender with a degree of confidence between 0 and 1, representing the likelihood that the information is on the chain.
- BitxHub uses smart contracts to fetch information from the ledger and send it to the Cross-chain gateway, translating it to the IBTP. The returned proof will differ depending on whether it's probabilistic or absolute finality. Probabilistic chains don't use signatures and therefore need to be signed by the cross-chain gateway, while absolute finality uses signatures on the chain, collected as proof.

The interoperability solutions which do not fulfill the requirements are:

- Abebe Relay uses smart contracts and their relay component inside the blockchain to allow communication. The three contracts will communicate together to get the signatures needed to validate the information on the other blockchain. The data will be sent to the relay, translating the data to a network-neutral language using Google's Protocol Buffers. Finally, the returned value will be delivered with an agreed-upon verification policy. However, this solution was designed for permissioned blockchains, where nodes sign the information needed so that it can be validated on the other blockchain. The consensus model of Abebe requires that all the certificates used in Blockchain A also exist on blockchain B for validation. Permissionless blockchains that do not have certificates to sign the data sent over the cross-chain make this method impossible to use.
- Interledger uses its protocol layers to come to an agreement on what is to be exchanged, the amount, and how to communicate. This is enabled using connectors, which have currency accounts on two blockchains, enabling a bridge between two blockchains. Once enough bridges have

been created to reach the desired currency, the protocol layers initiate the exchange and fulfill command is initiated between all the connectors. The currency is transferred from one blockchain to another, with a small fee paid to the connectors. The solution is very successful for Fungible Assets. For Data, connectors would have no or little incentive to convey the data.

- Polkadot's relay chain is developed with substrate blockchains in mind. To interoperate with other blockchains which are not a substrate, a bridge must be developed. There is a promising solution to an Ethereum bridge in their 2020 whitepaper [20]; however, a bridge would need to be developed for every kind of blockchain not being a substrate, making it too work-intensive.
- Hedera uses tokenization and token contracts, which can require proof in the form of signatures that the information given is authentic. This works exceptionally well in permissioned chains, where a signature system is used. However, for probabilistic consensus without signatures, Hedera falls short. Hedera created Hashport [21], which allows interoperability of digital assets between Hedera and Ethereum, essentially allowing it between Hyperledger and Ethereum. However, the current Hashport setup requires a 0.5% fee of the total transaction [51]. Since data has no value the Hashport is currently only applicable for FA. For Hedera to become desirable, the Hashport, would need to also exchange data.

For this reason, Interledger, Polkadot, Abebe Relay, and Hedera will no longer be considered. They do not meet the desired requirement to interoperate data between the two heterogeneous blockchains described.

5.2.2 Consensus model evaluation

The consensus model must be able to function with multiple different blockchains without there being a need to change the blockchain architecture or the interoperability architecture itself significantly. For example, suppose a solution requires massive changes in the solution's underlying architecture for each new blockchain. In that case, it is not scalable enough in the long run.

Of the remaining two options Bitxhub and SCIP, both methods pass this requirement.

- BitxHub's consensus model is based on the IBTP protocol and is translated and validated on the Cross-chain gateway. For this reason, what is needed is that the information sent to the IBTP has to satisfy the protocol standard. The Cross-chain gateway needs to be able to read the information and translate it to the IBTP protocol. The appropriate smart contracts that can communicate with the gateway will also be required, and validation rules for the blockchain will be necessary. The

Relay chain only operates with IBTP messages and does not care about the architecture of any of the App-chains involved and would not need to be changed.

- SCIP, the consensus model, is based on the SCIP, reached through the SCL through HTTP POST messages. For a new blockchain to be added, the necessary smart contracts would need to be disclosed, and the JSON schema used to handle translations between contracts would need to be updated. To our knowledge, based on the literature presented, no significant changes would need to be made in the SCIP or the SCL

5.2.3 Security related to interoperability

An adequate security level needs to be present in the solution. By nature, data requires the least proof to maintain the state and uniqueness of the asset, as it can exist on multiple ledgers at once. However, the solution must still prove that the data exists on the other blockchain.

- BitxHub provides customizable security, leaving it up to the cross-chain gateway admin to provide the security level desired by the blockchain. The Relay-chain is a permissioned blockchain, requiring a majority vote by the nodes in the Relay-chain to allow a new App-chain to join the ecosystem. The Relay chain ensures no tampering during the transit. For permissionless blockchains, the smart contract the Broker implemented will ensure that only contracts validated to access the Cross-chain gateway are allowed to access it.
- SCIP does not focus much on security. It manages to fulfill the requirements of being able to prove the data exists on the other blockchain, by returning a degree of confidence that it has been placed there. It also gets signed by the gateway that this information is correct. However other then this there is little to no security. Users can controll what they wish to expose, by writing the smart contracts they wish to expose on their SCIP gateway.

5.2.4 Performance

Both solutions do not disclose any factual information about the performance of the interoperability methods. Therefore using our chosen solution will be tested and discussed. Furthermore, both have single nodes in the SCIP gateway and the Cross-chain gateway, which can become problematic as they can be a bottleneck for throughput.

5.2.5 Cost

For a sufficient amount of data to be transferred from one blockchain to another, the cost of sending data should not be a significant hindrance.

Both SCIP and BitxHub are here very similar, therefore they will be addressed simultaneously. For both solutions, there are no costs associated with the cross-chain transfer of data. In the permissioned blockchains, with their consensus model, there are no fees to add information to the ledger, therefore storing information has no associated cost associated. However for the permissionless blockchain, with their requirement to provide a gas fee to place data on the ledger there is a small cost involved. Both solve this issue by making the host of the respective gateways deal with the cost. This could become a problem if the ecosystems become big. Still, with both having some form of privacy involved in their interoperability method, both should, in theory, know who is using their services. Payment options could be discussed off-chain.

5.2.6 Scalability

Both solutions have an excellent potential to incorporate more blockchains into their ecosystem. There is an upper limit of 64 App-chains inside a Relay-chain for BitxHub, while there does not seem to be an upper limit for Blockchains inside SCIP. However, the SCL would have to be updated with the new gateways in the ecosystem. Based on what was discussed for the consensus model, while it would require work, both solutions are adept at incorporating more blockchains. This is mainly because both abstract the blockchain logic in the SCIP and IBTP, making the communication homogenous. However, both have issues with their gateways. The Cross-chain gateway is currently only a single node, making it vulnerable to security and performance issues. The same problem exists for the SCIP Gateway. Both mention this issue in their respective papers this issue but don't seem to have a solution for it.

SCIP and BitxHub would be potential solutions for heterogeneous interoperability. However, BitxHub has a distinct advantage over SCIP.

The SCIP solution was developed in 2020, as part of a Ph.D. thesis by Ghareeb Falazi at the University of Stuttgart. Looking at the Github page given in the SCIP paper [9], it seems the last development on the project was two years ago, and no new articles have been published by Ghareeb Falazi on this topic according to Google Scholar [64]

BitxHub, on the other hand, has since 2020 been an open-source project. It has an active community developing solutions on it. It also has an active community on WeChat where users can ask questions answered by developers. BitxHub will see further development in the future, ensuring the solution's longevity.

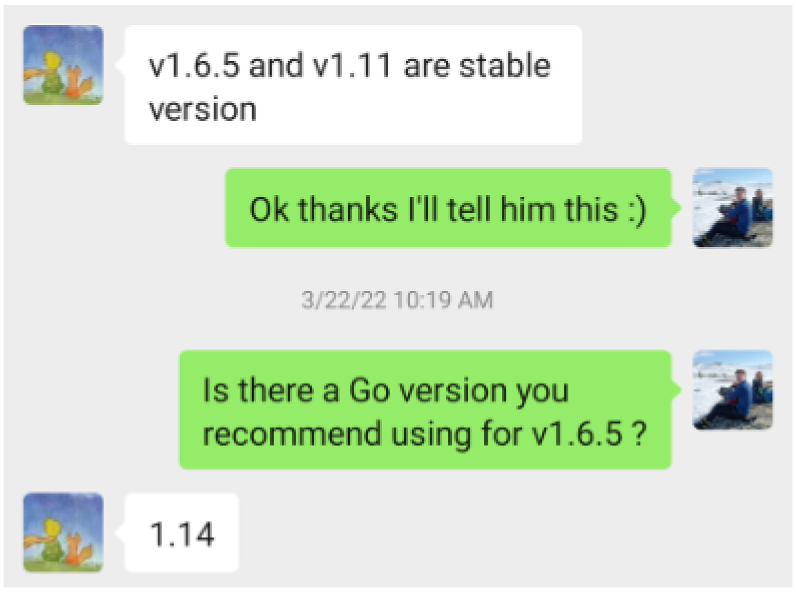
Table 4 provides an overview of what the different interoperability solutions pass on, shown with a checkmark, and what the fail on shown with a blank. The Support row is left open to indicate that this is not a requirement, but for the longevity and relevance of the project in the future, it is nice to have.

Table 4: The different interoperability solutions and what they would pass on for heterogenous interoperability

	BitxHub	SCIP	Abebe	Interledger	Polkadot	Hedera
Interoperability	✓	✓				
Consensus Model	✓	✓	✓	✓		
Security	✓	✓	✓	✓	✓	✓
Performance	Unknown	Unknown	Unknown	✓	✓	✓
Scalability	✓	✓	✓	✓		
Support	✓			✓	✓	✓

5.3 BitxHub development

The Installation of the necessary dependencies was done following the BitxHub environment preparations. For the project itself, it was decided to use BitxHub Version 1.6.5, and Golang V1.14.7. This is because BitxHub Version 1.6.5 is a stable version, and Golang V1.14.7 was recommended to be used with this particular version. This information is not noted on their documentation but was retrieved by asking one of the developers on WeChat as shown in Figure 16.


v1.6.5 and v1.11 are stable version

Ok thanks I'll tell him this :)

3/22/22 10:19 AM

Is there a Go version you recommend using for v1.6.5 ?

1.14

Figure 7: Conversation which lead to the use of BitxHub V1.6.5 and Golang

We chose to interoperate data between Hyperledger Fabric 2.3 and BitxHub, this would satisfy the heterogeneous requirement, but would also use the current most popular blockchains in SCM, as discovered during the pre-study in section 9

5.3.1 Relay Chain

The Relay Chain was made and built following the documentation on Relay chain deployment provided by BitxHub [65]. and it was chosen to use the Solo mode with four nodes in the Relay Chain. Once the files are made, there were no further changes that needed to be made.

5.3.2 Hyperledger Fabric Cross-chain creation

The Hyperledger Fabric 2.3 Cross-chain gateway was developed by a our colleague HaoMing Li, who changed the code in Fabric 1.4 retrieved from the Cross-chain gateway deployment by BitxHub [62].

Every Cross-chain gateway is built using two core components, this is the peer logic itself, which is used for every compatible blockchain [66], and a client for that particular blockchain, like Hyperledger or Ethereum. Once implemented together these two combined create the Cross-chain gateway for a specific App-chain.

In a normal Cross-chain gateway, it is the cross-chain gateway, which is responsible for communicating with the App-chain and Relay chain, everything must go through it. A high-level sequence diagram for how it would work in Ethereum is shown in Figure 8

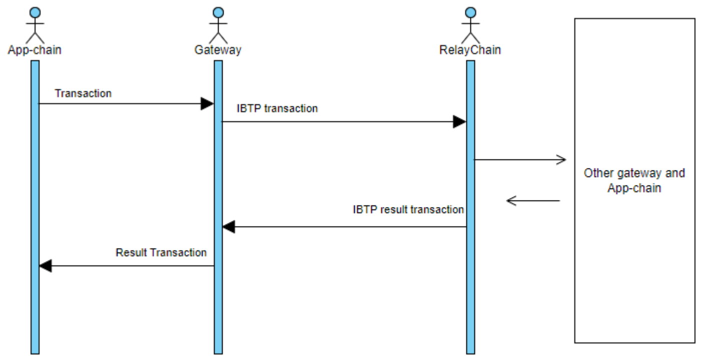


Figure 8: How the Ethereum Cross-chain gateway interacts with App-chain and Relay Chain

Hyperledger Fabric 2.3 solution is built on Fabric 1.4 solution. It was attempted to replace the Fabric 1.4 SDK with the Fabric 2.3 SDK in the Hyperledger client, however, this made it not compatible with the peer logic. For this reason, a third component was introduced, named the broker. The broker sits between the App-chain and Cross-chain gateway, it is responsible for translating the information sent and received into something readable for the Cross-chain gateway and the App-chain. With this solution, it was now possible to use the modern Hyperledger Fabric 2.3 instead of the older Hyperledger Fabric 1.4. A High-level sequence diagram in Figure 9 shows how the created solution interacts with the other components.

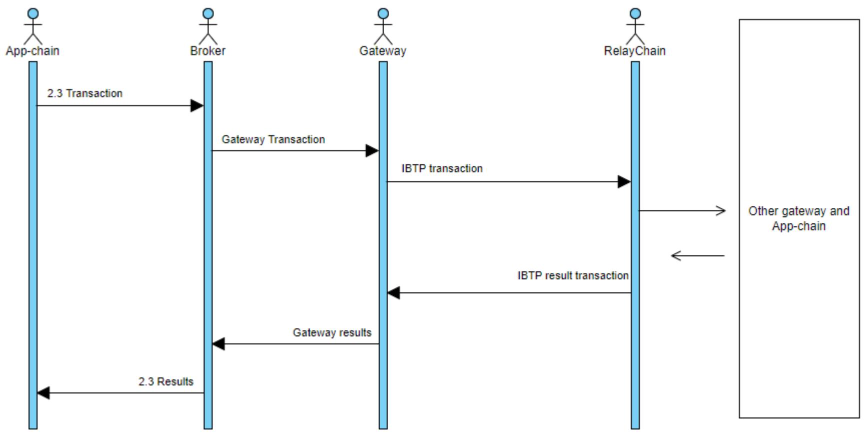
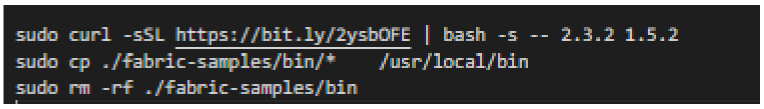


Figure 9: How the created Fabric 2.3 solution Cross-chain gateway interacts with App-chain and Relay Chain

5.3.3 Cross-chain gateway Hyperledger Fabric

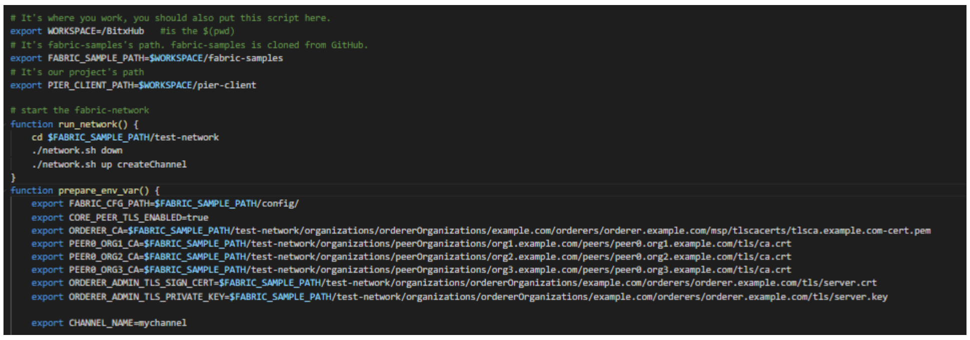
For the Fabric Cross-chain gateway to work, the first thing which must be done is to create a Hyperledger Fabric Test Network in the environment. The Fabric Test Network was chosen because this is a fully functioning Fabric network, which everyone can use. It was decided to use Fabric 2.3.2, because this was the latest release when the development was started. A simple script shown in Figure 10 can be used to download the whole fabric sample in the correct version, and from there the user can follow the instructions provided in **Using the Fabric test network** [67]



```
sudo curl -sSL https://bit.ly/2ysbOFE | bash -s -- 2.3.2 1.5.2
sudo cp ./fabric-samples/bin/*    /usr/local/bin
sudo rm -rf ./fabric-samples/bin
```

Figure 10: Initiate Fabric network

Once the channel is up and running, the environmental variables must be set up locally in the development environment. This is so communication can be done correctly when setting up the Chain code in the environment, and later for communication between the chains, the environmental variables which need to be set up are shown in Figure 11.



```bash
# It's where you work, you should also put this script here.
export WORKSPACE=/BitxHub   #is the $(pwd)
# It's fabric-samples's path. fabric-samples is cloned from GitHub.
export FABRIC_SAMPLE_PATH=$WORKSPACE/fabric-samples
# It's our project's path
export PIER_CLIENT_PATH=$WORKSPACE/pier-client

# start the fabric-network
function run_network() {
    cd $FABRIC_SAMPLE_PATH/test-network
    ./network.sh down
    ./network.sh up createChannel
}
function prepare_env_var() {
    export FABRIC_CFG_PATH=$FABRIC_SAMPLE_PATH/config/
    export CORE_PEER_TLS_ENABLED=true
    export ORDERER_CA=$FABRIC_SAMPLE_PATH/test-network/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem
    export PEER0_ORG1_CA=$FABRIC_SAMPLE_PATH/test-network/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt
    export PEER0_ORG2_CA=$FABRIC_SAMPLE_PATH/test-network/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt
    export PEER0_ORG3_CA=$FABRIC_SAMPLE_PATH/test-network/organizations/peerOrganizations/org3.example.com/peers/peer0.org3.example.com/tls/ca.crt
    export ORDERER_ADMIN_TLS_SIGN_CERT=$FABRIC_SAMPLE_PATH/test-network/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/tls/server.crt
    export ORDERER_ADMIN_TLS_PRIVATE_KEY=$FABRIC_SAMPLE_PATH/test-network/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/tls/server.key

    export CHANNEL_NAME=mychannel
```

Figure 11: The enviroumental variables needed to setup the network

Once the envioruenmntal variables are put in place, the chain code can be placed onto the blockchain.

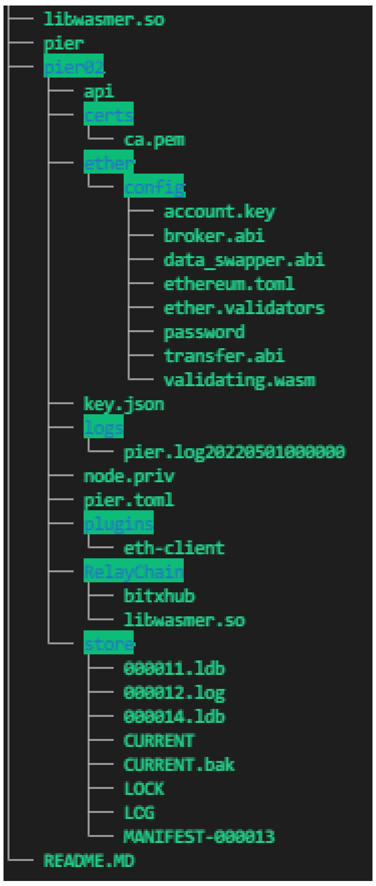
This is done by first packing the relevant Golang scripts into what is called a package. This means all the relevant scripts Broker, Data_Swapper, and Transfer. Once the chain code is packaged in a tar.gz file, it must first be installed onto the blockchain nodes. This must be done for each organization's peer, which is inside the channel, where the chain code should be installed. Once the chain code is installed, each organization must approve the chain code, before it finally can be committed and initiated.

Once the chain code is successfully committed to the nodes in the Hyperledger blockchain, the Cross-chain gateway is ready to be connected to the Relay chain.

5.3.4 Cross-chain gateway Ethereum

The Ethereum Cross-chain gateway was built following the documentation on Cross-chain gateway deployment provided by BitxHub [62].

Once the documentation has been followed, the user should end up with a folder tree as shown in Figure 12



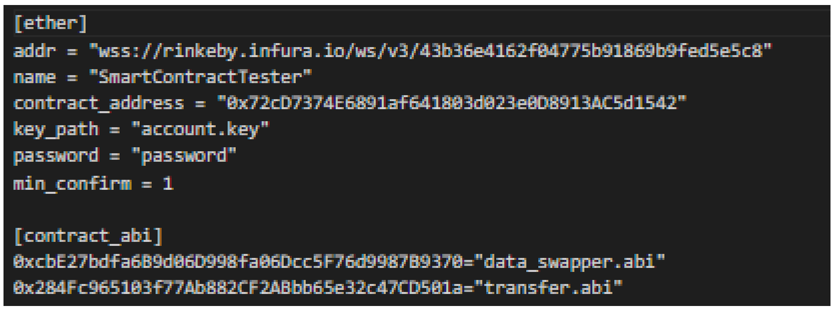
```
├── libwasmer.so
├── pier
├── pier02
│   ├── api
│   ├── certs
│   │   └── ca.pem
│   ├── ether
│   │   └── config
│   │       ├── account.key
│   │       ├── broker.abi
│   │       ├── data_swapper.abi
│   │       ├── ethereum.toml
│   │       ├── ether.validators
│   │       ├── password
│   │       ├── transfer.abi
│   │       └── validating.wasm
│   ├── key.json
│   ├── logs
│   │   └── pier.log20220501000000
│   ├── node.priv
│   ├── pier.toml
│   ├── plugins
│   │   └── eth-client
│   ├── RelayChain
│   │   ├── bitxhub
│   │   └── libwasmer.so
│   └── store
│       ├── 000011.ldb
│       ├── 000012.log
│       ├── 000014.ldb
│       ├── CURRENT
│       ├── CURRENT.bak
│       ├── LOCK
│       ├── LOG
│       └── MANIFEST-000013
└── README.MD
```

Figure 12: Ethereum Cross-chain gateway tree folder structure

The important changes which need to be made after the creation is in the ether/config files. In order to launch the Cross-chain gateway, there must first be smart contracts deployed on an Ethereum network.

To this end, Truffle [68] was used. Truffle is a Node.js npm package, which allows the developer to easily deploy contracts on an Ethereum network of their choosing and use compilers of their choosing. All the scripts in BitxHub V1.6.5 are written in solidity version 0.5.6, therefore it was chosen to use this version. The network chosen was Rinkeby Test network. An alternative would be to use Ganache-CLI which would give unlimited test ETH, however, Rinkeby acts more like the main network, and therefore would give a more realistic result when conducting tests. The choice of using Rinkeby Testnet meant we had to connect to the Rinkeby Testnet, to this end, it was chosen to use Infura [69]. Infura allows developers to easily connect to either a Testnet or the Mainnet, using the nodes which they have in the different blockchains, saving the developer the time of creating their own node in the ecosystem. Infura does this by providing endpoints to the nodes, which can be used to connect and communicate with the smart contracts developed. Finally to deploy a contract a user must provide a wallet with ETH, so the price for deploying and subsequently sending transactions can be paid for. We choose to use Matamask [70].

Once the contracts had been deployed on the Rinkeby Testnet, the information about the Blockchain connection (Infura), contract addresses and the wallet key had to be added to the Ethereum.toml file. This is because when the Cross-Chain gateway gets initiated, it must know which smart contracts to use, how to connect to the blockchain, and how to access the wallet for transaction payments Figure 13 shows an example of how it should look like.

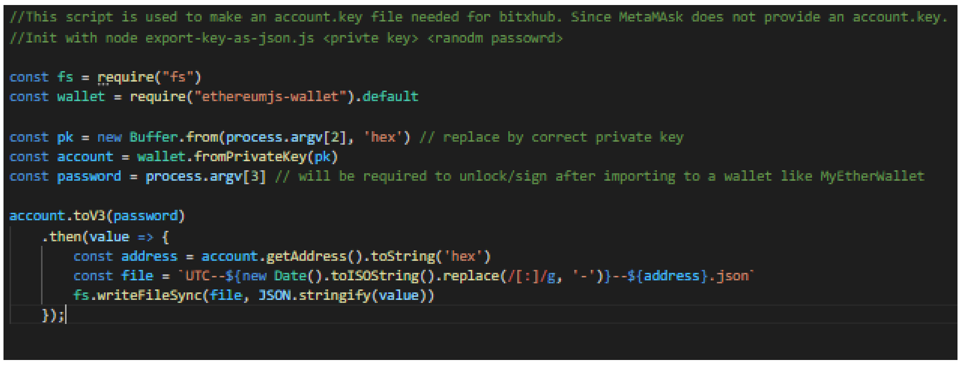


```
[ether]
addr = "wss://rinkeby.infura.io/ws/v3/43b36e4162f04775b91869b9fed5e5c8"
name = "SmartContractTester"
contract_address = "0x72cD7374E6891af641803d023e0D8913AC5d1542"
key_path = "account.key"
password = "password"
min_confirm = 1

[contract_abi]
0xcbE27bdfa6B9d06D998fa06Dcc5F76d9987B9370="data_swapper.abi"
0x284Fc965103f77Ab882CF2ABbb65e32c47CD501a="transfer.abi"
```

Figure 13: The structure of the Ethereum.toml file in V1.6.5

The other values in ether/config shown in Figure 12 also need to be populated with the correct values, so that when the Cross-chain gateway starts it will function as intended. The ABIs broker.abi, transfer.abi, and data_swapper.abi all can be populated by a string of the respective ABIs which were created when the contracts were deployed.

Because Metamask does not provide an account.key file, one had to be produced using the private key of the Metamask account, and a password of choice. The code snippet is shown in Figure 14 which was used to enable this. Once the the file with the account.key info is created, it can be copied to the account.key in ether/config, and the password used must be placed in the password file. The final file in ehter/config, validating.wasm was not touched during this project. This file is however the file that provides the validation rules for the Cross-chain gateway. If a user wishes to add or change the security requirements which are checked before the received data is seen as valid by the Cross-chain gateway, and sent to the underlying blockchain, then this file needs to be changed. BitxHub provides a basic tutorial on how this can be done in their Wiki [63].

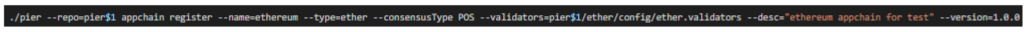


```
//This script is used to make an account.key file needed for bitxhub. Since MetaMAsk does not provide an account.key.
//Init with node export-key-as-json.js <privte key> <ranodm passowrd>

const fs = require("fs")
const wallet = require("ethereumjs-wallet").default

const pk = new Buffer.from(process.argv[2], 'hex') // replace by correct private key
const account = wallet.fromPrivateKey(pk)
const password = process.argv[3] // will be required to unlock/sign after importing to a wallet like MyEtherWallet

account.toV3(password)
    .then(value => {
        const address = account.getAddress().toString('hex')
        const file = `UTC--${new Date().toISOString().replace(/[:]/g, '-')}--${address}.json`
        fs.writeFileSync(file, JSON.stringify(value))
    });
```

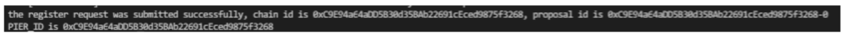
Figure 14: Code used to make account.key for metamask

With these changes, the Ethereum Cross-chain gateway can be connected to the Relay chain.

Once both Ethereum and Fabric are ready for deployment, they must first be registered. The registration will give the gateways a unique Pier-ID and Proposal-ID, which will be used to communicate and register respectively. The Registration command and the output given are shown in Figure 15



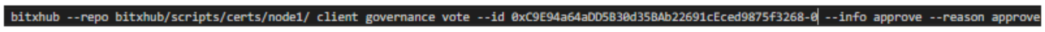
```
./pier --repo=pier$1 appchain register --name=ethereum --type=ether --consensusType POS --validators=pier$1/ether/config/ether.validators --desc="ethereum appchain for test" --version=1.0.0
```



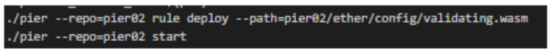
```
the register request was submitted successfully, chain id is 0xC9E94a64aDD5B30d35BAb22691cEced9875f3268, proposal id is 0xC9E94a64aDD5B30d35BAb22691cEced9875f3268-0
PIER_ID is 0xC9E94a64aDD5B30d35BAb22691cEced9875f3268
```

Figure 15: A registration command from Ethereum, and the yielded Pier and proposal ID

With the Proposal-ID, the nodes inside the Relay-chain can now approve the Cross-chain gateway to join the ecosystem. Once a majority of nodes have approved the Cross-chain gateway it is now possible to deploy the rules, and finally start the Cross-chain gateway, which now can communicate with the BitxHub Relay-chain. The commands for Ethereum to do this are shown in Figure 16



```
bitxhub --repo bitxhub/scripts/certs/node1/ client governance vote --id 0xC9E94a64aDD5B30d35BAb22691cEced9875f3268-0 --info approve --reason approve
```



```
./pier --repo=pier02 rule deploy --path=pier02/ether/config/validating.wasm
./pier --repo=pier02 start
```

Figure 16: Shows the commands used to connect to a node, and the commands to deploy rules and start the peer

The final setup is shown in Figure 17 The Ethereum and Hyperledger Cross-chain gateways are now connected to their respective Blockchains, as well as the Relaychain, and can communicate information to each other.

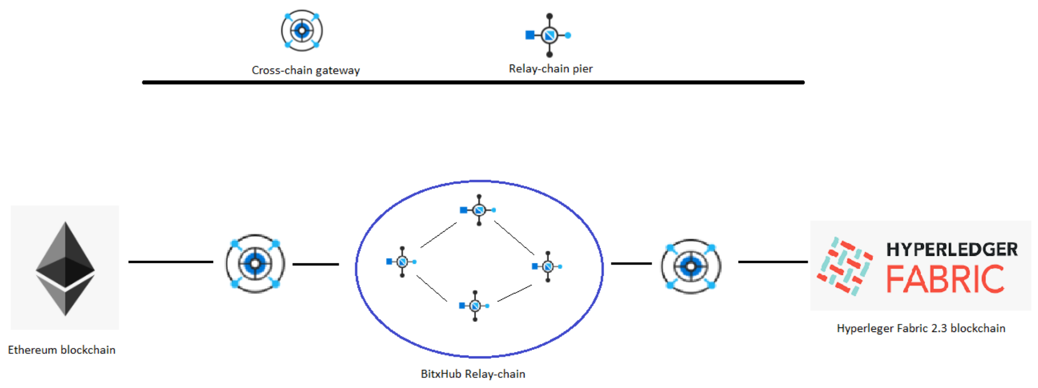
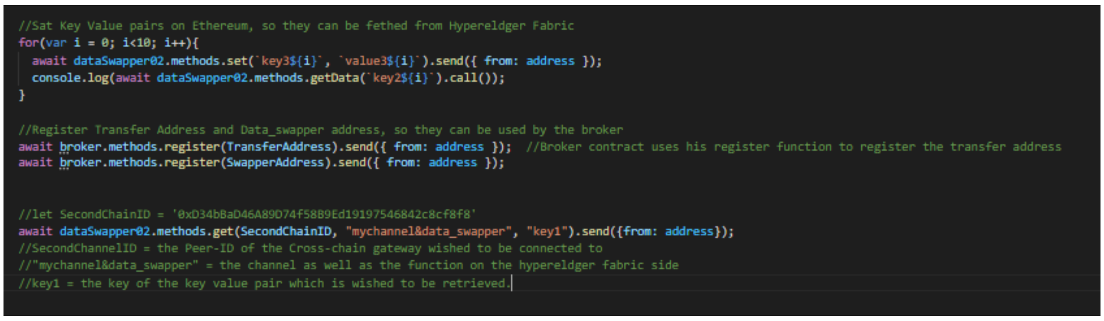


Figure 17: Bitxhub complete architecture

On the Hyperledger Fabric side, everything is now prepared to communicate, however on the Ethereum side, the broker contract must still approve the data_swapper and transfer the smart contract. This is for security reasons, because Ethereum is a permissionless blockchain, anyone who has the address to broker could in theory use it to send cross-chain information over. However, the developers thought of this and implemented a method to only allow approved smart contract addresses to communicate with the broker contract. For this reason, before we can communicate, we need to approve the addresses of data_swapper and transfer. Once they are registered, both sides can communicate with each other. A simple interaction is shown in Figure 18



```
//Sat Key Value pairs on Ethereum, so they can be fethed from Hypereldger Fabric
for(var i = 0; i<10; i++){
  await dataSwapper02.methods.set(`key3${i}`, `value3${i}`).send({ from: address });
  console.log(await dataSwapper02.methods.getData(`key2${i}`).call());
}

//Register Transfer Address and Data_swapper address, so they can be used by the broker
await broker.methods.register(TransferAddress).send({ from: address });  //Broker contract uses his register function to register the transfer address
await broker.methods.register(SwapperAddress).send({ from: address });


//let SecondChainID = '0xD34bBaD46A89D74f58B9Ed19197546842c8cf8f8'
await dataSwapper02.methods.get(SecondChainID, "mychannel&data_swapper", "key1").send({from: address});
//SecondChannelID = the Peer-ID of the Cross-chain gateway wished to be connected to
//"mychannel&data_swapper" = the channel as well as the function on the hypereldger fabric side
//key1 = the key of the key value pair which is wished to be retrieved.
```

Figure 18: A simple method to fetch data from HyperledgerFabric

Once everything is connected, the first thing which must be done is the ledgers must populated with data. Both sides must place some information on their respective ledgers in order for it to be fetched. After this on the Ethereum side, the broker must register the addresses of the data_swapper and transfer smart contracts, this must only be done once. Finally, a transaction can be made with the Hyperledger Fabric blockchain. Once this is done and the data_swappers **get** command has been called, the information will be retrieved following the sequence diagram shown in Figure 19

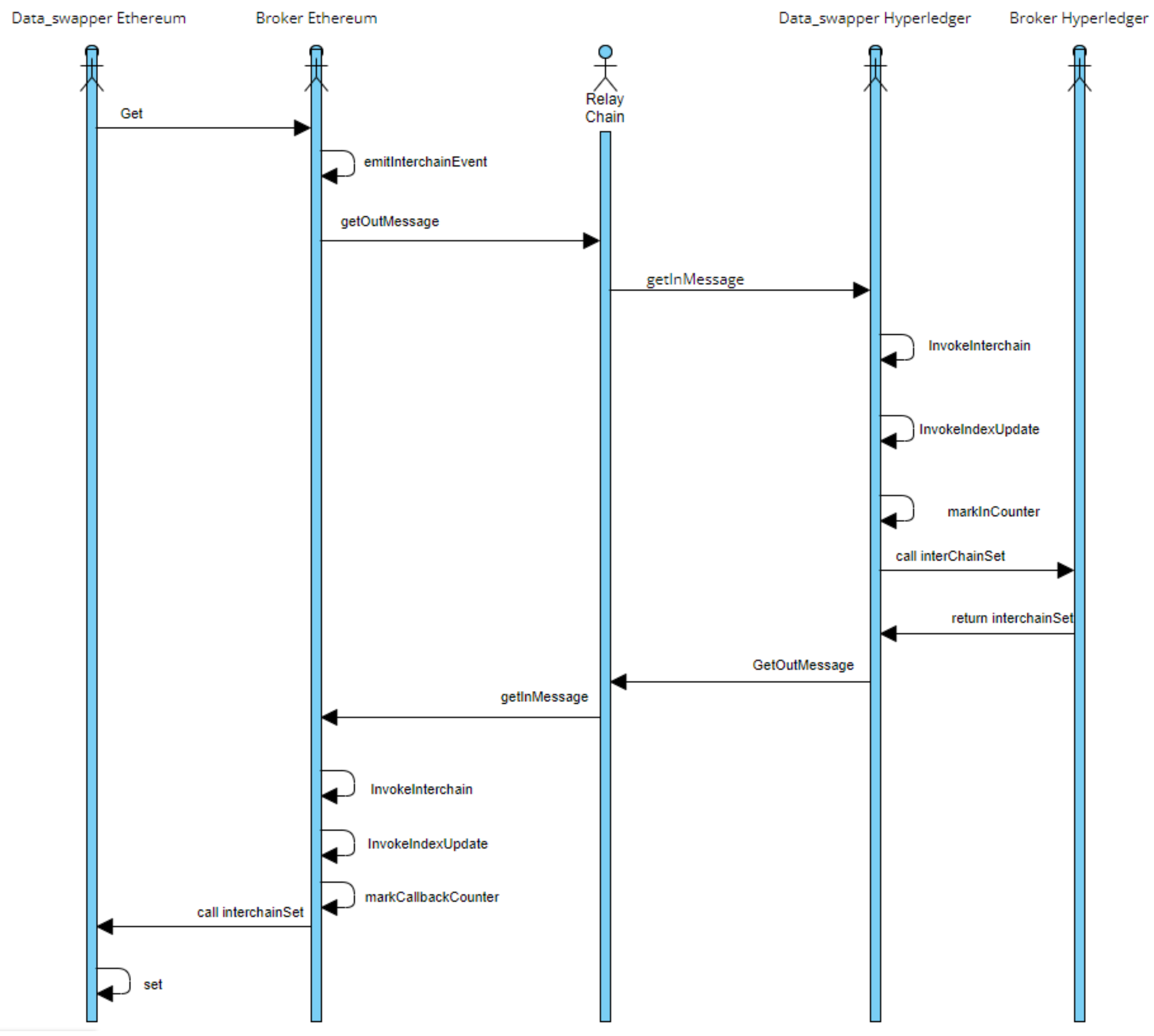


Figure 19: The complete sequence diagram from a get call to the eventual placing of the requested data in the blockchain. Cross-chain gateways are removed

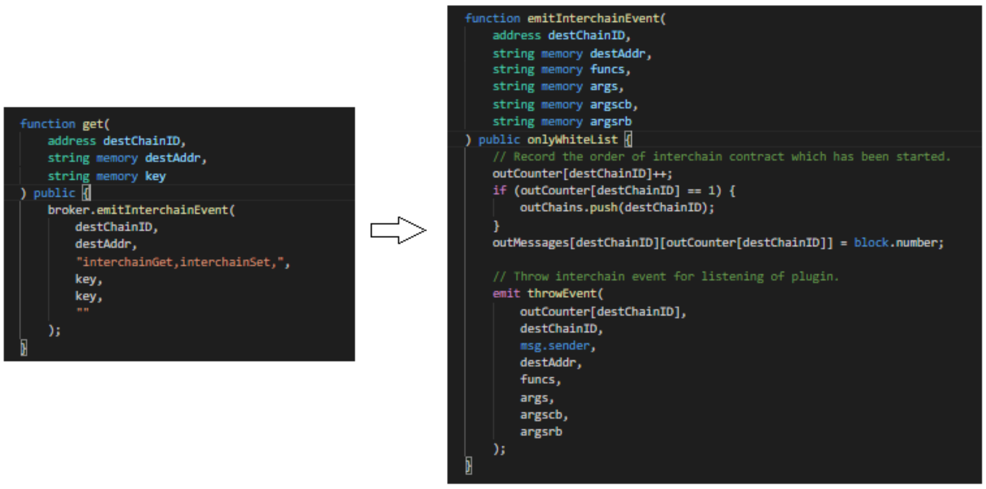
The first and only function which has to be called in order to retrieve information from the other blockchain is data_swapper.get(). The get function takes in three parameters

- Pier-ID: which was created when the Cross-chain gateway was first registered as shown in Figure 15.
- data_swapper location: For Ethereum, this is the smart contract address,

while for Hyperledger this is the channel it is in combined with the name of the contract. As shown in Figure 18

- data: The key from the Key-Value pair which the users wish to retrieve information about on the other blockchain.

The get function will then communicate with the broker contract, which then will communicate with the cross-chain gateway to send out this information. The code for this interaction is shown in Figure 20



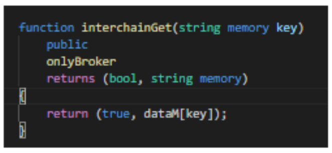
```
function get(
    address destChainID,
    string memory destAddr,
    string memory key
) public {
    broker.emitInterchainEvent(
        destChainID,
        destAddr,
        "interchainGet,interchainSet,",
        key,
        key,
        ""
    );
}
```

```
function emitInterchainEvent(
    address destChainID,
    string memory destAddr,
    string memory funcs,
    string memory args,
    string memory argscb,
    string memory argsrb
) public onlyWhiteList {
    // Record the order of interchain contract which has been started.
    outCounter[destChainID]++;
    if (outCounter[destChainID] == 1) {
        outChains.push(destChainID);
    }
    outMessages[destChainID][outCounter[destChainID]] = block.number;

    // Throw interchain event for listening of plugin.
    emit throwEvent(
        outCounter[destChainID],
        destChainID,
        msg.sender,
        destAddr,
        funcs,
        args,
        argscb,
        argsrb
    );
}
```

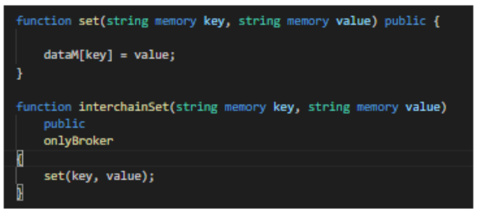
Figure 20: The code used in the Ethereum side to request information from the other blockchain

The Relay chain will then send this information to the appropriate Cross-chain gateway, where it will then trigger updates of the current state of the Cross-chain gateway, depending on if it is ingoing or outgoing.

In both cases, it will invoke InvokeInterchain and InvokeIndexUpdate. Depending on if it's ingoing or outgoing it will call MarkInCounter or markCallbackCounter respectively. These three functions all work on maintaining the correct state of the cross-chain gateways ensuring that information is dealt with appropriately. If it's an ingoing message, the broker will also call interChainGet. Interchain get is responsible for retrieving the desired information, and returning it to the broker, which then with the return address from the IBTP protocol will send it back to the Relay-chain. Finally, the information will be set in the blockchain which first requested the information. The get function is used to retrieve the information, and the two set functions used to set the information are shown in Figure 21.



```
function interchainGet(string memory key)
    public
    onlyBroker
    returns (bool, string memory)
{
    return (true, dataM[key]);
}
```



```
function set(string memory key, string memory value) public {

    dataM[key] = value;
}

function interchainSet(string memory key, string memory value)
    public
    onlyBroker
{
    set(key, value);
}
```

Figure 21: The get and set functions used in the cross-chain transaction

5.4 Performance and challenges

With a functioning solution presented in subsection 5.3, there was now a desire to see how this solution performs. The main focus was transaction speed. subsubsection 5.4.1 will present the transaction in its most basic form, sending one transaction from blockchain A to Blockchain B. subsubsection 5.4.2 will try to increase the transactions per second (tps) by increasing the number of messages sent in different time intervals. subsubsection 5.4.3 will observe how the system functions when sending higher amounts of data. The final part subsubsection 5.4.4 will present issues found while testing the performance of the interoperability solution

5.4.1 Average Transaction speed

With a functioning solution that was able to interoperate data between Hyperledger fabric and Ethereum, we wanted to see a bit how the solution performed. The reasoning for doing this was an observed lack of information about performance in the solutions which could interoperate data between heterogeneous blockchains, SCIP, and BitxHub.

For all the experiments checkmarks were set at appropriate places, measuring the time in milliseconds from 1970 and printed out when these checkmarks were hit. There were set a total of three checkmarks.

- Checkpoint One: was triggered when the **get** function was called.
- Checkpoint Two: Was set when the other blockchain first retrieved the request in **invokeInterchain**.
- Ckechpoint Three: Was set once the information desired was set on the blockchain, after **set** was called.

Once the information was gathered, the received request would be measured by taking:

(Checkpoint Two - Checkpoint One) / 1000

The Message returned would be measured by taking

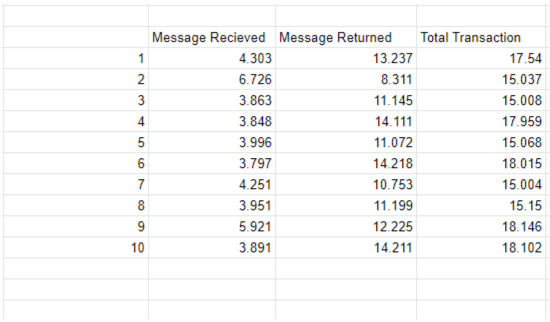
(Checkpoint Three - Checkpoint Two) / 1000

This would give us the three measurements Message Received, Message Returned, and total time in seconds.

The first experiment focused on transaction speed. Transaction speed is a relevant matrix to measure, as the current industry requires a high transaction speed in order to send stay relevant. The experiment was set up using the Hyperledger Fabric solution and Ethereum solution described. A transaction was sent from Hyperledger to Ethereum. The time was recorded, and another transaction was sent.

It was found that the average transaction speed between Hyperledger Fabric and Ethereum using BitxHub was 15.78 seconds This average was received after running ten transactions between Hyperledger Fabric and Ethereum on five different occasions, where one such experiment is shown in Figure 22. However, it seems that the Transaction speed is linked to the average block time inside the blockchain. This is because BitxHub needs to set the information requested by users into the respective blockchains. Therefore the whole transaction process needs to wait until the transaction is set in the blockchain.

For this reason, another experiment was set up using two Ethereum blockchains to talk to each other over BitxHub. The expected result would be that when two Ethereum blockchains communicate together, the information would first need to be requested. The request should take the time equivalent to one block creation, as some data gets updated, while the returning and saving this information on the second blockchain should take another block creation. It was expected that the total time should be somewhere close to 30 seconds to this extent.



	Message Recieved	Message Returned	Total Transaction
1	4.303	13.237	17.54
2	6.726	8.311	15.037
3	3.863	11.145	15.008
4	3.848	14.111	17.959
5	3.996	11.072	15.068
6	3.797	14.218	18.015
7	4.251	10.753	15.004
8	3.951	11.199	15.15
9	5.921	12.225	18.146
10	3.891	14.211	18.102

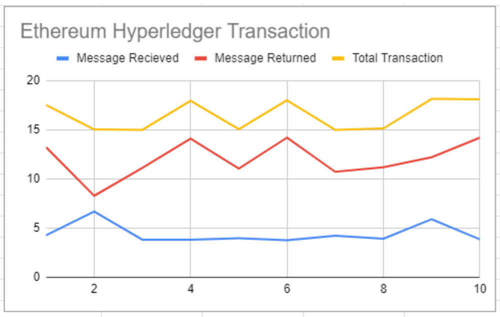
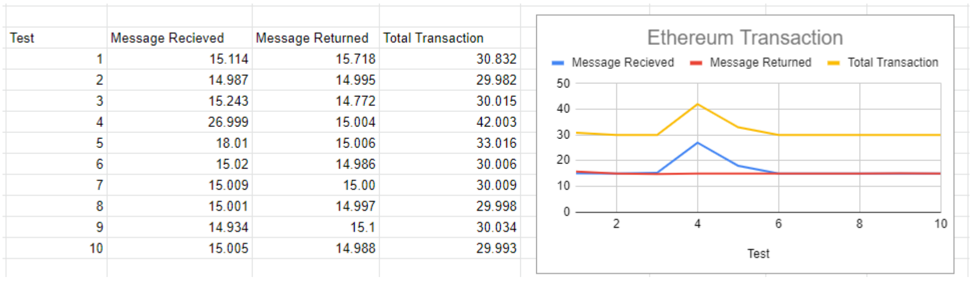


Figure 22: Average transaction speed between Ethereum and Hyperledger Fabric

The test results shown in Figure 23 do indeed support the hypothesis that the Transaction speed is linked to average block time. This means that the transaction speed between Ethereum and Hyperledger Fabric using BitxHub is not determined by the speed of the cross-chain transaction but rather the speed of Ethereum.

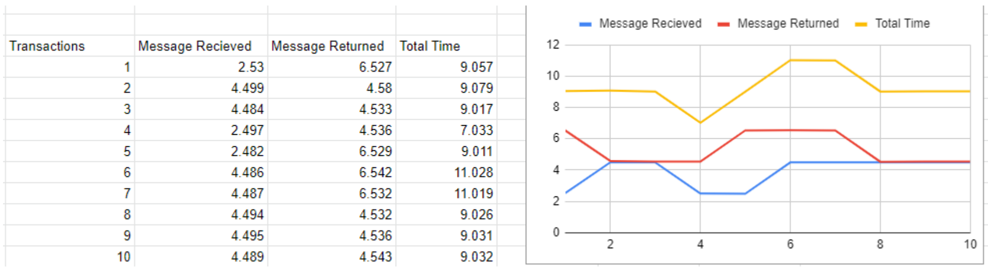


Test	Message Recieved	Message Returned	Total Transaction
1	15.114	15.718	30.832
2	14.987	14.995	29.982
3	15.243	14.772	30.015
4	26.999	15.004	42.003
5	18.01	15.006	33.016
6	15.02	14.986	30.006
7	15.009	15.00	30.009
8	15.001	14.997	29.998
9	14.934	15.1	30.034
10	15.005	14.988	29.993

Figure 23: Average transaction speed between Ethereum and Ethereum

A final experiment was done between Hyperledger Fabric and Hyperledger Fabric. The average here after three separate tests doing ten transactions, sending one every 25 seconds resulted in an average transaction time of 10.36 seconds. One such experiment is shown in Figure 24.

From this experiment, it can be proven that the BitxHub interoperability method is not the only slowing factor when interoperating data between Hyperledger and Ethereum, as the transaction speed between Hyperledger and Hyperledger is about 5 seconds faster than Ethereum and Hyperledger.



Transactions	Message Recieved	Message Returned	Total Time
1	2.53	6.527	9.057
2	4.499	4.58	9.079
3	4.484	4.533	9.017
4	2.497	4.536	7.033
5	2.482	6.529	9.011
6	4.486	6.542	11.028
7	4.487	6.532	11.019
8	4.494	4.532	9.026
9	4.495	4.536	9.031
10	4.489	4.543	9.032

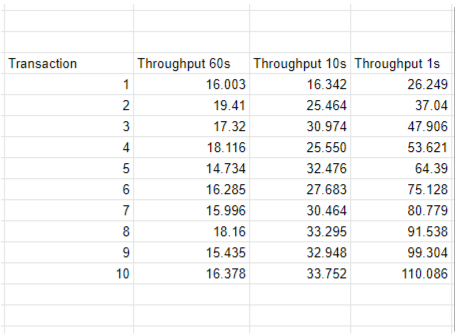
Figure 24: Average Transaction speed between Hyperledger 2.3 and Hyperledger 2.3

A transaction speed of one transaction every 15 seconds is very slow. Compared to existing solutions like Polkadot, which claims it can do 160 000 tps, without much proof to back this up [71], it can claim at least 10 000 tps cited in [49]. It should be mentioned that these between 160 000 - 10 000 are shared between all parachains in the domain. This means if 20 parachains were using the lower end, a para-chain could maximum expect 500 tps. Polkadot is, however, an interoperability solution for homogeneous blockchains. There are currently no articles measuring the transaction speed of two heterogeneous blockchains to the knowledge of the writer.

5.4.2 Throughput

So far, the transactions have been sent over BitxHub, in a fashion where we only see the average time between sending the transaction and later receiving the transaction. A potential way to increase the number of transactions per second is to send multiple transactions simultaneously. For example, if the system can handle two transactions within 15 seconds, the tps would be halved.

An experiment was set up where the first set of transactions was sent every minute, ensuring that even at the worst conditions there should not be any throughput issues. The second set of transactions was sent every 10 seconds. This is below the average throughput of 15.78 seconds, ensuring that multiple transactions would need to be handled at once. Finally, the third set of transactions was sent out every second. The results are shown in Figure 25



Transaction	Throughput 60s	Throughput 10s	Throughput 1s
1	16.003	16.342	26.249
2	19.41	25.464	37.04
3	17.32	30.974	47.906
4	18.116	25.550	53.621
5	14.734	32.476	64.39
6	16.285	27.683	75.128
7	15.996	30.464	80.779
8	18.16	33.295	91.538
9	15.435	32.948	99.304
10	16.378	33.752	110.086

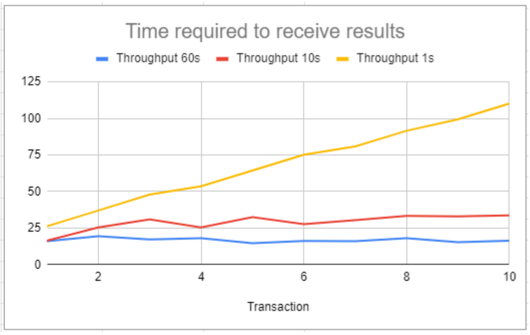
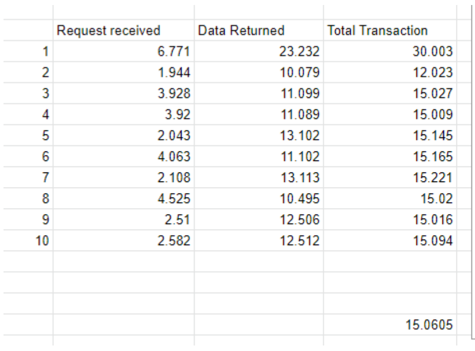


Figure 25: The average transaction speed measured when a transaction was send every 60 seconds,10 seconds and 1 second

In the 60-second throughput, which should not pose a challenge, the system worked as expected, giving a transaction time average of 16.16 seconds. This is a bit higher than the average found in the first experiment where the average was found to be 15.78. This could mean there is a small stress on the system, or the average block creation time was worse during this experiment. In messages sent every 10 seconds, there is at the beginning a typical throughput, but this quickly increases to about double the average transaction time, to 28.98 seconds for each transaction to be processed. Finally, if a transaction is sent every second, there is a linear increase in time needed to handle each transaction, averaging 68.60 seconds after only 10 transactions. Nevertheless, the total time to process all the transactions remained about the same. The ten-second throughput was at 15.775 seconds, while the one-second throughput was 16.210 seconds, indicating that the first transaction received needs to be completed before a new transaction can be handled. These results suggest that it is impossible to increase the transaction speed with the current Cross-Chain gateway that BitxHub has. The cross-chain would need to allow multiple transactions to be processed at once to enable an increase in transactions per second; however, if a single node could handle this increased workload is difficult to say.

5.4.3 Amount of Data

With the transaction speed seeming to be forced to around 15 seconds, another experiment was done to see how much data could be sent over BitxHub, without any increase in the transaction speed. Literature usually does not mention how big the payload can be for a single transaction. Hedera's whitepaper [17] mentions a 250 000 tps with 100 bytes max in each transaction and a consensus delay of 6 seconds. A solution to tackle a slow transaction speed is to send a lot of data in big bunches. for this reason, an experiment was done to see how much data could be sent over the solution, out there being an increase in transaction time.



	Request received	Data Returned	Total Transaction
1	6.771	23.232	30.003
2	1.944	10.079	12.023
3	3.928	11.099	15.027
4	3.92	11.089	15.009
5	2.043	13.102	15.145
6	4.063	11.102	15.165
7	2.108	13.113	15.221
8	4.525	10.495	15.02
9	2.51	12.506	15.016
10	2.582	12.512	15.094
			15.0605

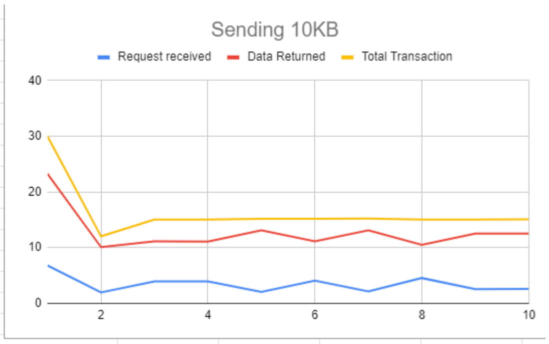
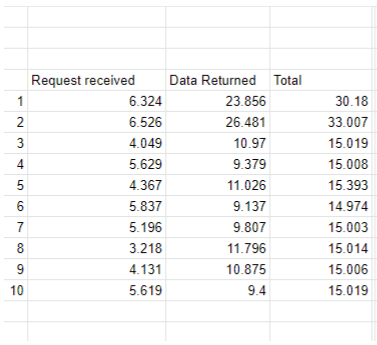


Figure 26: Average transaction speed when sending 10kb data over BitxHub

At 10KB per transaction the system, as shown in Figure 26 works as expected. The average transaction speed is around 15 seconds, which seems to occur consistently throughout the transactions.



	Request received	Data Returned	Total
1	6.324	23.856	30.18
2	6.526	26.481	33.007
3	4.049	10.97	15.019
4	5.629	9.379	15.008
5	4.367	11.026	15.393
6	5.837	9.137	14.974
7	5.196	9.807	15.003
8	3.218	11.796	15.014
9	4.131	10.875	15.006
10	5.619	9.4	15.019

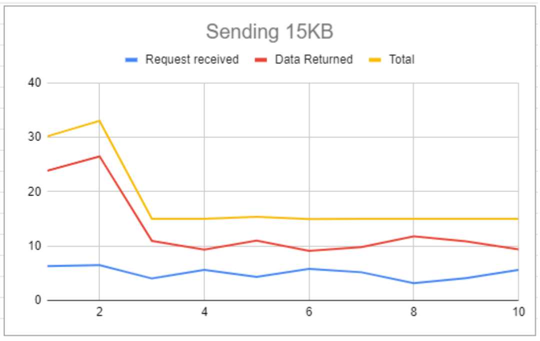
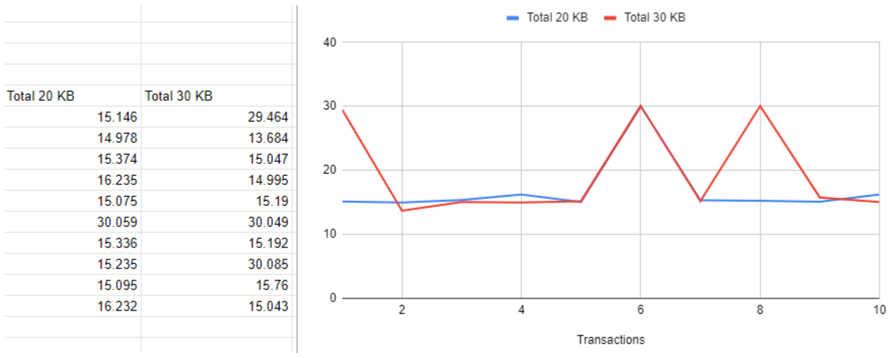


Figure 27: Average transaction speed when sending 10kb data over BitxHub

At 15KB per transaction, the system acts the same as in 10KB, there are a few outliers, but in the majority of the cases, the transaction speed is still around 15 seconds, which is the average block production time.



Total 20 KB	Total 30 KB
15.146	29.464
14.978	13.684
15.374	15.047
16.235	14.995
15.075	15.19
30.059	30.049
15.336	15.192
15.235	30.085
15.095	15.76
16.232	15.043

Figure 28: Average transaction speed when sending 20Kb and 30Kb data over BitxHub

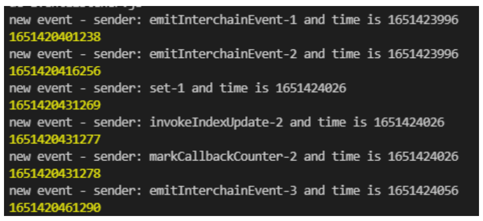
Finally as shown in Figure 28 two more experiments were done with 20Kb and 30Kb, while 20Kb had its outlier, there was a clear increase in the 30Kb experiment.

It was observed that there did come an increase in transactions taking more than 30 seconds to be processed, however, there was no clear correlation between transaction size, and increased transaction time. The best conclusion can be that transferring a high amount of data at once increases the risk of a transaction taking double the normal time.

5.4.4 Issues Discoveries

Double emit bug

One huge issue which can occur is that a single transaction gets sent twice by mistake. This issue, based on observations from the experiments seems to be an Ethereum Cross-chain gateway issue only. It occurs when Ethereum requests information from Hyperledger. This issue happens very rarely, and there is no method to replicate it. However, the consequence of this issue is that the Cross-chain gateway cannot send or receive new messages. This is most likely because it is still waiting for the returned information to return a second time. This issue is easily solved by simply restarting the cross-chain gateway. Once the Cross-chain gateway was restarted, the problem seems to resolve itself, and the requested and sent data was eventually retrieved without any loss. Figure 29 shows this issue. A message was sent; the message was however sent twice (emitInterchainEvent). The information was retrieved by BitxHub and saved on the chain (Set). The appropriate parameters for the Cross-chain gateway were updated (invokeIndexUpdate, markCallbackCounter). However, once a new transaction was sent (the third emitInterchainEvent) it never received a response because it most likely is waiting for an answer for the second emitInterchainEvent. A sequence diagram shows the suspected behavior Figure 30. It is unknown if the other blockchain receives the information and does not process it, or if it never receives the information because it could not be replicated, we failed to gather enough information to conclude anything about it.



```
new event - sender: emitInterchainEvent-1 and time is 1651423996
1651420401238
new event - sender: emitInterchainEvent-2 and time is 1651423996
1651420416256
new event - sender: set-1 and time is 1651424026
1651420431269
new event - sender: invokeIndexUpdate-2 and time is 1651424026
1651420431277
new event - sender: markCallbackCounter-2 and time is 1651424026
1651420431278
new event - sender: emitInterchainEvent-3 and time is 1651424056
1651420461290
```

Figure 29: If a sent transaction, for some reason invokes two EmitInterchain-Event, a single result will be returned, but any subsequent calls will be in limbo.

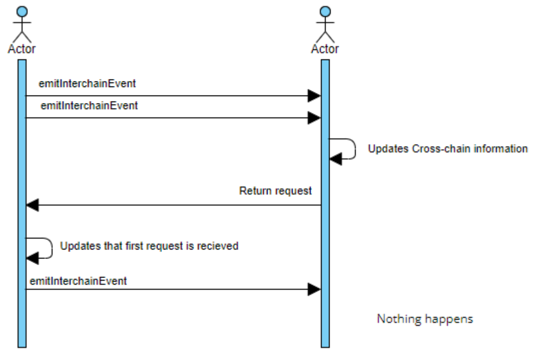
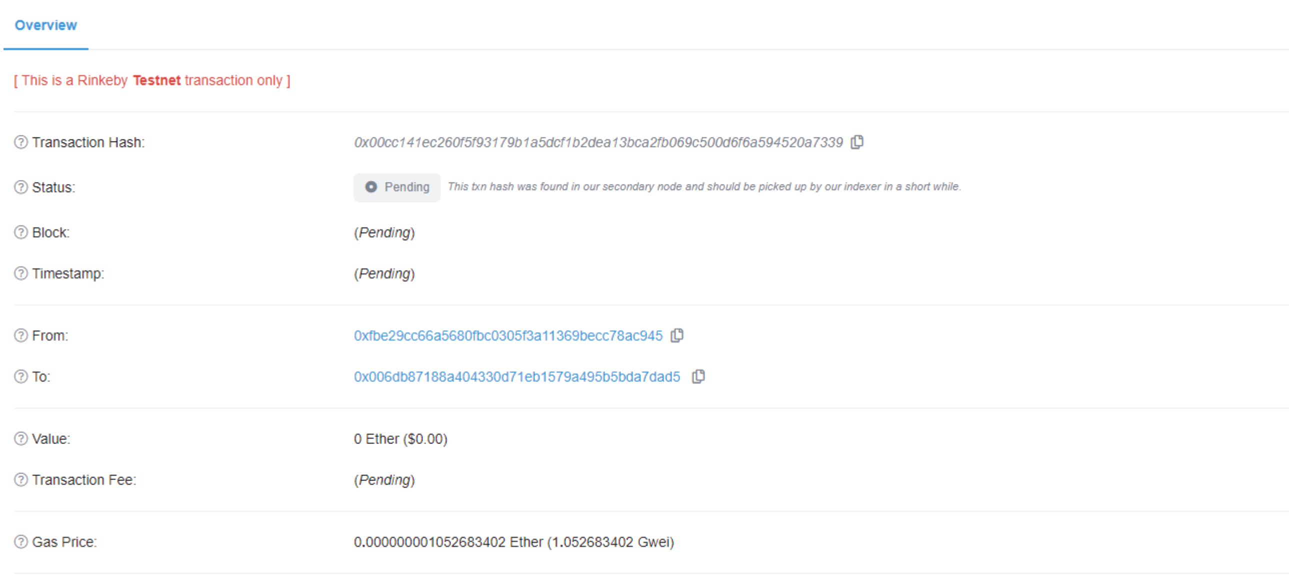


Figure 30: A sequence diagram of the Double emit bug.

Gas price issue

This problem occurs once the gas price on the Ethereum blockchain fluctuates a lot. A gas price that is below the average gas price will be set as the reward for placing the transaction on a block Figure 31. However, because the gas price is lower than the current average gas price for getting a block on the blockchain (shown in Figure 32), the transaction will be in limbo until the gas price drops to the amount in the transaction. Then, the transaction will inevitably be a success, as shown in Figure 33. The time this takes is the issue. Sometimes this issue will be resolved in seconds. However, it has also taken 29 minutes for this to be resolved. The way the Cross-chain getaway operates will result in all ingoing and outgoing transactions having to wait until this inevitably is resolved. This is the equivalent of a 30-minute downtime, where nothing can occur. This is a problem that occurs on Ethereum and might be fixable by setting the gas price willing to be used to a higher priority, however, this is not immune to a big spike either.


Overview

[This is a Rinkeby **Testnet** transaction only]

Transaction Hash:	*0x00cc141ec260f5f93179b1a5dcf1b2dea13bca2fb069c500d6f6a594520a7339*
Status:	Pending *This txn hash was found in our secondary node and should be picked up by our indexer in a short while.*
Block:	(*Pending*)
Timestamp:	(*Pending*)
From:	0xfbe29cc66a5680fbc0305f3a11369becc78ac945
To:	0x006db87188a404330d71eb1579a495b5bda7dad5
Value:	0 Ether ($0.00)
Transaction Fee:	(*Pending*)
Gas Price:	0.000000001052683402 Ether (1.052683402 Gwei)

Figure 31: Gas Price at the bottom, which was set when sending the transaction

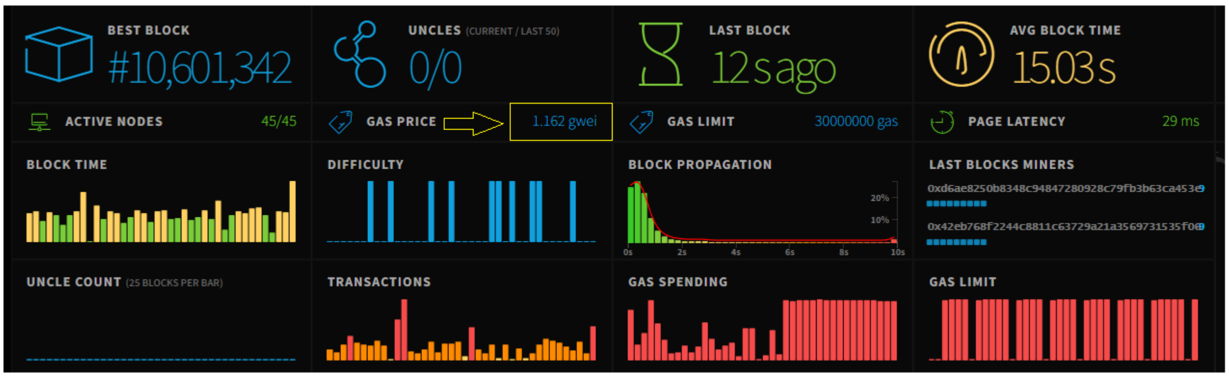
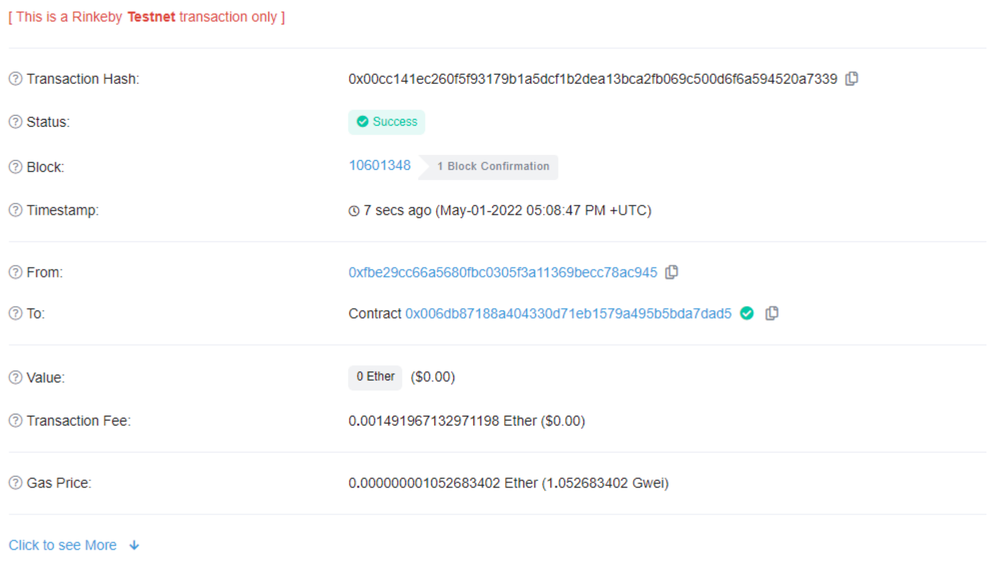


Figure 32: The current gas price in the Rinkeby Testnet, once the issue occured pointed to by the yellow arrow



[This is a Rinkeby **Testnet** transaction only]

Transaction Hash:	0x00cc141ec260f5f93179b1a5dcf1b2dea13bca2fb069c500d6f6a594520a7339
Status:	Success
Block:	10601348 1 Block Confirmation
Timestamp:	7 secs ago (May-01-2022 05:08:47 PM +UTC)
From:	0xfbe29cc66a5680fbc0305f3a11369becc78ac945
To:	Contract 0x006db87188a404330d71eb1579a495b5bda7dad5
Value:	0 Ether ($0.00)
Transaction Fee:	0.001491967132971198 Ether ($0.00)
Gas Price:	0.000000001052683402 Ether (1.052683402 Gwei)

Click to see More

Figure 33: The transaction will eventually be resolved, once the gas price falls below the gas price set in the transaction

5.5 Self-soveregin identities

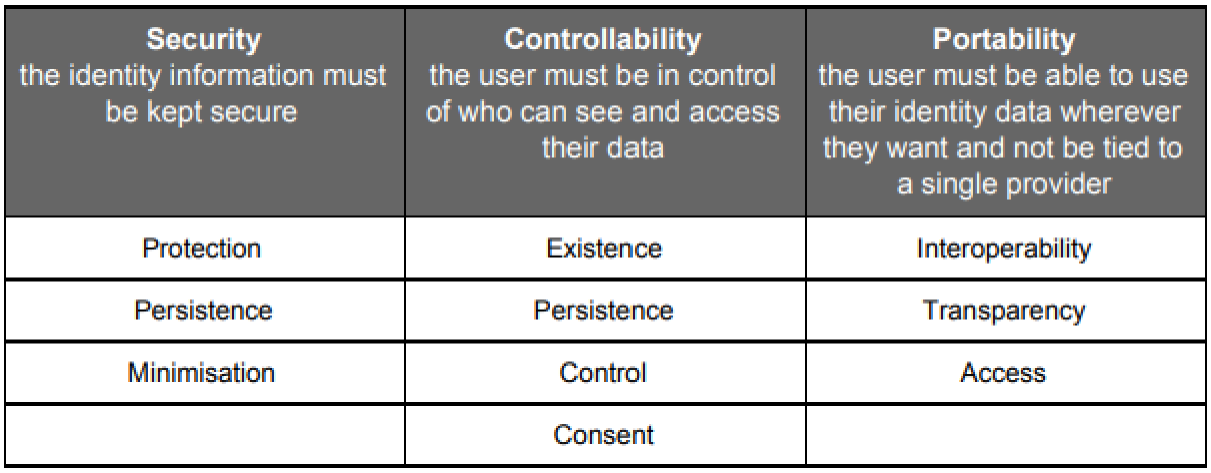
When searching for the best option for SSI, it was crucial that the solution followed Allen's Ten Principles of Self-Sovereign Identity [48], and was also using blockchains as an underlying technology to enable these ten principles better, presented in the background in subsection 2.5.

The writer decided to use Hyperledger Indy/Sovrin as the platform for enabling SSI authentication between the two blockchains, Hyperledger Fabric and Ethereum. Hyperledger Indy and Sovrin are essentially the same things. In 2017 the Sovrin Foundation transferred the open-source codebase to the Linux Foundation to become the Hyperledger Indy project [72].

The Sovrin network is a blockchain designed purely for identity. The blockchain itself is what they call public permissionless. This means that anyone can join in the ledger. Still, the consensus decision is made by Stewards, which are a select few nodes responsible for achieving global consensus inside the blockchain [73]. These nodes use a variation of the BFT protocol, called Redundant Byzantine Fault Tolerant protocol (RBTF), enabling high throughput compared to purely permissionless ledgers [72]. Currently, there are a total of 46 organizations [74], who have agreed to abide by the requirements of Sovrin, and are now running nodes operating the blockchain.

Sovrin acknowledges the need to standardize the format of an identity. Because digital credentials need to be read by a machine, it needs to be in a format that machines can understand. This means that multiple standards currently in the ecosystem are detrimental to widespread use. Just like the internet has a standard format for TCP, there will eventually need a stable standard for identity if it should see widespread adaption. For this reason, Sorvin uses the standards created from W3C, called verifiable claims [3, 72, 75]. This standard is shown in Figure 4 and relies on a certain level of trust between verifier and issuer.

Sovrin took the requirements from Allen [48] and grouped them into three different categories [3] Security, Controllability and Portability, of which they, through different solutions, attempt to satisfy.



Security the identity information must be kept secure	**Controllability** the user must be in control of who can see and access their data	**Portability** the user must be able to use their identity data wherever they want and not be tied to a single provider
Protection	Existence	Interoperability
Persistence	Persistence	Transparency
Minimisation	Control	Access
	Consent	

Figure 34: Sovrins intepretation of Allen's Ten Principles of Self-Sovereign Identity

5.5.1 Security

Sovrin provides pairwise-pseudonymous Decentralized Identifiers (DIDs) and public keys for every relationship to create a secure channel of communication between two parties, called onboarding [3, 76]. These communications will happen off-chain, typically on edge devices, as storing the encrypted data on a public blockchain could result in future decryption, resulting in the exposure of sensitive information [44, 75].

Not anyone can become an issuer inside the blockchain. To become an issuer, you must obtain the **TRUST ANCHOR** identity, which can only be given to you by a steward. These are individuals or organizations for whom there is sufficient evidence of trustworthiness to believe they will live up to the Sovrin standards; examples of this could be universities or government organizations [77]. Not allowing everyone to be an issuer makes it a lot harder to acquire a fake identity. Furthermore, because an issuer has to sign all verifiable claims with their key, it would also be easy to find out who issued a phony claim [78].

When a credential is created, a revocation ID also comes with it. This revocation ID is placed in the holder's credential, and a cryptographic accumulator is maintained on the Sovrin Blockchain. If the issuer has to remove the credential, like when the credential holder quits his job, the revocation ID is removed from the accumulator. If this credential is then later used by the holder, it will be denied because there is a check to see if the revocation ID exists in the accumulator [75].

Sovrin has developed methods where an entity can validate itself through two methods to a validator. This is through requested attributes or requested predicates. These can either be self-attested or need to be issued by an issuer of the validators choosing [44]. Requested predicates are boolean assumptions, allowing users to disclose wanted information, like the person over 18 years old, without giving away the date of birth. Additionally, this system of requesting information allows the user to himself disclose only the minimum required information needed by the verifier, satisfying the **minimalization** requirement.

5.5.2 Controllability

If a key is rotated by an issuer, which for security reasons, it should do from time to time, the key used on the holder's claim will no longer be valid. Because of the nature of blockchains, keeping track of each block will not be the case, as the claim only needs to be signed with the key used at the time of creation. This ensures persistence; even if a key is removed, the only way to remove a claim is either by deleting it from the wallet or by getting it revoked [78].

Once a claim has been issued, it is entirely in the control of the holder of the claim. An issuer will sign the claim when giving the claim to the holder, so all parties can know who issued the claim, but it is the key belonging to the holder which is used to verify the claim [78]

When a verifier wishes to get the appropriate proof needed for a transaction, the verifier must issue a proof request. A proof request is a file that describes the sort of proof that would satisfy the validation. Once the holder receives the proof request, it is up to the holder to scan their identity wallet for the relevant information or choose not to disclose the information needed [79]

Achieving a distributed consensus with the participation of many nodes ensures that the ledger state becomes practically immutable and irreversible after a certain period.

Private data is not stored on the blockchain and is entirely in the control of the holder. However, Sovrin has chosen a public permissioned schema, meaning that the consensus is left to a few stewards. This increases performance but does not make it truly decentralized, as a majority of the stewards could easier collude compared with a truly permissionless blockchain [46]

5.5.3 Portability

The Sovrin blockchain is based on open standards developed by the W3C [80], and the software is produced with an open-source license provided by the Linux foundation and made into the Hyperledger Indy project [81], satisfying the transparency requirement.

Access for Self-Sovereign identity has two critical access points:

- The storage of the SSI material: because all information is stored off-chain [76] it needs to be stored somewhere where the holder is in complete control of the data. This is mainly done with wallets. Currently, Sovrin allows users to keep their information in three different types of wallets, as well as recover the wallet should a device be lost [82]. Having multiple wallet options means that users can store their information in different wallets, removing wallet providers as a single point of failure for access.

- The access to authenticate the SSI material: To do any transactions with the SSI material, the Holder must access the Sovrin network, which should be possible at all times. The Sovrin network is a blockchain network and is, therefore, not owned by any entity; for this reason, even if a steward decided to shut down, the remaining stewards would keep the system up, making it nearly impossible for the system to go down.

From the above information, it was determined that Sovrin would be the best fit for the project. It sits in a unique situation, having developed a blockchain for the sole purpose of dealing with identities. This uniquely distinguishes it from other solutions, which usually are built on different blockchains like Ethereum or Bitcoin. By having their own blockchain, they can, in the long run, guarantee the protection promise of always prioritizing the rights of the users. Other solutions are more at the mercy of the blockchain they have built their solution on.

Sovrin is aware of the need for standards in credentials and has therefore chosen to adopt the open standards of W3C. While it is not set in stone that these standards will prevail, they currently look promising, and the awareness to invest in an open standard shows an interest in longevity for the project.

5.6 Creation of the SSI solution

This section will present the whole creation process of the SSI solution and, finally, display the experiment used to prove SSI can deanonymize data sent over the cross-chain solution.

5.6.1 Preparation

The Installation of the necessary dependencies was done following the Hyperledger Indy Build Indy SDK [83]. This allowed for the right environment to be built. Luckily there were no collisions in required dependencies between Indy and BitxHub

It was decided to use the Hyperledger Indy cluster. The Indy cluster is accessible through the Hyperledger Indy Github page [84]. The Hyperledger Indy cluster is a docker image, which was set up following a setup tutorial from medium [85].

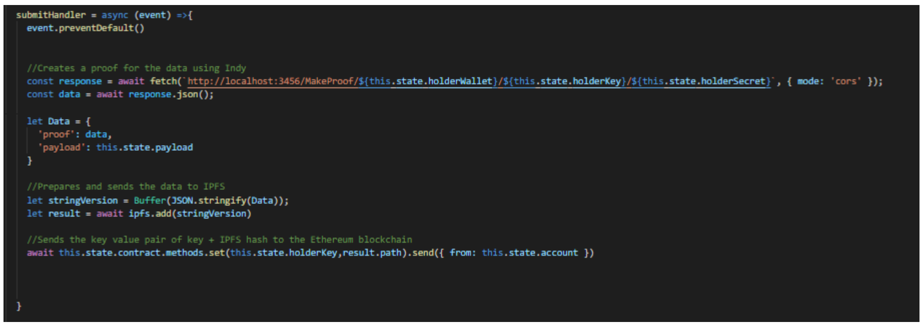
The Hyperledger Indy cluster is needed because this gives the author the ability to be a steward inside the cluster. As described, this privilege is required to provide the TRUST ANCHOR status required to issue credentials. This would not be possible in the Sovrin network, as the necessary security clearance to become an issuer makes it impossible.

For the method to communicate with the Indy cluster, it was chosen to use the NodeJS version Indy SDK for Node.js [86]. Chain-code in Hyperledger fabric 2.3 allows the fabric-contract-api for Node.js to communicate with entities off-chain. This could potentially allow verification of the proof inside the chain code for the Hyperledger fabric side.

5.6.2 Development

From the experiment done with BitxHub subsection 5.2 we know it is possible to interoperate data between Hyperledger Fabric 2.3 and Ethereum. For this reason, we will only talk about what needed to be done for it to work in the different blockchains.

For Ethereum, there were some challenges to overcome. Ethereum uses Solidity [87] as the smart contract language. However, solidity does not allow off-chain communication inside the deployed smart contract. For this reason, it was necessary to create a decentralized application (Dapp). A Dapp is essentially a regular web page or App; however, it enables transactions with the Ethereum blockchain, using the smart contracts created and deployed there. For the development of the Dapp, it was chosen to use React [88], a well-known Javascript library for building user interfaces. Further, the backend communication was handled by the Web3 package, which enables communication with smart contracts deployed on an Ethereum ecosystem. This helped us get the information that was wished to be stored on the blockchain and the credentials on the Dapp application, which could create proof and keep it on the blockchain. However, the proof is a huge JSON schema, and JSON is not a supported type by Solidity. Further stringifying the JSON schema made it too big for a String variable in Solidity. For this reason, it was concluded that we should use the Inter-Planetary File System IPFS [89]. IPFS is a peer-to-peer storage service that allows users to store information on IPFS, returning a hash, which can be used to access this information later. IPFS hashes ensure that the content has not been tampered with because any changes to the information would create a different hash. The user can therefore check if the information received matches the hash; if it does not, the information has been tampered with and is invalid.



```
submitHandler = async (event) =>{
  event.preventDefault()


  //Creates a proof for the data using Indy
  const response = await fetch(`http://localhost:3456/MakeProof/${this.state.holderWallet}/${this.state.holderKey}/${this.state.holderSecret}`, { mode: 'cors' });
  const data = await response.json();

  let Data = {
    'proof': data,
    'payload': this.state.payload
  }

  //Prepares and sends the data to IPFS
  let stringVersion = Buffer(JSON.stringify(Data));
  let result = await ipfs.add(stringVersion)

  //Sends the key value pair of key + IPFS hash to the Ethereum blockchain
  await this.state.contract.methods.set(this.state.holderKey,result.path).send({ from: this.state.account })



}
```

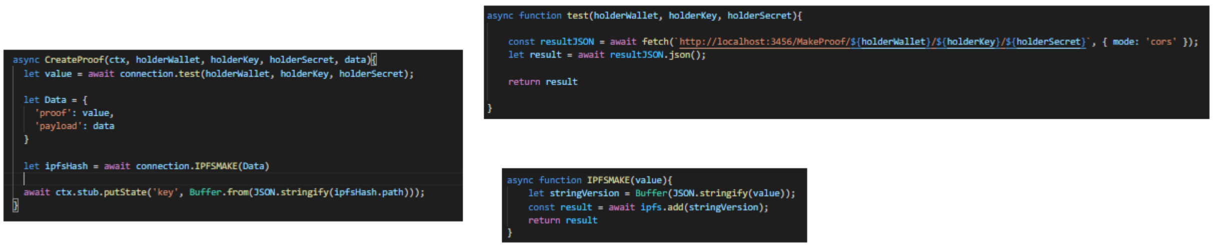
Figure 35: A code snipit from the react Dapp enabeling the user to make a proof and store the IPFS hash on the blockchain

With these changes, it was possible to create a proof, store it on IPFS, take the received hash, and store it on the Ethereum blockchain, as shown in Figure 35.

Once the changes were made in Hyperledger Fabric shown in Figure 36 and Figure 37. It was possible to send data over BitxHub and later retrieve this information to validate it.

Working with Hyperledger Fabric there were no significant challenges. Hyperledger Fabric allows developers to use commonly used programming languages, GO and Node.js being the primary ones. Further, the Node.js version of the Chaincode allows developers to communicate with off-chain sources after being packaged and deployed on the Hyperledger nodes. For this reason, it was possible to create a chain code which took the login credentials as arguments, as well as the data which the user wished to store on the blockchain. Furthermore, the use of the node-fetch npm package [90] allowed for communication with the already developed holder, issuer, and verifier, enabling the creation of the necessary proof for SSI validation.

In Figure 36, the functions in the chain code are shown, which enable the interaction with both the Ipfs and the holder's makeProof function, which will complete the sequence of actions shown in Figure 40. Once the proof is created, this is, together with the data provided, stored in a simple JSON schema, which using the IPFSMAKE function, sends this to IPFS and returns the hash to retrieve it. Finally, using the fabric-shim npm, the information is blacked on the blockchain as a key-value pair.



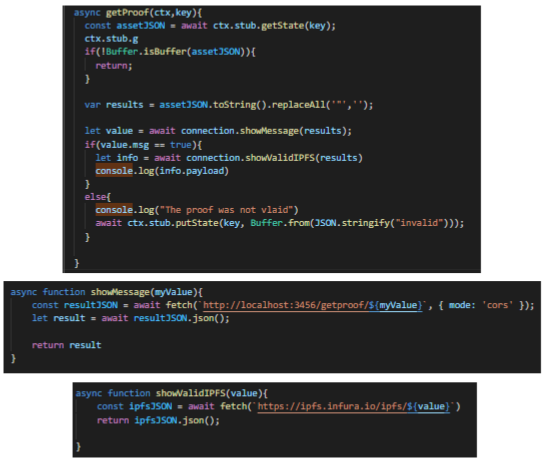
```
async CreateProof(ctx, holderWallet, holderKey, holderSecret, data){
  let value = await connection.test(holderWallet, holderKey, holderSecret);

  let Data = {
    'proof': value,
    'payload': data
  }

  let ipfsHash = await connection.IPFSMAKE(Data)

  await ctx.stub.putState('key', Buffer.from(JSON.stringify(ipfsHash.path)));
}
```

```
async function test(holderWallet, holderKey, holderSecret){

    const resultJSON = await fetch(`http://localhost:3456/MakeProof/${holderWallet}/${holderKey}/${holderSecret}`, { mode: 'cors' });
    let result = await resultJSON.json();

    return result

}
```

```
async function IPFSMAKE(value){
    let stringVersion = Buffer(JSON.stringify(value));
    const result = await ipfs.add(stringVersion);
    return result
}
```

Figure 36: Code example of how Hyperledger Fabric is able to make an SSI proof inside chain code using node-fetch and ipfs-http-client npm packages

Once information is retrieved, the getProof as shown in Figure 37 can be utilised to validate the data. The desired information is retrieved using fabric-shim's getState, which returns the IPFS hash. The IPFS hash is then used by the holders getProof shown in Figure 44. If the credential turns out to be valid, the information is fetched from IPFS and displayed.



```
async getProof(ctx,key){
  const assetJSON = await ctx.stub.getState(key);
  ctx.stub.g
  if(!Buffer.isBuffer(assetJSON)){
    return;
  }

  var results = assetJSON.toString().replaceAll('"','');

  let value = await connection.showMessage(results);
  if(value.msg == true){
    let info = await connection.showValidIPFS(results)
    console.log(info.payload)
  }
  else{
    console.log("The proof was not vlaid")
    await ctx.stub.putState(key, Buffer.from(JSON.stringify("invalid")));
  }

}
```

```
async function showMessage(myValue){
    const resultJSON = await fetch(`http://localhost:3456/getproof/${myValue}`, { mode: 'cors' });
    let result = await resultJSON.json();

    return result
}
```

```
async function showValidIPFS(value){
    const ipfsJSON = await fetch(`https://ipfs.infura.io/ipfs/${value}`)
    return ipfsJSON.json();

}
```

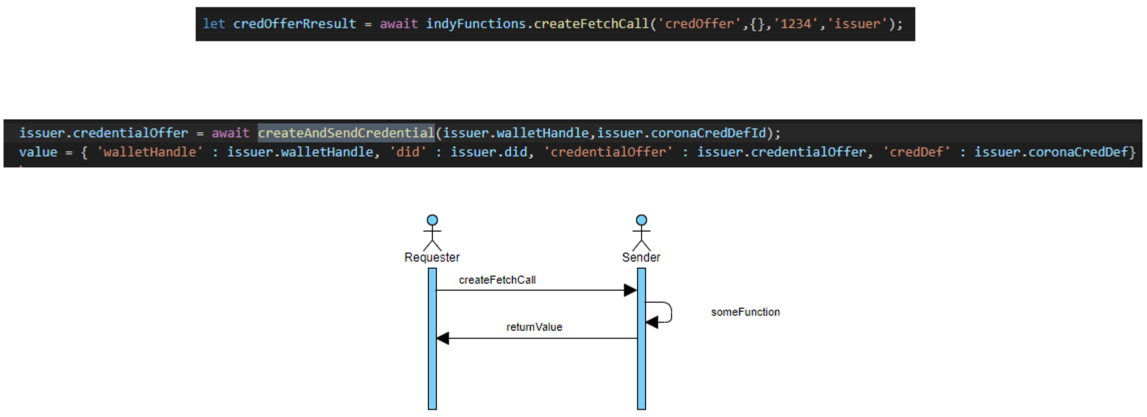
Figure 37: Code example of how Hyperledger Fabric is able retrieve SSI proof inside chain code using node-fetch and ipfs-http-client npm packages

5.6.3 Setup and experiment

From the observations made about heterogeneous blockchain interoperability, it was observed that in both SCIP and BitxHub, the cross-chain node is responsible for signing the incoming information so that it can be validated inside the receiving blockchain. This means that all information that passes through the cross-chain gateway will belong to the user responsible for maintaining the cross-chain gateway in the eyes of the receiving blockchain. If data placed through the cross-chain gateway turns out to be false, the auditing of who placed it would end at the node maintaining the cross-chain gateway, as knowledge of who placed it on the other blockchain gets abstracted when transferred. The point of this experiment is to try to deanonymize the information sent over the blockchain by attaching verifiable proof to the information sent. This proof should show that an entity that the receiving blockchain (verifier) trusts was the one who sent this information and can offer some credentials of who in that trusted entity placed this information.

The author decided to make three entities, the holder, issuer, and verifier. The issuer and verifier were constantly accessible to simulate their accessibility on the blockchain. However, the holder must log in and out of his wallet to proceed with transactions to simulate how a user would act.

To this end, it was decided that the holder, issuer, and verifier should all be accessible on the local host. The holder would need to log in and was responsible for all interaction engagement with the two other entities. For example, the holder might communicate with the verifier to communicate a verification process. This would be done by the holder calling the service port from the verifier to do the wished-for actions. An example is provided in Figure 38

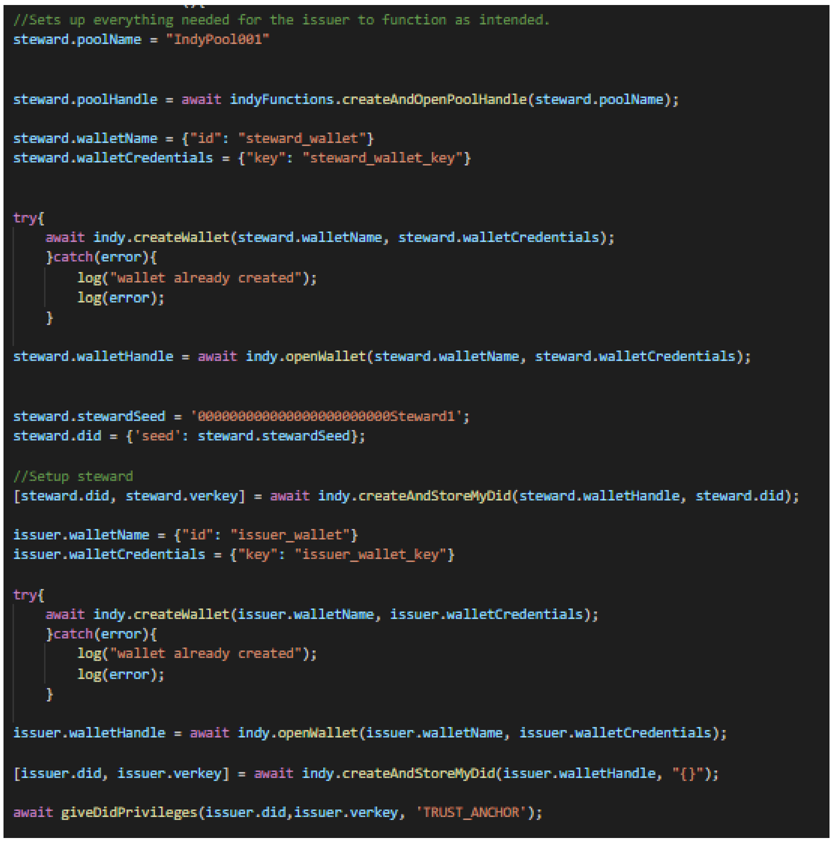


```
let credOfferRresult = await indyFunctions.createFetchCall('credOffer',{},'1234','issuer');
```

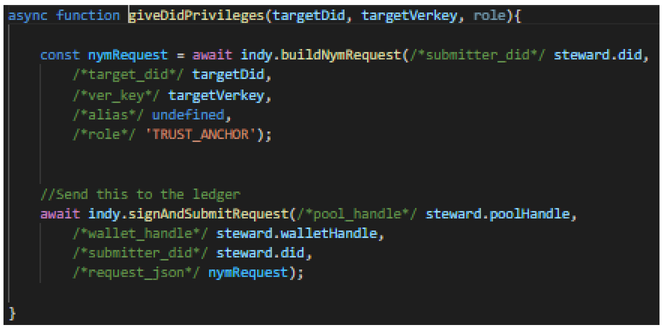
```
issuer.credentialOffer = await createAndSendCredential(issuer.walletHandle,issuer.coronaCredDefId);
value = { 'walletHandle' : issuer.walletHandle, 'did' : issuer.did, 'credentialOffer' : issuer.credentialOffer, 'credDef' : issuer.coronaCredDef}
```

Figure 38: Normal request sequence between two entities

Once the Indy cluster had been started, all three entities, issuer, holder, and verifier, needed to be started. This is to first connect to the Indy cluster, create their respective wallets and DIDs, hold the created credentials, and communicate with the other entities to sign certificates. The issuer will also create a wallet and DID for the steward in the initiation phase. This is needed so the steward can give the issuer the TRUST ANCHOR privilege to the issuer. The process is shown in Figure 39. Finally, the issuer will create schemas and credential definitions needed for the later creation of credentials.



```
//Sets up everything needed for the issuer to function as intended.
steward.poolName = "IndyPool001"


steward.poolHandle = await indyFunctions.createAndOpenPoolHandle(steward.poolName);

steward.walletName = {"id": "steward_wallet"}
steward.walletCredentials = {"key": "steward_wallet_key"}


try{
    await indy.createWallet(steward.walletName, steward.walletCredentials);
    }catch(error){
        log("wallet already created");
        log(error);
    }

steward.walletHandle = await indy.openWallet(steward.walletName, steward.walletCredentials);


steward.stewardSeed = '000000000000000000000000Steward1';
steward.did = {'seed': steward.stewardSeed};

//Setup steward
[steward.did, steward.verkey] = await indy.createAndStoreMyDid(steward.walletHandle, steward.did);

issuer.walletName = {"id": "issuer_wallet"}
issuer.walletCredentials = {"key": "issuer_wallet_key"}

try{
    await indy.createWallet(issuer.walletName, issuer.walletCredentials);
    }catch(error){
        log("wallet already created");
        log(error);
    }

issuer.walletHandle = await indy.openWallet(issuer.walletName, issuer.walletCredentials);

[issuer.did, issuer.verkey] = await indy.createAndStoreMyDid(issuer.walletHandle, "{}");

await giveDidPrivileges(issuer.did,issuer.verkey, 'TRUST_ANCHOR');
```



```
async function giveDidPrivileges(targetDid, targetVerkey, role){

    const nymRequest = await indy.buildNymRequest(/*submitter_did*/ steward.did,
        /*target_did*/ targetDid,
        /*ver_key*/ targetVerkey,
        /*alias*/ undefined,
        /*role*/ 'TRUST_ANCHOR');


    //Send this to the ledger
    await indy.signAndSubmitRequest(/*pool_handle*/ steward.poolHandle,
        /*wallet_handle*/ steward.walletHandle,
        /*submitter_did*/ steward.did,
        /*request_json*/ nymRequest);

}
```

Figure 39: The code first connects to the indy cluster, and then creates credentials for the Steward and issuer, before finally the issuer is given the Trust ANCHOR priviledge

With the finished code implemented, two steps must be done for the SSI credentials to be created, the proof to be made, and the proof to be verified. These two methods are makeProof and getProof, shown in Figure 40 and Figure 44 respectively.

makeProof takes the wallet credentials of the users and the desired data, which should be placed on the blockchain. From here, we will explain the different steps for makeProof in the sequence diagram Figure 40.

- **credOffer:** is used simply to initiate the sending of a credential offer to the holder. The issuer will then issue an identity based on a credential definition that the company owns. returning **SendCredentialOffer**
- **credReq:** we assume that the holder indeed wants a credential from Company One. With his DID, wallet, and MasterSecret, the holder will sign a request to create a credential based on the credential definition presented and send it to the issuer.
- **IssuerCreateCredential:** The issuer will now populate the credential definition, which was asked for and sign it. This is then returned to the holder.
- **StoreCredential:** The holder will take the credential and store it in his wallet.

  Some of the code used from credoffer to StoreCredential is shown in Figure 41
- **AskRequirements:** Will ask the verifier for the requirements needed to satisfy their authentication and return this to the holder. An example is shown in Figure 42, where the requested attributes, as well as the requested predicates, are shown. The user must also provide the information based on the restrictions, in this case the credDefID, which was selected.
- **PopulateProof:** The population of proof takes the stated requirements shown in Figure 42 and looks inside the holders wallet to find the necessary attributed, which satisfies the requirements to create a proof, consisting of the stated requirements, the requested credentials found in the wallet, the schemas, credential definitions, and revocation states used to create the proof. the code used to Populate the proof is shown in Figure 43

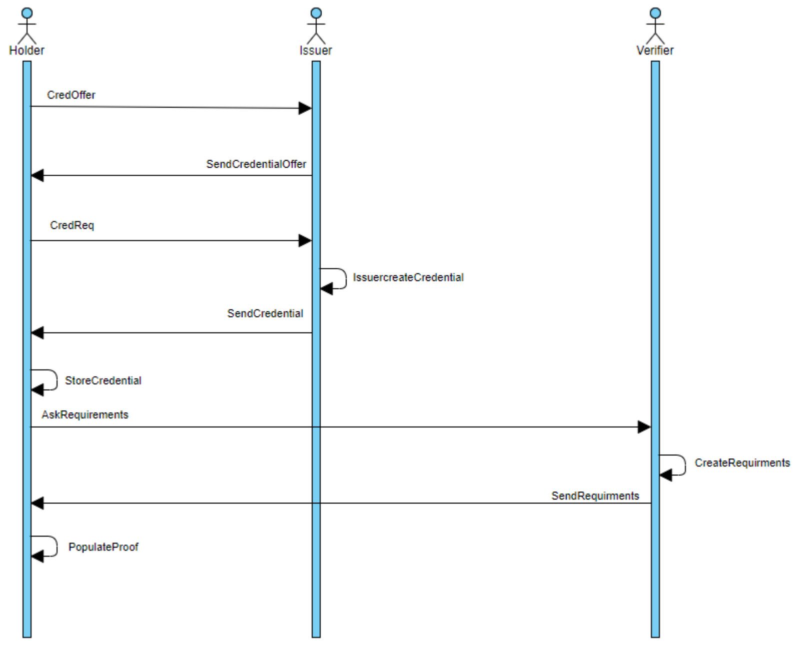
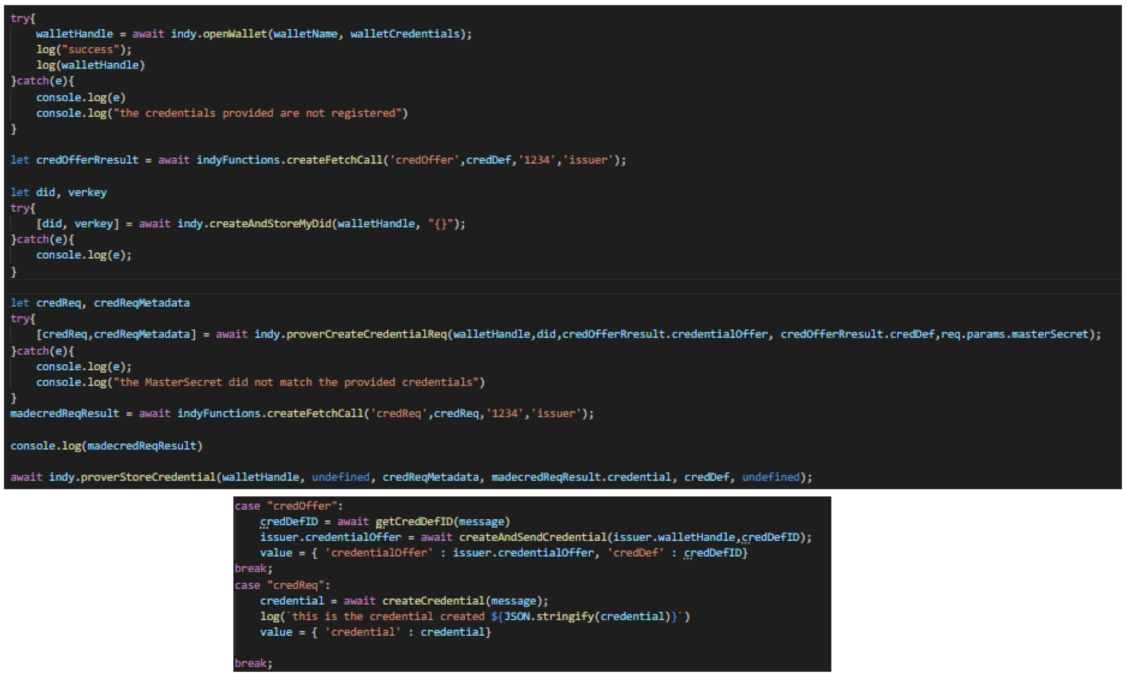


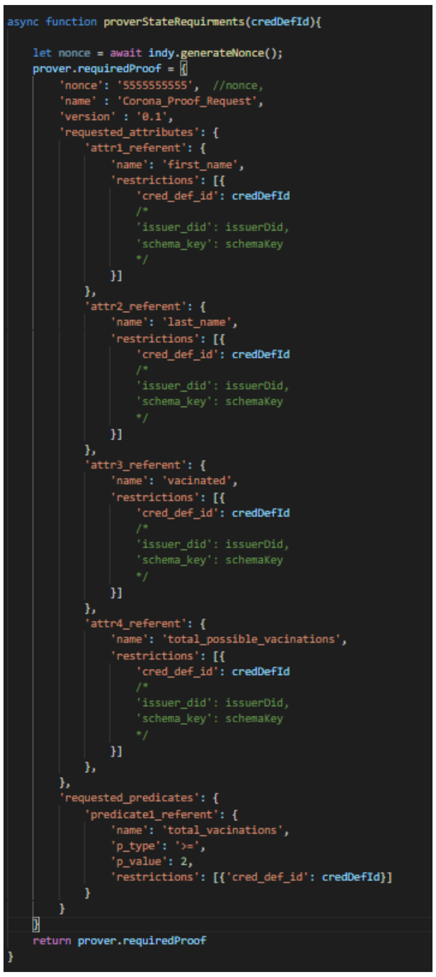
Figure 40: The complete sequence from makeProof



```
try{
    walletHandle = await indy.openWallet(walletName, walletCredentials);
    log("success");
    log(walletHandle)
}catch(e){
    console.log(e)
    console.log("the credentials provided are not registered")
}

let credOfferRresult = await indyFunctions.createFetchCall('credOffer',credDef,'1234','issuer');

let did, verkey
try{
    [did, verkey] = await indy.createAndStoreMyDid(walletHandle, "{}");
}catch(e){
    console.log(e);
}

let credReq, credReqMetadata
try{
    [credReq,credReqMetadata] = await indy.proverCreateCredentialReq(walletHandle,did,credOfferRresult.credentialOffer, credOfferRresult.credDef,req.params.masterSecret);
}catch(e){
    console.log(e);
    console.log("the MasterSecret did not match the provided credentials")
}
madecredReqResult = await indyFunctions.createFetchCall('credReq',credReq,'1234','issuer');

console.log(madecredReqResult)

await indy.proverStoreCredential(walletHandle, undefined, credReqMetadata, madecredReqResult.credential, credDef, undefined);
```

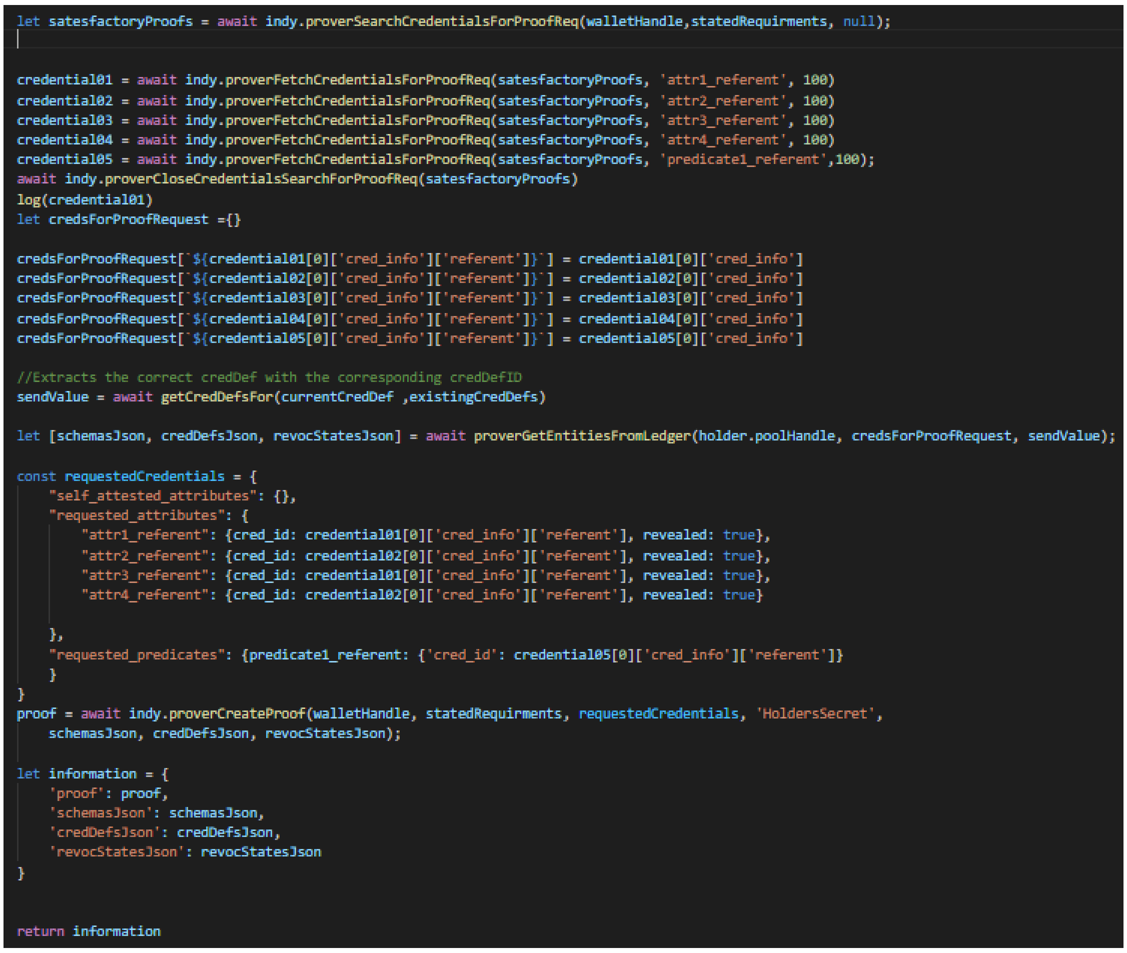
```
case "credOffer":
    credDefID = await getCredDefID(message)
    issuer.credentialOffer = await createAndSendCredential(issuer.walletHandle,credDefID);
    value = { 'credentialOffer' : issuer.credentialOffer, 'credDef' : credDefID}
break;
case "credReq":
    credential = await createCredential(message);
    log(`this is the credential created ${JSON.stringify(credential)}`)
    value = { 'credential' : credential}

break;
```

Figure 41: Code snippits from makeProof showing parts of the code from credOffer to StoreCredentials



```
async function proverStateRequirments(credDefId){

    let nonce = await indy.generateNonce();
    prover.requiredProof = {
        'nonce': '5555555555',  //nonce,
        'name' : 'Corona_Proof_Request',
        'version' : '0.1',
        'requested_attributes': {
            'attr1_referent': {
                'name': 'first_name',
                'restrictions': [{
                    'cred_def_id': credDefId
                    /*
                    'issuer_did': issuerDid,
                    'schema_key': schemaKey
                    */
                }]
            },
            'attr2_referent': {
                'name': 'last_name',
                'restrictions': [{
                    'cred_def_id': credDefId
                    /*
                    'issuer_did': issuerDid,
                    'schema_key': schemaKey
                    */
                }]
            },
            'attr3_referent': {
                'name': 'vacinated',
                'restrictions': [{
                    'cred_def_id': credDefId
                    /*
                    'issuer_did': issuerDid,
                    'schema_key': schemaKey
                    */
                }]
            },
            'attr4_referent': {
                'name': 'total_possible_vacinations',
                'restrictions': [{
                    'cred_def_id': credDefId
                    /*
                    'issuer_did': issuerDid,
                    'schema_key': schemaKey
                    */
                }]
            },
        },
        'requested_predicates': {
            'predicate1_referent': {
                'name': 'total_vacinations',
                'p_type': '>=',
                'p_value': 2,
                'restrictions': [{'cred_def_id': credDefId}]
            }
        }
    }
    return prover.requiredProof
}
```

Figure 42: an example of a requierment



```
let satesfactoryProofs = await indy.proverSearchCredentialsForProofReq(walletHandle,statedRequirments, null);


credential01 = await indy.proverFetchCredentialsForProofReq(satesfactoryProofs, 'attr1_referent', 100)
credential02 = await indy.proverFetchCredentialsForProofReq(satesfactoryProofs, 'attr2_referent', 100)
credential03 = await indy.proverFetchCredentialsForProofReq(satesfactoryProofs, 'attr3_referent', 100)
credential04 = await indy.proverFetchCredentialsForProofReq(satesfactoryProofs, 'attr4_referent', 100)
credential05 = await indy.proverFetchCredentialsForProofReq(satesfactoryProofs, 'predicate1_referent',100);
await indy.proverCloseCredentialsSearchForProofReq(satesfactoryProofs)
log(credential01)
let credsForProofRequest ={}

credsForProofRequest[`${credential01[0]['cred_info']['referent']}`] = credential01[0]['cred_info']
credsForProofRequest[`${credential02[0]['cred_info']['referent']}`] = credential02[0]['cred_info']
credsForProofRequest[`${credential03[0]['cred_info']['referent']}`] = credential03[0]['cred_info']
credsForProofRequest[`${credential04[0]['cred_info']['referent']}`] = credential04[0]['cred_info']
credsForProofRequest[`${credential05[0]['cred_info']['referent']}`] = credential05[0]['cred_info']

//Extracts the correct credDef with the corresponding credDefID
sendValue = await getCredDefsFor(currentCredDef ,existingCredDefs)

let [schemasJson, credDefsJson, revocStatesJson] = await proverGetEntitiesFromLedger(holder.poolHandle, credsForProofRequest, sendValue);

const requestedCredentials = {
    "self_attested_attributes": {},
    "requested_attributes": {
        "attr1_referent": {cred_id: credential01[0]['cred_info']['referent'], revealed: true},
        "attr2_referent": {cred_id: credential02[0]['cred_info']['referent'], revealed: true},
        "attr3_referent": {cred_id: credential01[0]['cred_info']['referent'], revealed: true},
        "attr4_referent": {cred_id: credential02[0]['cred_info']['referent'], revealed: true}

    },
    "requested_predicates": {predicate1_referent: {'cred_id': credential05[0]['cred_info']['referent']}}
    }
}
proof = await indy.proverCreateProof(walletHandle, statedRequirments, requestedCredentials, 'HoldersSecret',
    schemasJson, credDefsJson, revocStatesJson);

let information = {
    'proof': proof,
    'schemasJson': schemasJson,
    'credDefsJson': credDefsJson,
    'revocStatesJson': revocStatesJson
}


return information
```

Figure 43: code snippit from populate proof

With the steps completed, the proof is now together with the data sent to IPFS, where it is stored. The hash returned is then stored on the blockchain of either Hyperledger or Ethereum, as shown in Figure 36 and Figure 35 respectively.

When sent over BitxHub as presented in subsection 5.2 the verification of the proof must be done. In both cases, the data verification can only be done after the information has been placed on the blockchain, signed by either the Hyperledger user responsible for handling the Cross-chain gateway or the Ethereum wallet connected to the Ethereum Cross-chain gateway.

On the Ethereum side, the verification must happen off-chain. The connection to IPFS and the verifier is impossible to reach over Solidity smart contracts. Therefore the IPFS hash must be retrieved from the Solidity smart contract. However, on Hyperledger Fabric, using Node.js smart contracts, it is possible to connect to get the IPFS, send it to the verifier, which extracts value from the IPFS hash and validates the proof, to then send it back. If the proof turns out to be true, the Chain code can then extract the IPFS information and display the payload or if wished for, information stored in the proof about the user who placed the information on the other chain. this is shown in Figure 37, and a sequence diagram for the getProof is shown in Figure 44.

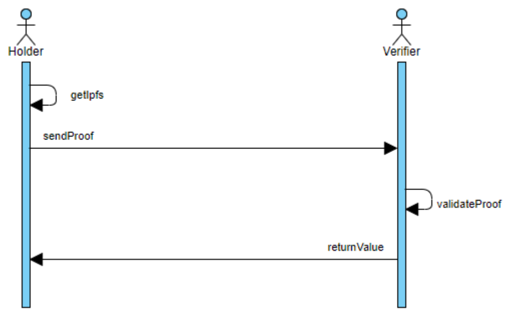
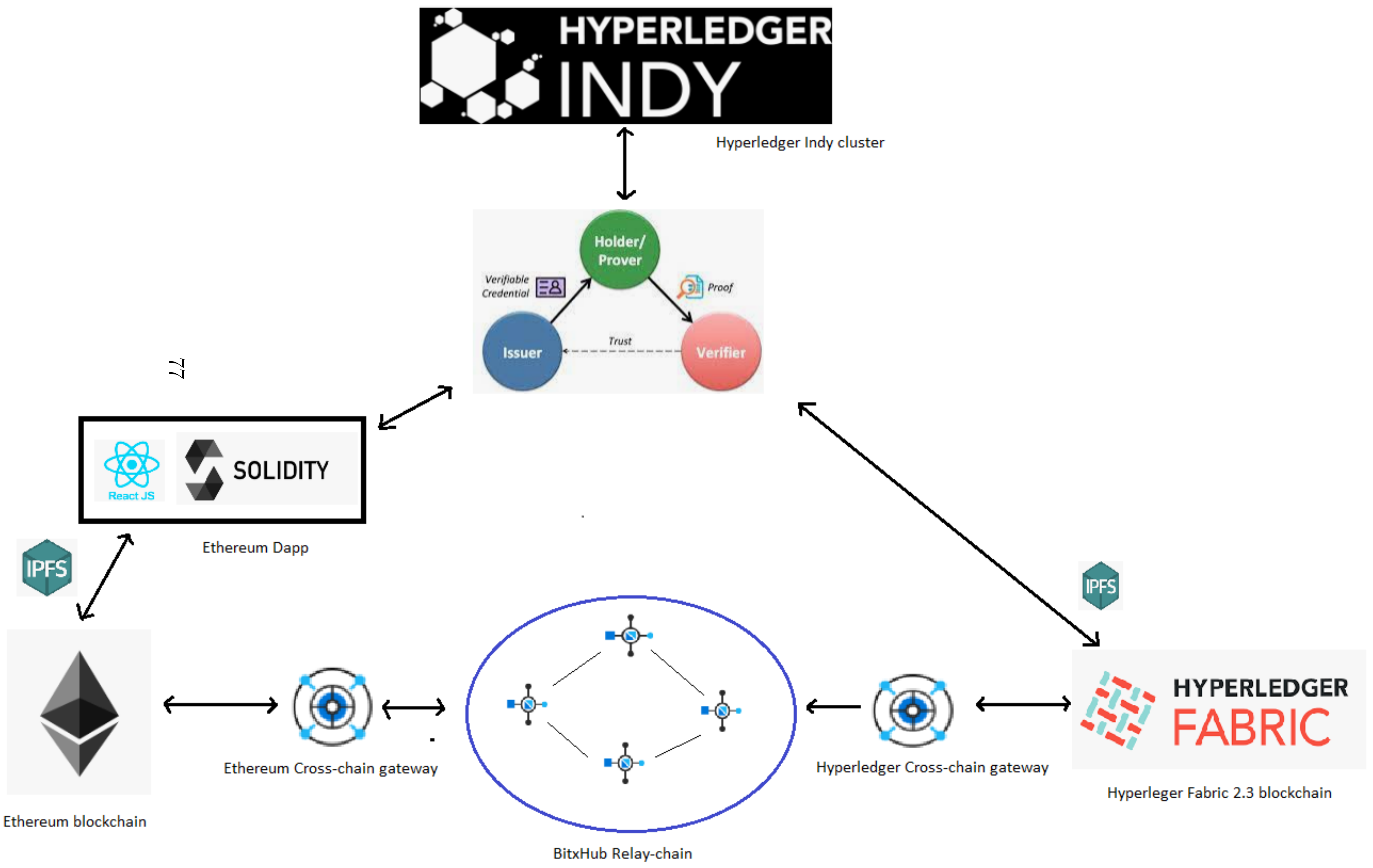


Figure 44: The complete sequence from getProof

The experiment proves it is possible to send a verifiable SSI proof and the data over the BitxHub Cross-chain gateway. The verifier can himself choose the credDef required to satisfy the verification they desire and therefore upholds the requirement that the verifier can choose a trusted proof. The other blockchain is also able to verify this proof, off-chain for Ethereum and on-chain for Hyperledger Fabric. Therefore, both chains can use this proof to audit information by deanonymizing the information sent over. In Figure 45 we present the compleate coded solution combining the two solutions BitxHub, and SSI.

The results show a possibility to verify data being sent over the heterogeneous blockchain using SSI. The needed information could be changed based on what was needed for proof. And most importantly, a proof could be shown to be valid or invalid. The extra time it would take to authenticate a proof was only 1.39 seconds.


HYPERLEDGER INDY
Hyperledger Indy cluster
Holder/ Prover
Verifiable Credential
Proof
Issuer
Trust
Verifier
React JS
SOLIDITY
Ethereum Dapp
IPFS
IPFS
Ethereum blockchain
Ethereum Cross-chain gateway
BitxHub Relay-chain
Hyperledger Cross-chain gateway
HYPERLEDGER FABRIC
Hyperleger Fabric 2.3 blockchain

6 Discussion

The discussion chapter will describe the contribution of the findings from section 5 and explain new insights that emerged as a result of our research. The discussion will be broken down into the three research questions posed in this thesis.

6.1 Interoperating data between two heterogeneous blockchains

A fair amount of work has been done on cross-chain interoperability, most of it being covered in [49]. However, from this literature review, there was no information about the current existing interoperability solutions that could interoperate heterogeneous data. The world economic forum also published a paper stating the need for heterogeneous interoperability [16]. However, their solution at the data of publishing was using APIs, further accentuating the need for this information in literature. Finally, in our pre-study, we found solutions that were promising for SCM interoperability. Figuring out if the solutions could interoperate data between heterogenous blockchains was not something we had considered in the pre-study, but became very important in the thesis.

Our findings highlight the solutions that are currently able to interoperate data between heterogeneous blockchains and see if they pass other criteria needed for SCM blockchains. Here it was found that two solutions currently exist which can interoperate data, those being BitxHub subsection 5.1 and SCIP subsection 8.4. It was argued that while two solutions exist, BitxHub is currently the more attractive solution. BitxHub provides better security than SCIP, which seems not to have considered it much under development. BitxHub is also an open-source project with an active community developing solutions on it, promoting longevity for the project. SCIP seems to have been created as a Ph.D. project and has not seen any development since then.

A working solution between Hyperledger 2.3 and Ethereum was also developed. The development process was documented, and the working solution will be linked to this thesis.

While the literature review was done extensively during the pre-study, due to sickness, the pre-study is by now one year old. BitxHub was found as a possible new solution to the interoperability solution when the master thesis was started again. This shows rapid development in the blockchain ecosystem; for this reason, there might be more solutions out there, which slipped through the cracks, but to our best knowledge, the two found solutions are the current only solutions. Hyperledger is, however, developing their own solution to this problem which they call Hyperledger Cactus subsection 8.6. Cactus is still in the early stages of development and was therefore not mentioned in this thesis. However with their current proposed capabilities and solutions, this project might, in the future, become a desirable solution competing with BitxHub.

6.2 Performance and issues of the selected interoperability solution

The performance of heterogeneous blockchains was a topic not covered by any literature, which presented a heterogeneous interoperability solution. BitxHub 5.1 mentioned "high performance" in their conclusion [19], while SCIP 8.4 does not mention anything about performance. With the creation of a functioning BitxHub solution between the heterogenous blockchains Hyperledger Fabric 2.3 and Ethereum, we shed some light on the performance and observed issues found while conducting the experiments.

The results found that the current average transaction takes 15.78 seconds from first sending the transaction request to receiving the desired information. It was observed that the size of the data sent was not a significant issue for the transaction speed; however, the throughput was hindered by how the Cross-Chain Gateway is developed to handle multiple transactions. A linear build-up was observed in processing time, indicating the Cross-Chain Gateway as a bottleneck. Two problems were also observed related to the Ethereum side of the Cross-chain gateway, which could negatively impact the transaction time. These are the double emit bug and gas price issue described in the results.

For a wider adaption of interoperability for data, the transaction speed (tps) needs to be increased. While other blockchains systems talk about 10 000 tps [49], this interoperability solution can only get out one transaction every 15 seconds. It was proven that the limiting factor for a single transaction was using Ethereum as one of the blockchains. The Fabric to Fabric solution only used 10.36 seconds. Having Ethereum in the interoperability solution will always make it the limiting factor. Improving the cross-chain architecture would not increase speed, as the bottleneck already is Ethereum. For this reason, for a higher tps, the Cross-Chain gateway must be able to process multiple transactions at once. A possible solution to this would be to adopt some ideas from Polkadot's subsection 8.1 XCMP, currently in development [91]. Instead of needing the first transaction to be received before the next transaction is sent, all transactions are stored in one of two queues, the inbound and outbound queues. Once these transactions are then acted on, the queues are updated to indicate what has and has not yet been received. This would allow for multiple transactions to be sent without waiting for each transaction before the next is created. Allowing for multiple transactions to be handled at once would also reduce the impact of the gas price issue presented in the results. With numerous transactions allowed, a transaction that would take a long time to be received would not stall the whole system until this information is received.

Another problem is that the Cross-chain gateway currently is a single node leading to two problems.

- Signature: Currently, Permissionless blockchains in BitxHub need a validation signature that the information exists on the blockchain. Permissionless blockchains don't have nodes to sign these transactions, which falls on the cross-chain gateway. However, since there is currently only one node, this node has all the power to validate the information, leading to a potential for malicious behavior.

- Single Point of Failure: Currently, if the node stops working, this will lead to no accessibility of the blockchain it is serving, leading to downtime.

A possible solution would be to have multiple nodes communicating together in a cluster like suggested by Hyperledger Cactus subsection 8.6. Having more nodes would solve these two issues.

Combining these two suggestions would, in theory, create a much more robust interoperability system. However, the changes which would have to be implemented in the BitxHub system. How demanding the implementation of the changes would be is unknown to us.

6.3 Self-sovereign used to deanonymize cross-chain data between two heterogeneous blockchains

From the related work, it was found that for academia, SSI is still mainly in a phase where formalizations of definitions, improvements, and justification for their existence are in focus. For industry, SSI is primarily used to showcase use-cases to prove a need for the product. This solution similarly provides a use case for SSI. However, the environment it is used in is new. Using SSI to deanonymize information sent over two heterogenous blockchains over an interoperability solution, to our best knowledge, is a unique and new case.

The approach has been developed and demonstrated to work in the Interoperability case, using BitxHub to interoperate data between Ethereum and Hyperledger Fabric 2.3.

With these findings, it is possible to solve the anonymity issue when two blockchains don't have the same identity schemes, which is common to heterogeneous blockchains. For heterogeneous blockchains, the observed tendency from SCIP [9] and BitxHub [19] is to let an entity sign the information on their behalf. This creates data stored on the blockchain which can't be correctly audited. With SSI, adding proof to the data transferred, it is again possible to figure out who placed this data on the blockchain. The results show that with the provided proof, it is possible to authenticate who set this data and determine if the proof provided is valid or not.

The created code was only a minimum viable proof. The main focus was figuring out whether it could make SSI communicate with Hyperledger Fabric and Ethereum over a cross-chain gateway. To make the product attractive for the industry, a more flashed-out product would have to be created, using supported wallets and SSI created in either the Staging network or the main

Sovrin network. If this would create new challenges, threatening the validity of these findings is unknown.

The most significant limitation to the findings discussed is that validating the information sent over can only be done after storing the data in the blockchain. This allows the proof provided with the data to only be used for security/auditing checks after the data has been placed on the chain. However, we can suppose the proof could be checked before the information was placed on the blockchain. In that case, this could become an extra layer of security in cross-chain messaging, where information sent is validated to come from a desirable source before it is placed on the ledger. For Ethereum, which uses Solidity, this seems impossible; however, for Hyperledger Fabric, this could become possible if the code in BitxHub would be changed. The BitxHub chain code is currently written in Golang, while the SSI chain code needs to be built in Node.js to talk to off-chain sources. This forces two separate chain codes to be deployed on the peers. The Golang chain code must first retrieve the information, and later the Hyperledger chain code can validate the data. Suppose it were possible to change the Golang chain code to become Node.js chain code. If this would not lead to any communication issues between the chain code and BitxHub's Hyperledger Cros-chain gateway, everything could be written in Node.js, enabling authentication of the data retrieved from Ethereum before it was placed on the blockchain ledger.

7 Conclusions

Heterogenous interoperability methods for blockchains were the focus of this thesis. SSI was used to solve the problem of authentication.

A literature study investigated existing interoperability methods. Only SCIP and BitxHub are able to perform heterogeneous interoperability. BitxHub was selected as the best choice, because it is open source and being further developed and maintained, while SCIP seems to be a dormant program.

A working solution was successfully created allowing interoperability of data between the two most popular heterogeneous blockchains Ethereum and Hyperledger Fabric 2.3, using BitxHub. the code for the solution can be found here.

The created interoperability solution was used to evaluate BitxHub on performance. BitxHubs transaction speed is 15.78 seconds. BitxHub is, however, not the reason for this slow transaction speed. Ethereum limits the performance. The speed cannot be increased, because BitxHub is configured to only handle one transaction at a time. Trying to increase the throughput, only results in a higher transaction time.

Bitxhub allows, however, a big amount of data to be transferred at once. Transactions could be sent with 30KB data. The only risk would be that a transaction time would take 30 seconds instead of 15, however, this did not occur too often.

Finally, two issues were discovered, these being the double emit bug, and the gas price issue. The double emit bug makes it impossible to transfer data over the cross-chain solution, while the gas price issue stalls the solution until the transaction with a low gas price is resolved.

From the two heterogeneous interoperability methods, it was found that information being sent over heterogenous cross-chain gateways gets anonymized. Data sent over the cross-chain gateway no longer contains information about who placed it on the blockchain. A solution was created using SSI, creating a verifiable proof of the sent data. The code can be found here. Using Hyperledger Indy together with IPFS made it possible to create proofs, which could be saved on either blockchain and sent over BitxHub to be verified later.

Including a verifiable proof deanonymized the transmitted data, allowing for better auditing. However, it is only possible to verify the information after it has been placed on the blockchain. Solidity does not allow for off-chain transactions happening on the smart contract. For Hyperledger there is, however, a possibility to make on-chain authentication happen.

BitxHub as an interoperability solution is currently too slow for commercial Blockchain-based supply chain systems. Therefore, future work should be focused on increasing the transactions per second for BitxHub. For this to be possible, multiple transactions must be sent over simultaneously. The Cross-chain gateway's configurations must be changed. Currently the cross-chain gateway is also a single point of failure, only consisting of a single node, preferably this also needs to be changed to increase security.

SSI could be improved in the future by enabling Hyperledger fabric to validate the verifiable proof before the data is stored, and execute appropriate action depending on the validity of the information sent over. This would vastly increase the legitimacy to use SSI as an authentication method for heterogeneous interoperability methods, as the security evaluation would (at least on the Hyperledger side) happen before the information is placed and not after, as it currently is.

References

[1] H. Fabric, "How fabric networks are structured." https://hyperledger-fabric.readthedocs.io/en/latest/network/network.html, 2020. Accessed 23.01.2021.

[2] M. S. Ferdous, F. Chowdhury, and M. O. Alassafi, "In search of self-sovereign identity leveraging blockchain technology," *IEEE Access*, vol. 7, pp. 103059–103079, 2019.

[3] S. Foundation, "Sovrin™: A protocol and token for self-sovereign identity and decentralized trust." https://sovrin.org/library/sovrin-protocol-and-token-white-paper/, 2018. Accessed 23.01.2021.

[4] G. Wood, "Polkadot: Vision for a heterogeneous multi-chain framework," *White Paper*, 2016.

[5] J. Burdges, "Availability and validity." https://w3f-research.readthedocs.io/en/latest/polkadot/Availability_and_Validity.html, 2020. Accessed 09.02.2021.

[6] J. Petrowski, "Polkadot consensus part 3: Babe." https://medium.com/polkadot-network/polkadot-consensus-part-3-babe-dcc2e0dd8878, 2019. Accessed 07.02.2021.

[7] Hedera, "Webinar: Using the hedera consensus service with hyperledger fabric." https://www.youtube.com/watch?v=elWRmHqRoww&t=585s, 2020. Accessed 23.01.2021.

[8] Interledger, "Interledgerarchitecture." https://interledger.org/rfcs/0001-interledger-architecture/#connectors, 2020. Accessed 18.01.2021.

[9] G. Falazi, U. Breitenbücher, F. Daniel, A. Lamparelli, F. Leymann, and V. Yussupov, "Smart contract invocation protocol (scip): A protocol for the uniform integration of heterogeneous blockchain smart contracts," in *International Conference on Advanced Information Systems Engineering*, pp. 134–149, Springer, 2020.

[10] E. Abebe, D. Behl, C. Govindarajan, Y. Hu, D. Karunamoorthy, P. Novotny, V. Pandit, V. Ramakrishna, and C. Vecchiola, "Enabling enterprise blockchain interoperability with trusted data transfer (industry track)," in *Proceedings of the 20th International Middleware Conference Industrial Track*, pp. 29–35, 2019.

[11] Hyperledger, "Hyperledger cactus whitepaper." https://github.com/hyperledger/cactus/blob/main/whitepaper/whitepaper.md, 2020. Accessed 12.01.2021.

[12] S. Yadav and S. P. Singh, "Blockchain critical success factors for sustainable supply chain," *Resources, Conservation and Recycling*, vol. 152, p. 104505, 2020.

[13] D. L. Chaum, *Computer Systems established, maintained and trusted by mutually suspicious groups*. Electronics Research Laboratory, University of California, 1979.

[14] S. Nakamoto and A. Bitcoin, "A peer-to-peer electronic cash system," *Bitcoin.–URL: https://bitcoin. org/bitcoin. pdf*, vol. 4, 2008.

[15] V. Buterin *et al.*, "Ethereum: A next-generation smart contract and decentralized application platform," 2014.

[16] L. P. Nadia Hewett, Margi van Gogh, "Inclusive deployment of blockchain for supply chains: Part 6 – a framework for blockchain interoperability," 2020.

[17] L. Baird, M. Harmon, and P. Madsen, "Hedera: A public hashgraph network & governing council," *White Paper*, vol. 1, 2019.

[18] E. S. Stefan Thomas, "A protocol for interledger payments," 2015.

[19] H. Q. T. Co., "Bitxhub whitepaiper inter-blockchain technology platform v1.0.0." https://upload.hyperchain.cn/BitXHub%20Whitepaper.pdf, 2019. Accessed 12.01.2021.

[20] J. Burdges, A. Cevallos, P. Czaban, R. Habermeier, S. Hosseini, F. Lama, H. K. Alper, X. Luo, F. Shirazi, A. Stewart, *et al.*, "Overview of polkadot and its design considerations," *arXiv preprint arXiv:2005.13456*, 2020.

[21] Hashport, "Hashport." https://www.hashport.network/, 2021. Accessed 29.04.2022.

[22] P. A. M. Hanns Christian Hanebeck, Nadia Hewett, "Inclusive deployment of blockchain for supply chains: Part 3 – blockchain-based supply chainsystem (scm) – which one is right for you?," 2019.

[23] H. Fabric, "Transaction flow." https://hyperledger-fabric.readthedocs.io/en/release-2.2/txflow.html.

[24] N. H. Sheila Warren, Christoph Wolff, "Inclusive deployment of blockchain for supply chains: Part 1 – introduction," 2019.

[25] E. A. et al, "Hyperledger fabric." https://hyperledger-fabric.readthedocs.io/en/release-2.2/whatis.html#hyperledger-fabric, 2020. Accessed 22.01.2021.

[26] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, "An overview of blockchain technology: Architecture, consensus, and future trends,"

in *2017 IEEE international congress on big data (BigData congress)*, pp. 557–564, IEEE, 2017.

[27] H. Fabric, "Registering and enrolling identities with a ca." https://hyperledger-fabric-ca.readthedocs.io/en/latest/deployguide/use$_C$*A.html*.

[28] H. Fabric, "The ordering service." https://hyperledger-fabric.readthedocs.io/en/latest/orderer/ordering_service.html, 2020. Accessed 23.01.2021.

[29] H. Fabric, "Glossary." https://hyperledger-fabric.readthedocs.io/en/latest/glossary.html#endorsement-policy, 2020. Accessed 23.01.2021.

[30] H. Fabric, "Peers." https://hyperledger-fabric.readthedocs.io/en/release-2.2/peers/peers.html, 2020. Accessed 23.01.2021.

[31] P. A. M. Hanns Christian Hanebeck, Nadia Hewett, "Inclusive deployment of blockchain for supply chains part 3 – public or private blockchains – which one is right for you?," World Economic Forum, 2019.

[32] B. Pillai, K. Biswas, and V. Muthukkumarasamy, "Blockchain interoperable digital objects," in *International Conference on Blockchain*, pp. 80–94, Springer, 2019.

[33] V. Buterin, "Chain interoperability," *R3 Research Paper*, 2016.

[34] A. Singh, K. Click, R. M. Parizi, Q. Zhang, A. Dehghantanha, and K.-K. R. Choo, "Sidechain technologies in blockchain networks: An examination and state-of-the-art review," *Journal of Network and Computer Applications*, vol. 149, p. 102471, 2020.

[35] MYCRYPTOPEDIA, "Full node and lightweight node." https://www.mycryptopedia.com/full-node-lightweight-node/, 2018. Accessed 29.01.2021.

[36] Qinwen, "Polkadot introduction." https://medium.com/@qinwen228/polkadot-introduction-815abecebe8b, 2019. Accessed 05.02.2021.

[37] M. Herlihy, "Atomic cross-chain swaps," in *Proceedings of the 2018 ACM symposium on principles of distributed computing*, pp. 245–254, 2018.

[38] A. Deshpande and M. Herlihy, "Privacy-preserving cross-chain atomic swaps," in *International Conference on Financial Cryptography and Data Security*, pp. 540–549, Springer, 2020.

[39] G. B. Association, "Self-sovereign identity." https://www.bundesblock.de/wp-content/uploads/2019/01/ssi-paper.pdf, 2018. Accessed 23.01.2021.

[40] Y. Liu, D. He, M. S. Obaidat, N. Kumar, M. K. Khan, and K.-K. R. Choo, "Blockchain-based identity management systems: A review," *Journal of network and computer applications*, vol. 166, p. 102731, 2020.

[41] L. Stockburger, G. Kokosioulis, A. Mukkamala, R. R. Mukkamala, and M. Avital, "Blockchain-enabled decentralized identity management: The case of self-sovereign identity in public transportation," *Blockchain: Research and Applications*, vol. 2, no. 2, p. 100014, 2021.

[42] Hyperledger, "What is ssi did?." https://hydraledger.io/what-is-ssi-did/, 2019. Accessed 29.09.2021.

[43] N. Naik and P. Jenkins, "Does sovrin network offer sovereign identity?," in *2021 IEEE International Symposium on Systems Engineering (ISSE)*, pp. 1–6, IEEE, 2021.

[44] HyperledgerIndy, "Indy walkthrough." https://hyperledger-indy.readthedocs.io/projects/sdk/en/latest/docs/getting-started/indy-walkthrough.html, 2021. Accessed 23.01.2021.

[45] M. Oza, "Decentralized identity — owning it!." https://medium.com/coinmonks/decentralized-identity-owning-it-94987f97649f, 2022. Accessed 05.04.2022.

[46] A. Mühle, A. Grüner, T. Gayvoronskaya, and C. Meinel, "A survey on essential components of a self-sovereign identity," *Computer Science Review*, vol. 30, pp. 80–86, 2018.

[47] A. Tobin and D. Reed, "The inevitable rise of self-sovereign identity," *The Sovrin Foundation*, vol. 29, no. 2016, 2016.

[48] C. Allen, "The path to self-sovereign identity." http://www.lifewithalacrity.com/2016/04/the-path-to-self-soverereign-identity.html, 2016. Accessed 29.09.2021.

[49] R. Belchior, A. Vasconcelos, S. Guerreiro, and M. Correia, "A survey on blockchain interoperability: Past, present, and future trends," *arXiv preprint arXiv:2005.14282*, 2020.

[50] S. D. Hub, "Front page." https://substrate.dev/, 2020. Accessed 11.02.2021.

[51] Hashport, "How hashport works." https://www.hashport.network/how-it-works/, 2022. Accessed 29.04.2020.

[52] L. Baird, M. Harmon, and P. Madsen, "Hedera: A public hashgraph network & governing council," *White Paper*, vol. 1, 2019.

[53] A. Mühle, A. Grüner, T. Gayvoronskaya, and C. Meinel, "A survey on essential components of a self-sovereign identity," *Computer Science Review*, vol. 30, pp. 80–86, 2018.

[54] M. S. Ferdous, F. Chowdhury, and M. O. Alassafi, "In search of self-sovereign identity leveraging blockchain technology," *IEEE Access*, vol. 7, pp. 103059–103079, 2019.

[55] U. Der, S. Jähnichen, and J. Sürmeli, "Self-sovereign identity – opportunities and challenges for the digital revolution," *arXiv preprint arXiv:1712.01767*, 2017.

[56] M. Shuaib, S. Alam, M. S. Nasir, and M. S. Alam, "Immunity credentials using self-sovereign identity for combating covid-19 pandemic," *Materials Today: Proceedings*, 2021.

[57] cheqd, "Self-sovereign identity use cases." https://www.cheqd.io/blog/self-sovereign-identity-use-cases.

[58] adnovum, "Exploring the potential of self-sovereign identity with representative use cases." https://www.adnovum.com/blog/exploring-the-potential-of-self-sovereign-identity-with-representative-use-cases, 2022. Accessed 29.09.2021.

[59] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design science in information systems research," *MIS quarterly*, pp. 75–105, 2004.

[60] I. Bider, P. Johannesson, E. Perjons, and L. Johansson, "Design science in action: developing a framework for introducing it systems into operational practice," 2012.

[61] BitXHub, "Bitxhub documentation." https://meshplus.github.io/bitxhub/bitxhub/introduction/summary/, 2020. Accessed 29.09.2021.

[62] BitxHub, "Cross-chain gateway deployment." https://meshplus.github.io/bitxhub/bitxhub/usage/single_bitxhub/deploy_pier/, 2022. Accessed 12.01.2021.

[63] BitXHub, "Bitxhub validation." https://meshplus.github.io/bitxhub/bitxhub/dev/rule/, 2020. Accessed 29.09.2021.

[64] G. Scolar, "Ghareeb falazi." https://scholar.google.com/citations?hl=en&user=8f1KQJsAAAAJ&view_op=list_works&sortby=pubdate, 2022. Accessed 29.04.2022.

[65] BitxHub, "Relay chain deployment." https://meshplus.github.io/bitxhub/bitxhub/usage/single_bitxhub/deploy_bitxhub/, 2022. Accessed 29.04.2022.

[66] BitxHub, "Pier repo." https://github.com/meshplus/pier.git.

[67] H. Fabric, "Using the fabric test network." https://hyperledger-fabric.readthedocs.io/en/release-2.2/test$_n$*etwork.htmlbring* – *up* – *the* – *test* – *network*.

[68] truffle, "Homepage." https://trufflesuite.com/, 2022. Accessed 29.04.2020.

[69] infura, "Homepage." https://infura.io/, 2022. Accessed 29.04.2020.

[70] metamask, "Homepage." https://metamask.io/, 2022. Accessed 29.04.2020.

[71] B. Insider, "Not ethereum, polkastarter, the dex protocol will launch on polkadot." https://www.bitcoininsider.org/article/94688/not-ethereum-polkastarter-dex-protocol-will-launch-polkadot, 2022. Accessed 29.04.2020.

[72] Sovrin, "Sovrin: A protocol and token for self-sovereign identity and decentralized trust." https://sovrin.org/library/sovrin-protocol-and-token-white-paper/, 2018. Accessed 29.09.2021.

[73] D. Reed, J. Law, and D. Hardman, "The technical foundations of sovrin," *The Technical Foundations of Sovrin*, 2016.

[74] Sovrin, "Stewards." https://sovrin.org/stewards/, 2022. Accessed 29.09.2021.

[75] J. C. Nauta and R. Joosten, "Self-sovereign identity: A comparison of irma and sovrin," *Technical Report TNO2019R11011, Tech. Rep*, 2019.

[76] A. Satybaldy, M. Nowostawski, and J. Ellingsen, "Self-sovereign identity systems," in *IFIP International Summer School on Privacy and Identity Management*, pp. 447–461, Springer, 2019.

[77] Sovrin, "Sovrin provisional trust framework." https://www.evernym.com/wp-content/uploads/2017/07/SovrinProvisionalTrustFramework2017-03-22.pdf, 2017. Accessed 29.09.2021.

[78] P. Windley, "How sovrin works," *Sovrin Foundation*, pp. 1–10, 2016.

[79] H. Indy, "Negotiate proof." https://hyperledger-indy.readthedocs.io/projects/sdk/en/latest/docs/how-tos/negotiate-proof/README.htmlf, 2018. Accessed 29.09.2021.

[80] W3C, "Verifiable credentials data model v1.1." https://www.w3.org/TR/vc-data-model/, 2022. Accessed 05.04.2022.

[81] H. Indy, "Indy." https://hyperledger-indy.readthedocs.io/projects/sdk/en/latest/toc.html, 2022. Accessed 05.04.2022.

[82] Sovrin, "Interoperability series: Sovrin stewards achieve breakthrough in wallet portability." https://sovrin.org/sovrin-stewards-wallet-portability/, 2020. Accessed 29.09.2021.

[83] H. I. SDK, "Setup indy sdk build environment for ubuntu based distro (ubuntu 16.04)." https://hyperledger-indy.readthedocs.io/projects/sdk/en/latest/docs/build-guides/ubuntu-build.html, 2018. Accessed 29.04.2022.

[84] H. I. SDK, "Indy sdk github." https://github.com/hyperledger/indy-sdk, 2018. Accessed 29.04.2022.

[85] S. Maldeniya, “Setup hyperledger indy pool in local linux environment using docker.” https://medium.com/@smaldeniya/setup-hyperledger-indy-pool-in-local-linux-environment-using-docker-304d13eb86dc, 2018. Accessed 29.04.2022.

[86] H. Indy, “Indy sdk for node.js.” https://www.npmjs.com/package/indy-sdk, 2022. Accessed 29.04.2022.

[87] Solidity, “Solidity.” https://docs.soliditylang.org/en/v0.8.13/, 2022. Accessed 29.04.2022.

[88] React, “React.” https://reactjs.org/, 2022. Accessed 29.04.2022.

[89] IPFS, “Ipfs.” https://ipfs.io/, 2022. Accessed 29.04.2022.

[90] node fetch, “Node fetch.” https://www.npmjs.com/package/node-fetch, 2022. Accessed 29.04.2022.

[91] P. Wiki, “Cross-chain message passing (xcmp).” https://wiki.polkadot.network/docs/en/learn-crosschain, 2020. Accessed 09.02.2021.

[92] P. Network, “A tale of two technologies presentation transcript.” https://medium.com/polkadot-network/a-tale-of-two-technologies-presentation-transcript-e7397c1c7a49, 2018. Accessed 12.01.2021.

[93] CryptoSeq, “Polkadot — an early in-depth analysis — part three— limitations and issues.” https://medium.com/@CryptoSeq/polkadot-an-early-in-depth-analysis-part-three-limitations-and-issues-d8b0a795a3e, 2020. Accessed 09.02.2021.

[94] M. Brill, R. Freeman, S. Janson, and M. Lackner, “Phragmén’s voting methods and justified representation,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, 2017.

[95] J. Petrowski, “Polkadot consensus part 2: Grandpa.” https://medium.com/polkadot-network/polkadot-consensus-part-2-grandpa-fb1963ef6c70, 2019. Accessed 07.02.2021.

[96] C. Gorenflo, S. Lee, L. Golab, and S. Keshav, “Fastfabric: Scaling hyperledger fabric to 20 000 transactions per second,” *International Journal of Network Management*, vol. 30, no. 5, p. e2099, 2020.

[97] PolkadotWiki, “Frequently asked questions (faqs).” https://wiki.polkadot.network/docs/en/faq, 2020. Accessed 09.02.2021.

[98] Polkadot, “Polkadot home.” https://polkadot.network/, 2020. Accessed 15.02.2021.

[99] Psrity, “A scalable, interoperable & secure network protocol for the next web.” https://www.parity.io/polkadot/, 2020. Accessed 15.02.2021.

[100] X. Luo, "Cross-chain messaging." https://w3f-research.readthedocs.io/en/latest/polkadot/networking/4-xcmp.html, 2020. Accessed 15.02.2021.

[101] J. Petrowski, "Polkadot consensus part 4: Security." https://medium.com/polkadot-network/polkadot-consensus-part-4-security-eb3180b6d7e4, 2019. Accessed 09.02.2021.

[102] PolkadotWiki, "Polkadot consensus." https://wiki.polkadot.network/docs/en/learn-consensus?fbclid=IwAR2hc46AGFBbnAJ0LFl1FaXA_8xVs_qu0cI0eiIm38u23rTOnZfQxBhqTAw, 2020. Accessed 09.02.2021.

[103] 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)*, pp. 193–210, IEEE, 2019.

[104] Hedera, "Mainnet nodes." https://docs.hedera.com/guides/mainnet/mainnet-nodes, 2020. Accessed 24.01.2021.

[105] L. Baird, "The swirlds hashgraph consensus algorithm: Fair, fast, byzantine fault tolerance," *Swirlds Tech Reports SWIRLDS-TR-2016-01, Tech. Rep*, 2016.

[106] H. Hashgraph, "Hedera governing counci." https://hedera.com/council, 2020. Accessed 11.02.2021.

[107] H. Hashgraph, "Tokenization on hedera whitepaiper." https://hedera.com/hh_tokenization-whitepaper_v1_20201207.pdf, 2020. Accessed 11.02.2021.

[108] Hyperledger, "Hyperledger quilt." https://www.hyperledger.org/use/quilt, 2020. Accessed 22.01.2021.

[109] Interledger, "Bilateral transfer protocol 2.0 (btp/2.0)." https://interledger.org/rfcs/0023-bilateral-transfer-protocol/, 2020. Accessed 11.02.2021.

[110] Interledger, "Stream: A multiplexed money and data transport for ilp." https://interledger.org/rfcs/0029-stream/, 2020. Accessed 11.02.2021.

[111] Interledger, "Simple payment setup protocol (spsp)." https://interledger.org/rfcs/0009-simple-payment-setup-protocol/, 2020. Accessed 12.02.2021.

[112] Interledger, "Interledger protocol v4." https://interledger.org/rfcs/0027-interledger-protocol-4/, 2020. Accessed 17.01.2021.

[113] J. Gray and A. Reuter, *Transaction processing: concepts and techniques*. Elsevier, 1992.

[114] Interledger, "Connector risk mitigations." https://interledger.org/rfcs/0018-connector-risk-mitigations/, 2020. Accessed 12.02.2021.

[115] G. Falazi, A. Lamparelli, U. Breitenbuecher, F. Daniel, and F. Leymann, "Unified integration of smart contracts through service orientation," *IEEE Software*, vol. 37, no. 5, pp. 60–66, 2020.

[116] A. Lamparelli, G. Falazi, U. Breitenbücher, F. Daniel, and F. Leymann, "Smart contract locator (scl) and smart contract description language (scdl)," in *International Conference on Service-Oriented Computing*, pp. 195–210, Springer, 2019.

[117] Google, "Protocol buffers." `https://developers.google.com/protocol-buffers/`, 2019. Accessed 17.01.2021.

8 Appendix A: Interoperability solutions found in the pre-study

8.1 Polkadot

Polkadot [20, 4] was created by a co-founder of Etherium, Gavin Wood. Polkadot can be seen as two seperate parts, which interact with each other. The first part are the Parachains. A Parachain is an independant blockchain which then connects to the second part of Polkadot, the Relaychain. This is the hub, which enables all chains inside this hub to share information with each other. Figure 46 shows the system Gavin Wood envisioned.

Parachains are primarily created in Substrate, developed by Parity Technoligies, as a framework for developing blockchains. Substrate and Polkadot were designed to work together. This doesn't mean that there won't be more languages compatible with Polkadot in the future, but currently, this is the primary one [50, 92]. Substrate was developed to accommodate the network level functionality required by Polkadot in order to communicate with other Blockchains in the relaychain [20] shown in Figure[46]. Polkadot also wants to connect with existing successfull blockchains, illustrating Etherium as an example [4]. They hope to achieve this with Parachain bridges, which should intermediate between the two forms of blockchains, providing provable finality to other outside blockcahins about the state inside the Relaychain [20]. The bridges unfortunalty remain a challange for the development team, four years later still suggesting solutions on how to solve the problem.

The Relaychain is where the Parachains can interact with eachother. The Relaychain is also itself a blockchain, storing all the information about the communication between Parachains on its own ledger, but does not store the message itself [91]. Polkadot envisions a capacity of 100 Parachains, but this could change if Polkadot manages to create 2nd order Relaychains and nth order Relaychains. The nested Relaychains as of 2020 seem to still be under development, setting the limit to 100 Parachains at the moment [20]. There are four critical roles nodes can take for the Relaychain to work, these are: Validators, Firshermen, Nominators and Collators.

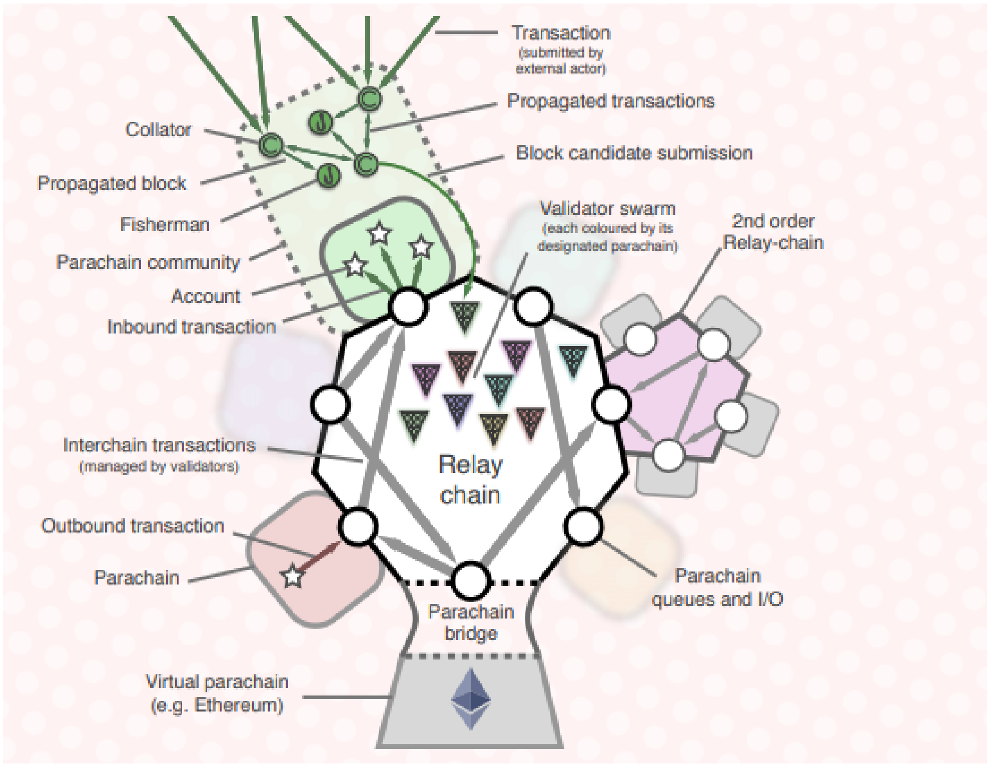


Figure 46: Shows the envisioned strucuture of Polkadot from 2016. Including the Fishermen, Validators, Collators. Image taken from [4]

- Validators: are the heavy lifters in the Relaychain. They are full nodes in the relaychain and interact with Parachain Collators which they are randomly assigned in regular intervals. This role is given to nodes being nominated with sufficiently high trust by the nominators, who have invested a sufficient amount of Polkadots token DOT. This creates a NPoS (Nominated Proof of Stake), if the validator is found to be malicious, a percentage of the invested DOTs are slashed from all nominators who nominated it. Validators will work together with the Collators on the assigned blockchain to validate the blocks provided by the Parachain, will make sure all messages have been recieved and answered, as well as create blocks on the Relaychain, as part of the BABE and GRANDPA hybrid consensus model.

- Collators: These are full nodes on their respective Parachains, having all the necessary information to author new blocks and create transactions. They watch the progress of the block production and consensus protocols in their Parachain. Under normal circumstances, they will collect and combine transactions, execute transactions to create an unsealed block, and provide it together with proof of validity block (B_{pov}) to one or more validators working for that Parachain.

- Fishermen: Fishermen are not related to the creation of the blockchains, but are rather observers, looking for malicious behaviour, motivated by a huge payoff should they find this maliciousness. The resources needed for a fisherman unlike the collator and validator is very small, there is no requirement for them being a full node. Fisherman themselves need a small bond in order to exist, so they don't spam the validators with false maliciousnes claims.

- Nominators: Nominators are any node who owns DOTS (Polkadots currencie) and wants to invest into a validators, they believe act honerably, motivated by a payoff, for successfull transactions.

It is assumed that $\frac{2}{3}$ of all nominators who place a stake in the validators are honest, giving a total of $\frac{2}{3}$ honest validators. Fishermen can be as dishonest as they want they will be found and punished by removing their small stake. Collators do not have a specific amount of honest participants for their parachin, but it's assumed that at least one collator is honest. Polkadot does however claim they have checks for a totally malicious member [20].

The NPoS is used to nominate the validators who are supposed to act on their behalf and are responsible for the security of the relaychain. The Validators are assumed to be bounded but grow linearly with the number of Parachains connected to the system, prefering 10 validators for each Parachain [5, 93]. This is presumably to avoid a problem where a too few validator are responsible for a single Parachain opening the possibility for validators colluding maliciousley. These Validators will be reelected every "era" of about roughly one day. Rewards or Slashings are given according to the elected Validators performance. The Validators fee should be higher then any DOT holder can afford, making it impossible for a single entity to support a single Validator. It is also expected that a significant amount of all DOTs will be used each era to promote the validators.

To ensure that validators are chosen as fairly as possible, so most possible nodes placing a stake get included, it has been set a great focus on gaining proportional representation, based on the Mathematician Edvard Phragmén's method to gain Proportional justified representation (PJR) [94]. Polkadot has made an adaption of PJR in order to secure that nominators, to a greater, extent, get represented by their chosen set of nominators they want to nominate. This ensures a wider decentralisation of nominators having a stake in the blockchain, ensuring that not only the biggest DOT holders control the system.

When blocks are created in the Parachain, and sent out to the Relaychain with some desired wish to exchange value, it is first sent to the collator, and needs to be validated by the validators currently validating the Parachain. Because the validators need to validate the Parachain block, it is not enough that the collator just presents the block to the validator. Validators are not full nodes of the Parachain, like the collators are, and therefore need more proof to accept a block. This is done by creating the proof of validity block B_{pov}, which is possible to verify even for validators. This is done using the parachins state transition validation function (STVF), which will be stored on the relaychain. The STVF will output the validity as a Merkel tree, header, and outgoing messages. There will ideally be ten Validators for each Parachain, going as low as 5, with an increased risk to security. Once enough validators approve the block, it is eligible to go on the Relaychain. Fishermen now have the oppurtunity to try to find malicious behaviour on the approved block as a second line of defence. As a third line of defence is a few randomly assigned validators which should proof the work done by the validators in the first iteration. The number of validators assigned is based invalidity reports made by fisherman, but it seams an extrea four validators for a total of 14 randonly assigned validators is ideal [93]. This third phase is done before GRANDPA gets a chance to vote, and finalize the block. With 14 validators looking at the proof, it is calcualted that it would take a malicious validator 50 years (Figure 47), assuming reorganisation of validators every five minutes, before a succesfull attack [5].

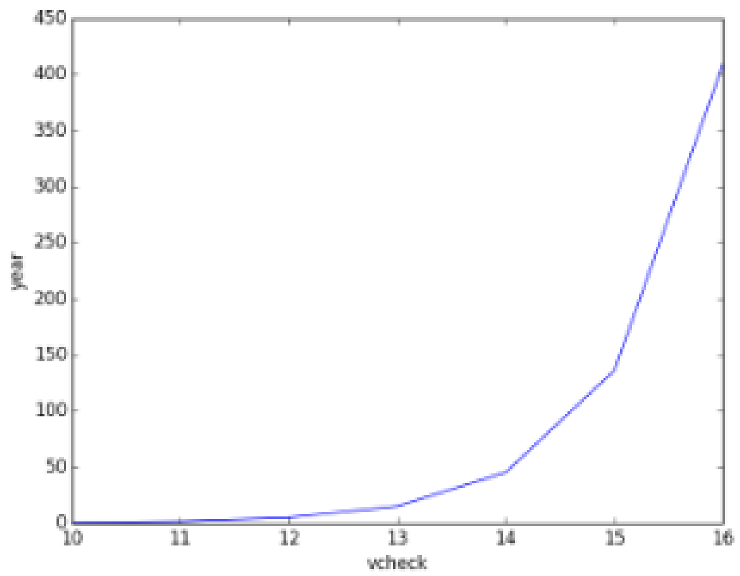


Figure 47: Shows the estimated time in years, it would take for a node to succeed in a malicious attack depending on how many other nodes are also checking. Image taken from [5]

When Polkadots Relaychain produces blocks, it uses both a fast block production engine called Blind Assignment for Blockchain Extransion (BABE) and a slower consensus algorithm to finalize the proposed blocks in BABE and removing all Forks called GRANDPA [20].

BABE consists of time divisions called epochs, where validators are allowed to suggest blocks for the blockchain. These epochs consist of multiple slots up to a given known bound R. There are two ways a Validator can become eligible to write a block in a slot. The first and preferred way is primary leadership, which is granded based on the evaluation of a verifiable random function (VRF). VRFs generate pseudo random numbers along with a proof that it was validly generated. The VRF takes an epoch random seed agreed by all nodes, a slot number and the nodes private key. Using these three together should give a random number, two factors, the epoche seed, and the slot are given, while the private key should in combination generate a truly random number. The validators will then create a number for each slot in the epoch, and if it comes below a threshold τ it is allowed to author a block in this slot. The second method comes form the fact that there might be cases where no validator comes blow the desired value τ . When this occurs, The Relaychain falls back to a round-robin fallback. Every Slot has a secondary leader agreed upon in the start of the epoch. This secures that every slot has a block author, and helps to guarantee consisten block time. The produced blocks will then be validated by GRANDPA. Grandpa diverges from a classical BFT algorithm with voting on chains, not blocks [95]. This allows for potentially more then one block beein accepted each iteration it finalizes blocks. GRANDPA works by having a primary validator. This validator was a part of the last block finalisation round, and holds an estimate from the previous round of the last block that could have been finalized last round, calculated based on the prevotes and precomits. The primary validator then invites all nodes which in the previous were part of the estimate, that is nodes which precommited. The primary will then broadcast its estimate for the last round, which the other validators will then prevote on. If it get more then $\frac{2}{3}$ of the prevotes, and it's a descendant of the finalized block of last round it passes. These two can then be combined to increase speed, by BABE following two rules for BABE to work with GRANDPA [6].

- It must always follow the fork, which has the latest finalized block.
- BABE must place the next block on the chain with the most primary blocks created by the previous BABE slot.

A practical example of how BABE would work tgether with GRANDPA is shown below in Figure 48.

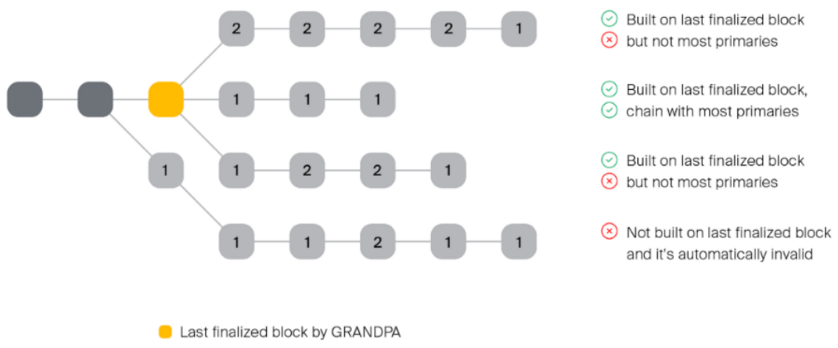

Built on last finalized block
but not most primaries
Built on last finalized block,
chain with most primaries
Built on last finalized block
but not most primaries
Not built on last finalized block
and it's automatically invalid
Last finalized block by GRANDPA

Figure 48: An example of how BABE needs to choose where to place the next block based on GRANDPA's need. Taken from [6]

Finally for actually communicating messages between two blockchains, The Relaychain uses Cross-Chain Message Passing (XCMP). XCMP consists of two parts

- Metadata about outgoing messages are included in the Relaychain, to later authenticate messages by the receiving Parachain.
- The massage bodies need to be distributed from sender to receiver.

Number two is actually essential, a message must be received, not receiving a message can potentially hinder a Parachain from building blocks. In order to ensure this every Validator which validated a B_{pov} and collators of the sending Parachain should keep all outgoing messages for a long period of time until they know it has been acted on.

In order to send messages from Parachain S to Parachain D, the message first needs to be communicated to the relaychain. This is done when a validator validates the B_{pov}. When it is in the Relaychain the Relaychain will update the Parachain header PH of Blockchain S, based on what messages were sent in Block B. The Parachain header contains a message root M of outgoing messages, as well as a bitfield indicating which Parachains were sent messages. The Root M is a Merkel tree of a hash chain. This hash chain contains all messages sent to Prachain S to Parachain D, allowing the receiving Parachain D to authenticated the message using the Merkel tree M. On the receiving side in Parachain D's header PH' contains a watermark. This watermark shows how far into the Relaychain blocks it has read, up to a certain block R, and also what particular message inside block R it acted on. When things have been acted on the watermark will also be updated.

All outbound messages will be placed by the collator into the outbound message queue, along with a destination and timestamp [91]. Collators from other Blockchains will routinely ping all blockchains to see if a message is there for them, based on what their current watermark. If they find a massage, this will update their watermark, which will be read in their next B_{pov}, notifying the whole system that this message has been received.

8.1.1 Consensus model

Polkadots consensus model is based on a hybrid consensus model, using BABE and GRANDPA [20]. The nodes involved in the consensus are the Validators chosen by the nominators in the system, giving the validators a NPoS to act on the behalf of the nominators who placed a stake in them. A hybrid consensus model means the finality gadget (what finalises the blocks) and the block production mechanism has been split. This speeds up the block production process, as blocks can be produced quick with BABE having probabilistic finality, and later in it's own time be validated by GRANDPA having provable deterministic finality.

8.1.2 Constraints

Polkadots currently biggest constraint is that it almost exclusively needs to be a Substrate built blockchain for it to be able to follow predescribed protocol specifications, making it hard for already depolyed permissioned Blockchains to migrate to Polkadot. It is further unlikely that Blockchains based on Hyperledger would be willing to integrate with Polkadot. Each Parachain is capable of around 1000 tps [93], while hyperledger is capable of at least 3000 tps going up to 20 000 tps [96], making it unlikely they would be willing to cooperate with a chain much slower than themselves.

Parachains must also have Collators, which will act as the communicator between the Parachain and the relaychain. These need to be full nodes of their respective Parachain. The blockchain the needs to accept messages from the collator, and subsequently answer them, as well as distribute Parachain blocks, and hold them for a certain period of time, satisfying polkadots need for all messages needing to be answered [20]. Polkadot also requires Parachains to have some form of PoV, so the Validators, which are not full nodes in the Parachain can validate the proposed blocks [20], this is to avoid potantial Censorship Attacks and Front Running attacks, where a malicious collator might either decide to withold information or prioretise certain messages [93]. Polkadot Also is constrained by the amount of validators it currently has. There should ideally be 10 validators for each Parachain, going down to 5 as a minimum [20]. Currently Polkadot manages to have 297 validators [97] in the system meaning that while it in theory can hold 100 Parachains, the real number is currently 29-59 Parachains, with the upper end significantly lowering the security it can provide [5].

8.1.3 Interoperability

Polkadot throught the use of XCMP provides a possability to create a one-way channel between two communicating Parachains. A pair of Parachains can have a total of two of these channels open one for sending and one for recieving [91]. Polkadot claim they can interoperate any type of data or asset [98, 99]. It is however not mentioned anywhere how polkadot should achieve anything except the transfer of Data, which can happen through XCMP [91], as it proves the authenticity of the data by storing a Merkel tree root in the Relaychain for message authentication [20]. Polkadot does however not assum anything about the Parachains [20], and once a message is sent over XCMP, it is not responsible for the distribution inside the Parachain [100]. It could be possible for a Parachain to use the XCMP, to send provable messages agreeing on asset transfers, and then themselves sorting out the burning or locking of assets as mentioned in section[2.4]. This would allow Polkadot to send all assets. This however is an assumption, and is not explained in any litterature.

8.1.4 Pros and Cons

Table 5: Pros and Cons of Polkadot.

PROS	**CONS**
Security	

• Uses NPoS to elect validators. The Stake is DOT tokens which will be slashed based on percentage of validators caught, raising to 100% at the $\frac{2}{3}$ mark, where the Byzantine Fault Tolerance failes for Polkadot [20]. • Polkadot achieves provable deterministic finality in their consensus model, meaning when GRANDPA has validated a block, it will never be changed, and this can be proven to eventual Bridge Blockchains [20]. • Polkadot uses an punishment system, if a Validator is found to act malicious or is not behaving. A certain percentage of DOTs will be removed from everyone who placed a stake in the validator if found. This can be very minor amounts, for not being offline, to a super linear slashing, going up to 100% if a high percentage of Validators are found to Equivocation. Placing a risk on trying to act maliciously [101, 20]. • For there to be a successfull attack, at least $\frac{1}{3}$ of Validators need to be malicious [20]. • All security for exahnging assets is done by the Relaychain, whihc every substrate blockchain inherits, when they become a parachin in Polkdaot [4, 20] • Polkadot checks the information placed on the Relaychain a total of three times before GRANDPA gets to do the final validation. The first check is done by the 10 Validators observing the blockchain, the second check is done by the firshermen, and the final check is done by a number of external to the Parachain validators for a final security check. [20] • Polkadot doesn't use a centralized time service, but instead uses it's own local clock and syncronizes with the rest [6, 20]	• Having few validators validate, means that it is open to potential attacks, where an outside party within the timeframe tries to discover and bribe the Validators in a certain Parachain. • If a faulty block is detected, after it has been finalized, Polkadot will have to roll back the state of all Parachains and all transactions inside the system to that point [93]. • Polkadot provides good security for the Relaychain, but the collators do not receive any incentives to do a good job from Polkadot, and there are limited ways to prove they are acting maliciousley [100]. rather they get incentives from their own Parachain, this can create censoring of transactions, both inbound and outbound, if the collators feel they are not incentivised enough by certain nodes [93].

Performance	
• Polkadot claims BABE will produce a block every 6 seconds [102], possibly going as low as 2-3 seconds if optimization works as intended, or increase if there come more parachins in the network [97]. • Every Parachain should be able to do about 1000 transactions per second [93].	• Bridges [4, 20] are a separate blockchain which can't be rolled back, therefore it has to be absolutely certain that there will not come any fisherman challenges. It is suggested a waiting period of 60 minutes, which is huge latency [93]. This however assumes they get bridges.
Scalability	

• Polkadot can handle a total of 100 Parachains in practice and is looking into expanding this with a 2nd order Relaychain, enabling all these chains to interoperate if wanted [20, 4].	• Polkadot in reality has 297 Validators [97], where a recomended amount for every Parachain is 10 and a lower bound is 5 [5], allowing them to currently have anywhere from 29-59 Parachains. • Polkadot has also looked into Bridges, in order to make it possible to include other Blockchains, currently focusing on BTC and ETH. They envision a bridge relay, understanding as much as possible of the bridged chain, and a bank for locking and releasing DOTs for ETH or BTC. The idea was first mentioned in the 2016 whitepaper [4] and still under development in the 2020 whitepaper [20], now with a design inspiration based on XClaim [103]. • Polkadot scaling past the first 100 Parachains seems challenging. Polkadot suggested looking into expanding this with a 2nd order Relaychain. This was a topic presented by Gavin Wood in his 2016 whitepaper [4], and is still a topic of interest in the 2020 whitepaper [20], where they say “We are also interested to increase scalability of Polkadot further, for example by investigating the idea of having nested Relaychains” • Polkadots currently exclusively needs to be a Substrate [50] built blockchain for it to be able to follow predescribed protocol specifications [20, 4], making it hard for already depolyed permissioned Blockchains to migrate to Polkadot
Costs	

• Polkadot is does not require GAS to place transactions of blocks.	• Polkadot is relying on auctioning off slots for Parachains in the relaychain [20]. This means smaller parachins without too many DOTs might never get the opportunity to participate in the chain, especially if their scalability remains a problem.

8.2 Hedera

Hedera Consensus Service (HCS) [17, 52] is one of the interoperability options emerging for especially Hyperledger fabric. It is a collection of nodes called the Hedera Mainnet, used for the fast creation of time-stamped messages, combined with the Hedera Mirrornet, to store and communicate the events agreed upon by the Hedera Mainnet, to the relevant parties. For HCS to communicate with the fabric blockchain, it will have to implement several SDKs and the Hedera API (HAPI) using protobufs. The HCS plugin will then be responsible for fragmenting and merging the blockchain's 6kb messages.

The Hedera Consensus Service is public, meaning that anyone can join it and, to a degree, see what is going on, by creating a mirror node.

Before connecting to Hedera, a blockchain client would configure one or multiple mirror nodes. The group would also define one or more topics attached to messages, which the client's application would send to the Hedera public network. They would finally also configure keys allowing those who have the keys to read the information sent for a specific topicID.

Hedera works with Fabric in the following way as shown in Figure 49 and explained below.

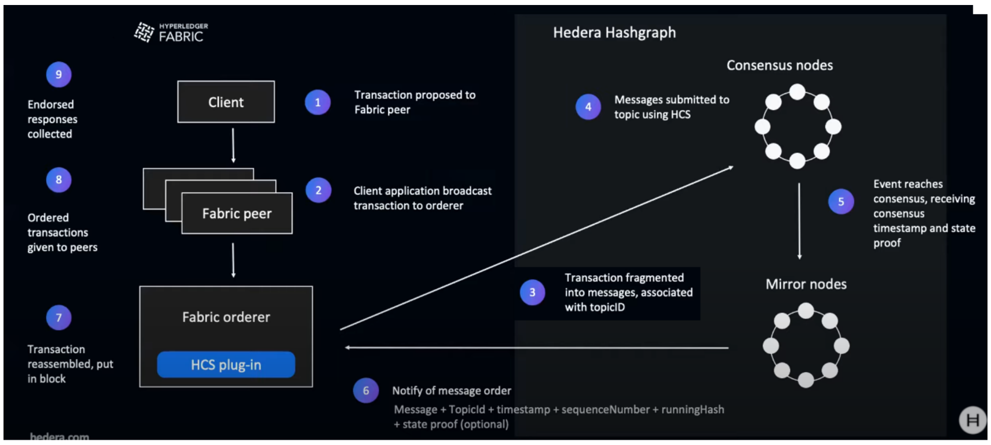

HYPERLEDGER FABRIC
Hedera Hashgraph
Consensus nodes
9 Endorsed responses collected
Client
1 Transaction proposed to Fabric peer
4 Messages submitted to topic using HCS
8 Ordered transactions given to peers
Fabric peer
2 Client application broadcast transaction to orderer
5 Event reaches consensus, receiving consensus timestamp and state proof
3 Transaction fragmented into messages, associated with topicID
Mirror nodes
7 Transaction reassembled, put in block
Fabric orderer
HCS plug-in
6 Notify of message order
Message + TopicId + timestamp + sequenceNumber + runningHash + state proof (optional)
hedera.com

Figure 49: Shows how the Hedera netowrk will comunicate a message from one blockchain to another. Images taken and edited together from a Hedera Webinar [7]

First, the Transaction is proposed in Hyperledger Fabric, and when a consensus is reached, this can be broadcasted to the HCS plugin for fragmentation. It is important to note that Hyperledger fabric's consensus relies on a deterministic consensus algorithm. This means that anything sent to Hedera will be guaranteed to be final and correct. This means all Hedera needs to do is prove to other users that nothing has been tempered with on the Hedera network itself. When sent to the orderer SDK, it is made into small packages called massages. Each message contains a TopicID, message and a fee, in Hederas own cryptocurrency Hbar.

The TopicID is the ID given to the topic the blockchain defined used to mark that this is information coming from them, enabling one or more mirror nodes to then publish the information to registered Hyperledger Fabric peers.

These messages are then sent to the Mainnet. The Hedera Mainnet is a currently permissioned network operated by the Hedera Governing Council [104]. Mainnet is a quick-acting information acceptance hub designed to set a stamp when all the nodes in the network came to a consensus of when an event was created for auditing purposes. This consensus is reached by gossiping to the other nodes, the information they received from a Blockchain client. The agreed-upon timestamp will be the medium time when every node who receives the message, allowing for a 100% final timestamp once all active nodes have received the information. This information will be sent to the Hedera Mirror nodes.

The information sent to the mirror net is the current package with a now added consensus timestamp, sequence number and a running hash. A running hash is used to inform how the fragments of a given topic's messages have come in relation to each other to reconstruct the whole message at the Blockchain client. The sequence number is used to tell how this message arrives relative to all the other messages sent to with this TopicID.

The massage is then sent to Blockchain clients with the Message, TopicID, timestamp, Sequence Number and running hash, where it is reconstructed by the blockchain's orderer using the keys necessary to reconstruct and read it. This is finally sent to the recieving blockchain.

8.2.1 Consensus model

The Hedera Consensus service [52] uses a Hashgraph protocol [105] to reach a consensus about when a message has been sent, using a gossip protocol to communicate with the Mainnet nodes and subsequently the Mirror nodes. The consensus is done in the Hedera Mainnet; currently, permissioned [104] and operated by the Hedera Governing Council [106] consisting of huge companies like IBM, LG, Google, ect. The message is thereafter sent to the Mirrornet, a public collection of read-only nodes used to store and process the information stamped by the Hedera Mainnet.

8.2.2 Constraints

Any blockchain wanting to participate in the Hedera Consensus service will have to integrate several SDKs and the Hedera API (HAPI). Before connecting to Hedera, a blockchain client would configure one or multiple mirror nodes. The group would also define one or more topics attached to messages, which the client's application would send to the Hedera public network. They would finally also configure keys allowing those who have the keys to read the information sent for a specific topicID. [52]. If the Blockchain is interested in using Tokens, it would have to comply with the Hedera Message Standard [107]. Data sent must be either provable deterministic finality or deterministic finality. Probabalistic finality, where information later is not added to the blockchain, could create false Data for other chains.

8.2.3 Interoperability methods

Hedera would interoperate, using tokens created in the different Blockchains. These tokens would be what Hedera defines as Security Tokens [107]. The tokens represent some form of value, either fungible or non-fungible. Hedera has two services for tokens, one on the Hedera Mirrornet, and the other going over the Hedera Consensus Service, which stores the tokens on blockchains. In order to get tokens over the Hedera Consensus Service, the tokens must follow the Token Message Standard [107], consisting of Application logic, used to define roles and behavior of a token, called the Token Contract. The nodes which contain the token contract logic are referred to as token nodes. The nodes will be responsible for using the application for validating the Hedera Consensus Service's toekn message. It would ensure that it complies with the roles and behaviours specefied for the specific token, for instance it could ensure that the appropriate keys were used to sign a required transaction, therefore validating the message as correct [107]. Token nodes will subscribe to the appropriate TopicID for the token, which it will validate or reject if a message containing the TopicID is made. Token contracts can be made more complex to include atomic swaps or automated event triggering. Atomic swaps would require all parties involved to be in both blockchains where the exchanges happen [52].

8.2.4 Pros and Cons

Table 6: Pros and Cons of Hedera.

PROS	CONS
Security	
• Hedera uses a Hashgraph Protocol [105], meaning giving it asynchronous byzantine fault tolerance (ABFT). This would stop any DDoS attack from working [52]. • A Hashgraph Protocol also ensures fair timestamps and transaction order and access, meaning that no censoring or prioritizing of messages can happen [52]. • Blockchains can send transactions to multiple nodes in the mainnet at once, ensuring that it is not sent to a malicious one withholding information [52]. • Messages can be encrypted, so only the appropriate parties with correct able to read the message [52].	• Hedera Governing Council governs the Hedera Mainnet, which consists of a relatively small amount of cooperations, could collude to skew consensus [52]. • Having too few Token nodes can cause maliciousness [107] and to what extent these Token nodes communicate to gome to a consensus of validity is unknown.
Performance	

• Hashgraphs are 100% efficient. Blocks are never suggested and later removed [52] • Hedera can achieve a very high throughput of up to 200 000 to 250 00 100 byte tps, with a latency of 6 seconds for consensus finalety, or 50 000 100 bye tps with a latency of 2 seconds consensus finality [52]. The throughput is achieved with 16 nodes run by the Governing Council [104].	• If many mirror nodes are created, this can have an impact on the performance, this would be small [52]
Scalability	
• Mirrornet nodes, which are read-only, don't participate in the Mainnet and therefore don't create latency problems for the Mainnet. • Hedera wishes to, in the future Shard their Mainnet, to increase performance, as not every node needs to process every transaction. Sharding could solve their Cons for scalability [52], unless too much overhead is added for shards communicating with each other.	• Latency in consensus finality might become a problem if too many nodes become active in the Mainnet. Many nodes reduce security concerns but increase latency. If Hedera would use 64 Nodes, they would not increase throughput but would advance latency to 20 seconds, assuming nodes spread across the globe [52] • With more mirror net nodes comes more demand for the throughput. Because Hedera doesn't scale well with more nodes observed from figure 1-3 in [52] problems with queues waiting to be processed by the Hedera Mainnet might arise. • The tokens created to communicate with different blockcahins in the Hedera Consensus Service, might be use case specific, meaning different tokens might need to be made.
Costs	

•	• Hedera does require a small amount of their token Hbar, for each message sent to the Hedera Mainnet, as a processing fee.

8.3 Hyperledger Quilt/Interledger

Interledger [18] and Hyperledger Quilt will be talked about in the same section. This is because Hyperledger Quilt claims to be a Java language implementation of the Interledger protocol [108].

Interledger [18] is a payment system, allowing two blockchains to trade currencies between the two blockchains, often using multiple other blockchains to achieve this. The critical assumption in Interledger is that some nodes are in both blockchains, enabling them to become connectors.

The unique idea with interledger is ledger provided escrow. This means instead of having a trusted third escrow, which might run away with your money, you only need to trust your own blockchains escrow to hold your funds, until it has been confirmed with proof that the other party has transferred the funds. Essentially meaning you need to be able to trust your own blockchain. This idea can be continued with very long chains of blockchains to finally trade the values which both parties wish for. To facilitate this, the intermediary connectors will set themselves available to trade currencies intermediary for a small fee. The participants in the trade have three roles:

- Sender: Is the party that organises the payment
- Reciever: The final recipient of the payment
- Connector: The intermediaries between a sender and a receiver, that forward the ILP packets. They do this service for a small fee provided to them. This is only needed if the sender and receiver party don't have a monetary system in common. In order to be a connector you need to have two accounts one on each chain which you connect, making it possible for it to intermediary between two currencies.

Interledger is inspired by the internet, and therefore takes similarities to it, dividing into layers of protocols [8], with different responsibilities. Lower level protocols provide basic functionality, while the higher level ones provide more advanced functionalities, while at the same time depending on the lower level functionalities.

- The lowest level protocol is the link protocol. This layer is often incorporated into a ledger plugin, because it needs to communicate settlements that occur int the underlying ledger. The link protocols are the ones providing secure two-way communication between two nodes in in the same ledger. communication is done over WebSocket, using the IL-RFC-23:Bilateral Transfer Protocol [109]

- The Interledger Protocol (currently ILPv4) these packets pass through all participants from sender, through possible connectors to reciever. This level handles the currency amounts, if currencies arrive or expire, finding the path between sender and reciever through possible connectors, as well as holding a cryptographic condition whose success condition is only known to the recipient.
- The Transport protocol is responsible for the end-to-end communication between sender and receiver. This layer is responsible for
  - Defining the condition for fulfillment that are used on the ILP layer (When will the currencies be released)
  - Deciding the speed at which packets can be sent
  - Determining the exchange rate of a payment
  - Encryption and Decryption

  Interledger recomends using STREAM [110] for the transport protocol, but does allow other transport protocols.
- the Application Protocol, this layer deals with destination discovery, where exactly should the money be sent, what transport layer protocol should be used, and any other information, which should be communicated in ILP packet data. Interledger suggests here using the Simple Payment Setup Protocol (SPSP) [111], which uses the recomended STREAM [110] transport layser protocol.

Currently Interledger is on it's 4th iteration called ILPv4 [112]. Different from Their first iteration introduced in their whitepaper [18], is the design of smaller, more homogenous packet amounts. This still means that you can send large amounts through a high level protocol, but it is now optimised for sending large volumes of small packets.

The ILPv4 now also uses Payment channels [112], this means that the connectors now send the IPv4 packets through to the next connector, instead of going through the blockchain ledger itself. This allows for the timeouts to be much shorter, because they don't need to take into consideration the processing time of slow POW based ledgers like Bitcoin, which would increase the payment time substantially. Payment channels are a way to use signed claims against funds held on a ledger, they can exchange a signed claim after every ILP packet is fulfilled, to keep the trust between ledgers as low as a single packet value. When the two trading blockchains are done with their trade, the connectors can then use the signed claim in their own ledger to balance the funds in their own ledger, allowing for greater speed.

Higher level protocols are generally used for initiating the communication, while lower level protocols are used for the actual transfer, but the user can also use higher level protocols to send currencies, although ILPv4 is set up for using lower level protocols, which therefore is recomended.

Interledger moves money by relaying packets. This is first done by sending a "prepare" packet, this allows all blockchains in the possible movement to prepare currency, and with the condition for releasing it. Connectors will here prepare the balances between them, as well as adjust the currency conversion, and fees they take.

When the Prepare packet finally arrives the reciever, and the reciever accepts the proposed amount, a fullfill packet will move down the chain back to the start, confirming the planned balance changes on all chains. At any point in the chain, the connectors or recievers can reject the proposal. Figure 50 shows the Interledger consensus.

This can happen becouse a connector failes to prepare the funds, a reciever rejects to offer, or an expiration happens. In all cases, the reject message goes down the chain to the reciever, and no balance is changed.

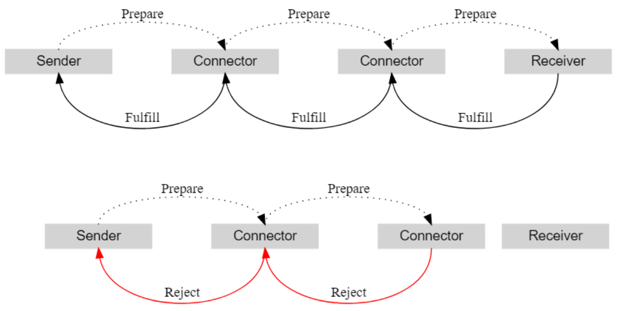


Figure 50: Shows the ILP lifecycle with the two possible outcomes of Fulfill or Reject. Taken from [8]

With smaller packets, it is now possible for the transport protocol to combine the packets into the desired amount, this allows small packets to be sent, at a much faster speed, finding the optimal route, and if rejected trying another route. Smaller packets also open for more routes, as connectors might not have the big chunk sum, but should have smaller sums prepared.

When the whitepaper [18] was propesed, they had two proposed methods, atomic mode and universal mode. Atomic mode has since been removed, as this required agreements in different blockchains which could not be generalised [8]. It is instead now fully commited to what is called universal mode in the whitepaper. Universal mode is similar to a two-phase commit, here each party is isolated from risk beyond their imidiate peers. However between the two pairs there is a chance for a non-blocking [113]. This should however be mitigated, as Connectors take a fee for their transactions, knowing about this risk, and with ILPv4 only very small amounts are being sent every iteration, meaning huge sums of money will not be lost for the node.

8.3.1 Consensus Model

Interledger works by the different protocol layers coming to an agreement on what is to be exchanged, the amount exchanged, and how to communicate. It works with generalized agreements, which should work with all blockchains, provided they have the Interledger Architecture implemented [18, 8].

8.3.2 Constraints

The Interledger is reliant on the fact that all participants using interledger must have nodes in multiple blockchains [18]. It is also important that the blockchains are able to communicate using WebSocket, as this is needed in the Link Protocol, to communicate between two nodes in the same blockchain. It is also beneficial to use STREAM [110] and SPSP [111], as it will presumably be more common on other blockchains, allowing for communication between more blockchains.

8.3.3 Interoperability

For interoperability it can exchange currencies. Any currency which there is a path for can be traded [18]. Interledger can't exchange data or non fungible assets. Data, connectors would have no incentive to convey the data. Non-fungible assets are unique assets, and therefore don't have any given exchange rate which interledger uses to exchange currencies over the connectors. There would also not be any incentives for the connectors to exchange the non-fungible asset, as they can't claim a fee from a non-fungible asset.

8.3.4 Pros and Cons

Table 7: Pros and Cons of Interledger/Hyperledger Quilt.

PROS	**CONS**
Security	

• Interledger provides ledger escrow, meaning that the locking and subsequent distribution of funds is done on the blockchain itself, avoiding potential 3rd party escrow acting maliciously. Now the risk is on the blockchain itself [18] • Connectors can choose to blacklist senders and receivers, if they choose, avoiding payment griefing [114]	• As of ILPv4 atomic mode is no longer in use, and are now using universal mode [8]. Universal mode uses a two phase commit, which is vulnerable to non-blocking [113], where money in a transaction can be lost. • Because single nodes are used as connectors, if an attacker can DoS the node after the outgoing payment transfer, but before they can fulfill their own transfer, making the node lose funds [114]. Mitigation methods can be taken, but this is for the individual node to take precautions.
Performance	
• ILPv4 has substantially reduced the time between proposal and transaction using Payment channels. The time it takes however is still reliant on the receiving blockchain's throughput. If the receiving blockchains throughput is fast, the proposal will be processed fast, if not the transaction can take time. • With the introduction of ILPv4 small packets open the possibility to interact with more connectors, as having a large transaction, which a connector could not balance would previously be a bottleneck [8]	•
Scalability	

• Interledger benefits from scalability, and can in theory scale indefinitely. Scalability opens more paths for transactions, allowing for bigger competition among connectors for better fees [18] and possible shorter paths between blockchains, as the right connectors exist .	• The opposite end of the spectrum is if there are too few blockchains using interledger, then fees can be high, paths can be large or it can be impossible to exchange the currencies.
Costs	
•	• Costs can become substantial. If a route between one currency to another becomes big. Because every connector requires a small fee, the fees can become big. [18]

8.4 Smart Contract invocation protocol

The Smart Contract Invocation protocol (SCIP) [9, 115] is an abstraction layer, that allows a blockchain to invoce smart contracts on another blockchain. It has developed an uniform message protocol sent over the SCIP to the other blockchain over an URL. Blockchain users need to operate a SCIP, exposing the smart contracts, they themselves wish to expose. The Consumers can then reach these SCIP Gateway, by using a Smart Contract Locator (SCL) [116]. SCL addresses smart contracts from outside their blockchains. An external consumer can invoke the contract and receive information from the blockchain that otherwise would be inaccessible. This is made possible over a gateway, a web-accessible agent that can mediate between the external consumer and the blockchain. SCL only allows invocation smart contract addressing only, assuming that the communication between external consumer and gateway over HTTP is adequately secured. The SCL uses and URL to address gateway, containing the following information in the HTTP POST message, shown in Figure 51

- Which type of blockchain is being addressed
- Which exact blockchain network, as gateways can have more networks accessible
- The blockchain-internal smart contract address or identifier.

`https://gateway.com?blockchain=ethereum&blockchain-id=eth-mainnet&address=0xa0b73...0b80914`.

Figure 51: An URL used in the SCL, in order to locate the correct smart contract. Here gateway.com is the domain of the gateway, blockchain=ethereum specifies that we want to reach an ethereum blockchain id=eth-mainnet specifies that it's the ethereum mainnet that should be contacted, and address=0xab... specifies the smart contract unique address which is wished contected. from [9]

The SCIP has parameters in the message, which are used to locate the appropriate smart contract using SCL. SCIP allows a blockchain to invoke 4 different methods, Invoke, Subscribe, Unsubscribe and Query.

The Invoke method allows a blockchain to invoke a specific smart contract on another blockchain, and receive the information desired, and works as follows and shown in Figure 52.

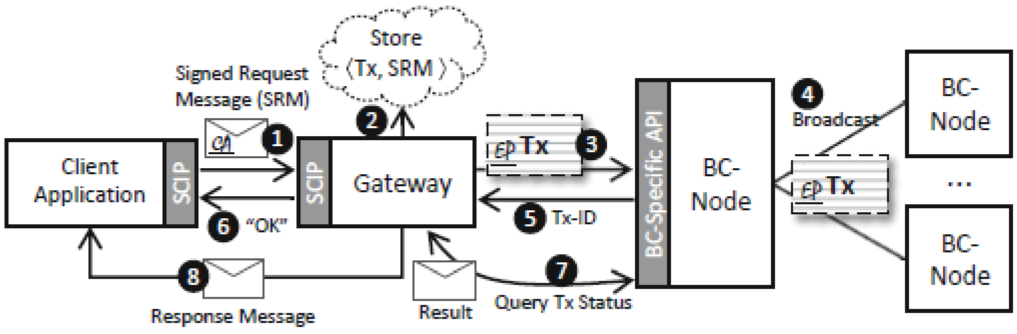


Figure 52: A visual explanation of what happens in step one too eight below, taken from [9]

- Step one: is communicating with the SCIP gateway to communicate what the receiving blockchain actually wants from the blockchain it will interoperate with. To attain the desired information, the receiving blockchain must send packets with information to the gateway. This packet contains:
  - Desired function identifier, which is the name of the smart contract
  - Parameters, which are inputs that should be put into the smart contract to be executed on the sending blockchain.
  - Callback URL: The URL which the gathered information of the sender must be sent to (the receiving blockchain)

  - Correlation identifier: Enables the receiving blockchain to know exactly what message it is receiving back, but assigning a unique identifier to every request and subscription.
  - Degree of confidence: a percentage going from 0 to 1 on how certain it should be that the information gathered actually ends up on the senders blockchain. Close to 1 is high and close to 0 is low.
  - How long in seconds the gateway should wait before deciding the degree of confidence that receiving blockchain is asking for is not obtainable.
  - Signature: the encoded signature of the receiver

  When all the desired information has been put into the message, the client application, signs it using the algorithm SHA256withECDSA amd the normative curve secp256k1, sending it to the gateway.

- Step two: This is where the information created gets sent over HTTP to the SCIP Gateway. It is signed by the gateway on the behalf of the receiving blockchain. The Signed transaction (Tx) and the Signed Request Message SRM are then permanently stored at the gateway. The reason it is signed by the gateway is that the receiving blockchain has no idea of the structure of the sending blockchain, preventing it from formulating a signature. Therefore it has to be signed by the gateway, which knows the technical details.
- Step three: The gateway sends the message to a node in the sender blockchain, using its API.
- Step Four: The node validates it and starts the consensus process by announcing it to the network of nodes.
- Step Five: When the consensus begins, it is assigned a unique ID and the Gateway is informed.
- Step Six: The gateway informs the receiver about the successful submission of the transaction.
- Step Seven: It will now query the sender blockchain node about the status of the transaction
- Step Eight: If the query yields sufficient confidence from the sender blockchain specified by the receiver's degree of confidence within the timefram of the timeout, the gateway sends the execution results back using the callback URL specified.

The SCIP also has a Subscribe method, allowing the blockchain to monitor desired occurrences on other blockchains, using the gateway as an information gatherer.

The Subscription method requires the receiving blockchain to create a subscription, using the Event and function identifier, which are simply names of the event and function, a callback URL, Correlation identifier, and degree of confidence. The last thing needed is a Filter, a boolean expression used to only notify the requester about events they actually are interested in. When later then an occurrence is detected by fitting the subscribed description, the gateway checks the degree of confidence, if it matches or is above the requested amount, a callback using the callback URL and the correlation identifier is issued with the information requested.

The Unsubscribe methode, would then unsubscribe the client from the desired smart contract function.

The Query Methode allows a client application to query the previous occurrences of an event. This will need the Function identifier and function, filter and a timeframe, which declares a timeframe to look for the desired occurrences. This will return a result of the occurrences with their timestamp, together with the Parameters of the searched for information.

8.4.1 Consensus model

The consensus is achieved on the invoked blockchain. SCIP is only an intermediary, allowing blockchains to expose smart-contracts to other blockchains, which they can invoke or subscribe to. The SCIP will however in the uniform messaging protocol return a degree of confidence that what has been palces on the other blockchain has actually been placed on the blockcahin. It seems that the SCIP will wait until the desired degree of confidence has been reached before it will return the message to the sender.

8.4.2 Constraints

The SCIP abstraction layer must be implemented [9] on both the blockchains that wish to interoperate and the appropriate smart contracts being exposed in a SCIP gateway. This gateway would also need to be made. Adequate security must be implemented [116], so a safe communication between the client application and SCIP gateway.

They mention having a distributed SCIP gateway to avoid single point of failure [9], however it seems that this solution must be self-implemented, and is not something they themselves offer.

8.4.3 Interoperability

SCIP is used to gather data from one blockchain and sending this information to another blockchain or database. Because it uses the smart contracts inside the blockchain it queries, it can receive proof adequate to the data consumer about how likely it is that this data will be on the blockchain [9]. The assets which can be exchanged are dependant on the smart contracts which the blockchain exposes.

The SCIP provides an uniform way to reach smart contracts on another blockcahin using the SCL [9], it also has a uniform protocol sent over the SCIP which is used to invoke the different smart contracts. As well as a JSON schema used to handle different encoding types which might occur in different smart contracts existing on blockchains, providing a 1-to-1 mapping to generate. corresponding native data types.

8.4.4 PROS and CONS

Table 8: Pros and Cons of SCIP.

PROS	**CONS**
Security	
• Using native smart contracts of the blockchain, which gives out-degree of confidence, the receiving party can know the probability that the invocation or query will have deterministic finality [9]. • The gateway needs a signature; if the gateway only accepts authenticated signatures, even if a malicious actor would try to invoke a smart contract, he could not because he would not have the necessary signatures. [9].	• In the whitepaper [9], there is mention of a distributed SCIP gateway, however, it seems to suggest this as self implementation, meaning it must be assumed, it currently is a single gateway, validating these concerns listed below. • The Gateway seems to be only a single database [9]; this could result in DoS attacks • The Gateway is also vulnerable to downtime assuming it is a single database. Consumers would during the downtime not be able to listen to the blockchain or invoke contracts. • The Gateway uses SQL to invoke smart contracts [116], leaving it open to potential SQL attacks if not secured. • Providers of the SCIP are responsible for Gas fees of their respective blockahins, this could lead to high costs for a provider, if someone decides to abuse posting information on the blockchain.
Performance	

	• depending on the degree of confidence wished for by the sender, a confirmation might take a long time, especially on probabilistic blockchains
Scalability	
• There can be an unlimited amount of gateways created, as long as the SCL in the blockchain knows how to contact them, it can have an infinite amount of gateways connected to it [9]. • More methods other than Invoke, Subscribe, Unsubscribe and Query could be added [9].	• For each SCIP gateway to work ideal, the two or more blockchains using the gateway need to communicate the use cases and create appropriate smart contracts, so the other parties can get the desired benefits [9, 115]
Costs	
• There are no costs associated with the SCIP.	• Providers are responsible for the cost associated with using blockchains requiering Gas fees

8.5 Ermyas Abeb Relay

Abebe et al [10]. define a relay, which itself will be a separate component within the blockchain network. The relay service, which they call it, should authenticate data from other blockchains by fetching the data and verifiable proofs from the sender blockchain.

The team has decided this to be a separate component, so all nodes don't need to implement the changes, requiring considerable changes inside the blockchain architecture. By introducing a separate component as the relay, the communication protocol can evolve independently. Having the relay separate allows for further implementations on the relay without interfering with the blockchain itself.

However, the implementation does assume that the two Blockchains that wish to interoperate have prior knowledge of each other. The Blockchains must know the identity of the other, as well as the configurations. The knowledge is needed for the blockchains to communicate with each other, while the configurations are needed so the blockchains can agree on a verification policy used when communicating, which requires the blockchains to know the verification methods of the other and to know what smart contracts exist, to invoke their query.

The relay itself communicates with other Blockchains using the relay's shared network-neutral language. The shared network-neutral language contains Google's Protocol Buffer [117]. The protocol is structured to provide the necessary details for addressing a network, ledger, and smart contract, the function name and arguments for remote queries, as well as verification policy, satesfiable for both parties of the requesting blockchain. This should yeld a response that includes the data queried and the proof that satisfies the requesters verification policy. Finally, the relay includes pluggable network drivers that translate the network-neutral protocol messages into calls that the specific blockchain will understand. The system needs system contracts, which will need to be implemented on all blockchain nodes. These will be used to enforce network rules for data exposure, and acceptance of what has been sent is deemed valid and can be placed on the blockchain. It can also be used to encrypt sensitive data between relays. These system contracts are

- Configuration Managment: contains the identity and configuration information about blockchains, which it can interact with, and is used by other system contracts for every cross network interaction, as well as setting the verification policies and acceptance, which incoming data must pass, verified by the Data acceptance contract.
- Data acceptance: allows the receiving blockchain to determine if the data, and corresponding proof satisfied the verification policy, before it is written on the ledger.
- Exposure control: sets and enforces access control policy rules against incomming requests, deciding what in the ledger and smart contracts can be exposed to the specific interoperable blockchain.

The system contracts will be deployed in the same way as application smart contracts using a supported smart contract runtime environment and language.

When everything is in place, the Blockchains will communicate in the following way shown in Figure 53

- Step One: The destination network submits a request to its local relay service by specifying the source network's unique name, ledger, contract, and function to invoke, along with any arguments. It also establishes the verification policy determined during the initialization phase.
- Step Two: The Relay Service uses the Configuration Management to discover the desired blockchain based on the destination network's name.
- Step Three: The destination blockchain's relay serializes the request and forwards the message to the source relay.

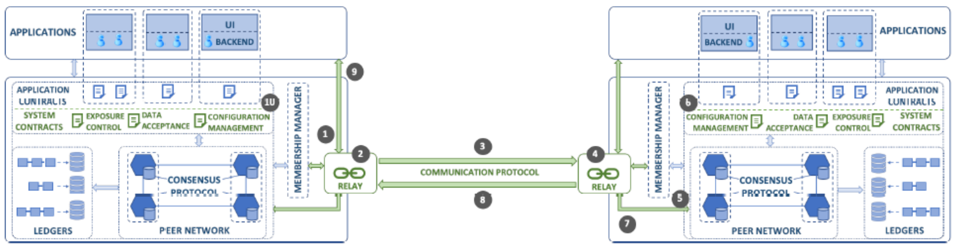


Figure 53: The figure shows how Abebe et al's Relay service communicate with different blockchains. Image taken from [10]

- Step Four: The source relay deserializes it, and depending on if there is a hierarchy in the nodes determines which nodes should get to see this message.
- Step Five: It then creates a query against the respective nodes in the network, based on the verification policy.
- Step Six: The nodes use the Exposure control to determine if the destination blockchain has the appropriate permissions to read the data.
- Step Seven: The result for each of the selected nodes create the proof satisfying the verification policy (assuming the destination blockchain has the appropriate permissions)
- Step Eight: The source relay serializes and sends it to the destination relay.
- Step Nine: The destination relay then sends it to the destination blockchain.
- Step Ten: The destination relay constructs a transaction, which includes the remote query and the proof. The Data acceptance contract then validates the result data and proof against the agreed-upon verification policy. If everything is in order, the ledger gets updated.

They managed to complete trade of a NFA (a Bill of Lading) and FA (money) in their use cases. This was done with the assumption that both parties were in both the two blockchains Tradelense, and We.Trade. However, they assumed that the trade itself was done between the two parties in person, and therefore there was no Atomic swap or similar methods. Making the trade in person means that both the NFA and FA can be seen as a trade of data and not an Atomic Swap. However, the team is aware of this and considers trying to expand the architecture with some form of atomic swap to widen what can be sent.

8.5.1 Consensus model

The agreement is met if the sending blockchain can provide an appropriate proof, which was previously agreed using the configuration management, and subsequently approved by the data acceptance when the blockchain received the data [10].

8.5.2 Constraints

The Relaychain must be implemented, with the Protocol Buffers, Network drives, and the system contracts, unlike the Protocol buffers and Network drives that need to be placed on all nodes in the two interoperating networks. It is expected the interoperating chains have prior knowledge of each other. Depending on the use cases for the interoperating blockchains, more smart contracts might need to be made [10].

8.5.3 Interoperability

It works only on receiving data. The team is looking into extending atomic and HTLC interoperability[10]. This would enable FA for NFA or FA for FA.

8.5.4 Pros and Cons

Table 9: Pros and Cons Abeb et al. Relay.

PROS	**CONS**
Security	

• The Data acceptance system control allows the receiving blockchain to verify the information received, ensuring the blockchain's data is valid [10]. • The Configuration management communicates with the sending blockchain on what proof the receiving blockchain requires for acceptable data [10]. • The Exposure control decides who can see what in the blockchain, meaning different blockchains can access various information depending on the trust between them [10] • System contracts can encrypt the message before being sent to the relay, ensuring that shared data is unreadable for the relay [10].	• The network relay is a single node and is therefore vulnerable to DoS attacks [10]. • The network relay is s single node and is therefore vulnerable to downtime if the server shuts down [10].
Performance	
• Using smart contracts on another ledger means the speed relies on the Blockchain, which has to process the query.	• Beein a single point of failure, if it failes and has downtime, nothing can be done during this period of time.
Scalability	

• The relay service is a separate component inside the network. For this reason, changes can be made on the relay without affecting the network itself [10]. • Plans are being made to extend the solution to incorporate Asset transfers, either using Atomic swaps or Hash Time locked constraints (HTLC), opening for broader use cases [10].	• The networks communicating with each other are reliant on some form of proof, which both networks can agree is valid [10]. • It is assumed that interoperating networks have prior knowledge of each others' identities and configurations recorded on the ledger [10]. • For each chain being interoperated, the two parties need to agree on all system contract requirements [10]. • More smart contracts might be necessary to implement, depending on the use cases of the interoperating blockchains.
Costs	
• Once running, there are no costs to use the relay.	•

8.6 Hyperledger Cactus

Hyperledger cactus [11] is a very new project in the Hyperledger family, aiming to create interoperability options between ALL Blockchains. This would enable Hyperledger to communicate with any other blockchain, be it Ethereum based, Bitcoin-based, or any other blockchain.

They hope to achieve this by extending a group of validator nodes, which provide the proof of state of the ledger they are connected to (shown in Figure 54). The validator nodes are ledger-specific plugins, implying that there must be created a smart contract for the validator nodes to observe the ledger state to finalize a proof. Giving the Validators the proof of the underlying blockchain means that the validators can communicate with other desired blockchains using their validators, bypassing the need for ledger-specific signatures. Verification would be done with the validator nodes of one blockchain providing their public key to another blockchain to accept the authenticity of the state the validators claim.

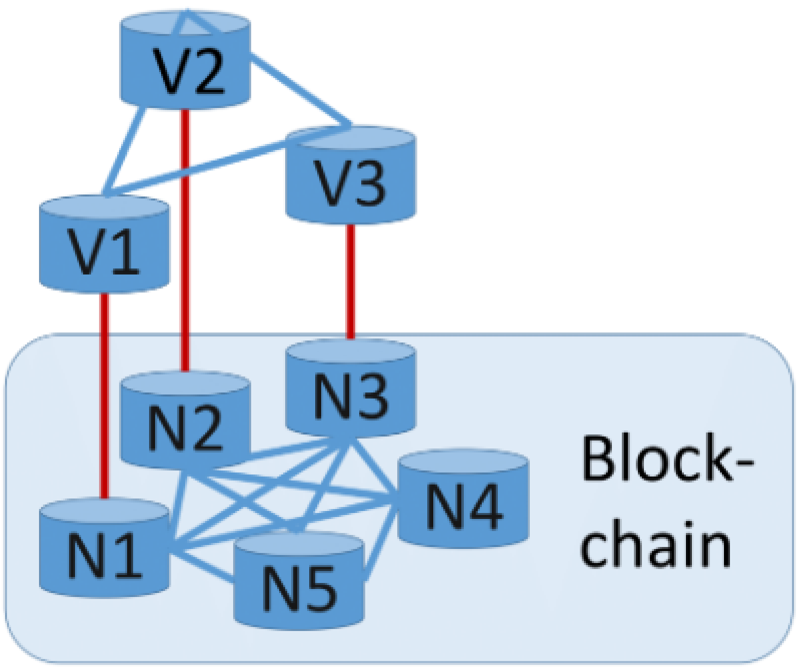


Figure 54: A proposed structure of how Hyperledger Cactuses Validator nodes would work with the Blockchain nodes. taken from [11]

The system would look something like this, with the validator nodes listening in on the blockchain state. These validator nodes would then run their algorithm to agree on a state, which the blockchain is currently in. This algorithm would be separate from the one used by the blockchain nodes.

Cactus aims to transfer all types of blockchain objects (Fungible assets, Non-fungible assets, and Data)

Hyperledger cactus is in the early stages of development; while it is a promising idea, they are just in version 0.3. The task they are setting out to solve is not an easy one either, but leaving the communication with the validators to the individual blockchain with smart contracts will reduce their workload.

9 Appendix B: Existing SCM systems, and their blockchains

Name	Type	Chain	Industry	Token	Crypto	Comercial	Year of Creation
Tracr	Permissioned	Ethereum	Expensive Goods	No	No	Yes	2018
Walmart	Permissioned	Hyperledger Fabric	Food	No	No	Yes	2018
MediLedger	Permissioned	Ethereum	Pharmaceutical	Yes	No	No	2017
Medical Chain	Consortium	Hyperledger Fabric /Ethereum	Health	Yes	No	No	2016
Robomed Network	Permissionless	Ethereum	Health	Yes	No	No	2017
CargoX	Permissioned	Ethereum	Shipping	Yes	No	Yes	2018
DexFright	Permissionless	Bitcoin	Shipping	No	Yes	-	2018
MTI	Permissioned	-	Shipping	-	-	-	2018
Everledger	Permissioned	Hyperledger Fabric	Expensive Goods	No	No	Yes	2015
Pharmaledger	Permissioned	Hyperledger Fabric	Health	-	-	-	2020
TradeLense	Permissioned	Hyperledger Fabric	Shipping	-	-	Yes	2018

Table 10: A snapshot of the different SCM and their features

Echtermeyer Hallvard

NTNU
Norwegian University of
Science and Technology