UNIVERSITÉ DE GENÈVE

ARCHITECTURE WHITE PAPER
16INFOTRAV

Architecture White Paper

Teacher Supervisor: Solana Eduardo

External Supervisor: Florent Martin

Author: João Quinta

May 2023

UNIVERSITÉ DE GENÈVE
FACULTÉ DES SCIENCES
Département d'informatique

Contents

1 Introduction **2**

2 Language **2**
2.1 City Actor . . . 2
2.2 Asset . . . 2
2.3 Perimeter . . . 2
2.4 Channel . . . 2
2.5 Topic . . . 2

3 Use Cases **2**
3.1 Use Case 1 . . . 2
3.2 Use Case 2 . . . 3
3.3 Use Case 3 . . . 3

4 Requirements **3**

5 Technologies **3**
5.1 Apache Kafka . . . 3
5.2 Distributed Ledgers(DTL) . . . 4
5.3 Self Sovereign-Identity(SSI) . . . 4
5.4 Key Exchange Protocols(KEP) . . . 4
5.4.1 Group Key Exchange Protocols . . . 4

6 Hyperledger Fabric Blockchain(HFB) **4**
6.1 Types of organizations . . . 4
6.2 Types of Peer Nodes . . . 5
6.3 Endorsement policies . . . 5
6.4 Interactions . . . 5

7 Architecture **5**
7.1 Communication channel . . . 6
7.2 Identification . . . 6
7.3 Federation . . . 7
7.4 Confidentiality . . . 9
7.4.1 Setting . . . 9
7.4.2 Identity Issuance and registration . . . 9
7.4.3 Common key computation . . . 10
7.4.4 New Identity registered . . . 12
7.4.5 Identity removed . . . 13
7.4.6 Key freshness . . . 14
7.5 Integrity and Provenance . . . 15
7.6 Business Transactions . . . 16
7.7 Types of blocks . . . 17

8 Multiple Channels **17**

1 Introduction

This document represents the White Paper of a proposed architecture.
It is not going to be shared outside the scope of my Master Thesis, this is why there are no citations or justifications based on literature reviews.
This document serves as a baseline for the rest of the project.

2 Language

This section will define some keywords that will be used throughout the document.

2.1 City Actor

City Actor represents any organisation that wishes to join the proposed framework, throughout this document, City Actor and organization will be used interchangeably.

2.2 Asset

An asset represents a physical object or a service, that is owned by a specific city actor as illustred in figure 1. I intentionally avoid using the word Digital Twin(DT) as it does not have an official definition. Using Asset as the primary term avoids potential confusion and misinterpretation associated with the term DT.

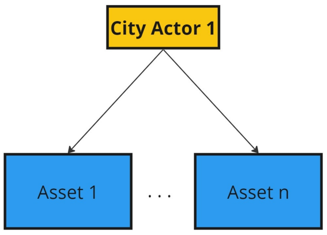

Figure 1: City Actor with n assets

2.3 Perimeter

We will use the word perimeter to describe the allowed organizations in the Hyperledger Fabric Blockchain(HFB).

2.4 Channel

In this document we will assume an unique channel in our HFB, but we should keep in mind that it is possible to create an unlimited number of private channels inside the same HFB. The same City Actor could join an unlimited amount of different channels, and register its assets in the various channels. Subsequent sections will discuss the advantages of the ability to join multiple channels within the network.

2.5 Topic

In the Apache Kafka infrastructure, a topic functions like a chat room, where authorized assets can produce messages and subscribe to receive messages from others. Each topic is associated with a specific HFB channel.

3 Use Cases

3.1 Use Case 1

To understand the thought process we first must understand the needs of the users (organizations).
The use case is quite simple, 2 organizations want their assets, who are either data creators or own data, to exchange data securely in a predefined perimeter, figure 2 bellow is a good example of such communication.
For example: a truck delivery company, wants its trucks to be able to share their live position to the store to which the trucks are delivering goods.

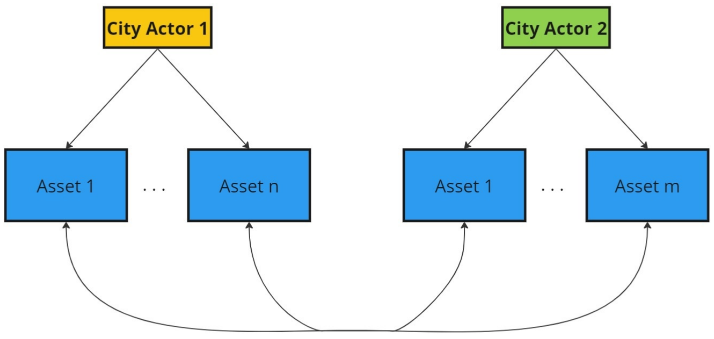

Figure 2: Communication channel between assets of different City Actors

3.2 Use Case 2

Following Use Case 1, present organizations want to add a new organization to the existing perimeter, while ensuring the privacy and security of previous communications.
For example: the new organization could be the farmer, its asset being its stock. The asset could share its own live stock availability with the shops.

3.3 Use Case 3

Following Use Case 2, organizations want to implement traceability to some of their business transactions, as well as automated payments without additional Trusted Third Parties(TTP) and associated costs.
For example: the store might want to set up its order with the farmer within the BC, rather than going through other systems that would involve additional costs to assure trust.

4 Requirements

To create a capable and trusted platform for data sharing among CA, certain key requirements must be met. These requirements ensure that all organizations feel secure and confident in using the platform for their communication needs.
To that end, we must guarantee:

(1) **Availability**: the platform must always be online.

(2) **Reliability**: an asset experiencing connectivity issues must be able to retrieve messages that may have been missed during the connectivity outage.

(3) **Efficiency**: The platform should enable group communication among assets, rather than isolated individual communication, to optimize communication efficiency.

(4) **Confidentiality**: only those within the predefined perimeter will be able to access and read the data.

(5) **Integrity**: the data wasn't modified, and is complete.

(6) **Provenance**: all messages can be verified as originating from a trusted and genuine source.

5 Technologies

This section briefly introduces the key technologies utilized in the proposed architecture.

5.1 Apache Kafka

Apache Kafka is a distributed streaming platform that enables efficient, high-throughput, fault-tolerant, and scalable data streaming.

5.2 Distributed Ledgers(DTL)

Distributed ledgers play a crucial role in ensuring immutability and secure transaction tracking in the proposed system.

5.3 Self Sovereign-Identity(SSI)

SSI allows for secure, privacy-preserving identity verification without relying on Trusted Third Parties, aligning with the system's objective of decentralized trust.

5.4 Key Exchange Protocols(KEP)

KEPs, such as Diffie-Hellman, facilitate secure communication by enabling private and secure key exchanges between entities in the proposed architecture.

5.4.1 Group Key Exchange Protocols

Group Key Exchange Protocols, like Binary Tree Group Diffie-Hellman (BTDH), enable efficient N:N communication, improving the scalability of secure messaging within the system.

6 Hyperledger Fabric Blockchain(HFB)

In this section, we will provide an in-depth analysis of the default components required for the use of HFB.

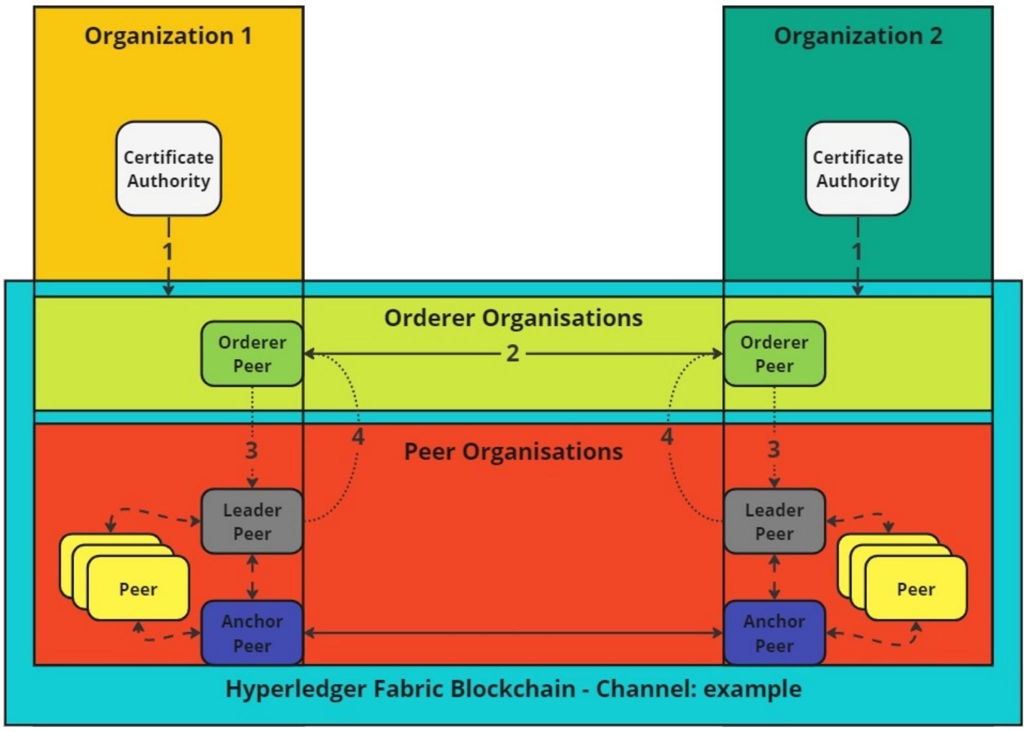

Figure 3: Hyperledger Fabric Blockchain general architecture between two organizations

6.1 Types of organizations

There are two categories of organizations, Orderer organizations and Peer organizations. The same organization may be an Orderer organization, as well as a Peer organization. These different categories simply represent different responsabiltites and capabilities withing the network.

Orderer organizations own Orderer Peers, responsible for maintaining the Orderer service. They manage transaction ordering and block distribution to other network participants but do not have access to the actual transaction

contents.
In a test environment consensus can be performed with protocol SOLO, an unique Orderer peer exists, meaning the Orderer service is centralized. However, in a production environment, the Orderer service would ideally be maintained by all, or some of the organizations in the channel, and the consensus would be achieve via RAFT or KAFKA.
In figure 3 both organizations have an Orderer peer, and manage the Orderer service together.

Peer organizations own Peer nodes of various types, responsible for maintaining the actual ledger and execute chaincodes (SC) in the network as well as endorsing transactions. In figure 3 organizations 1 and 2 manage various types of peer nodes.

Both organizations in figure 3 have a CA that is issuing a X.509 based on Public Key Infrastructure(PKI), the standard certificates in HFB, to each of its peer nodes. Just like before, in a test environment we may have a single CA, however in a production environment each organizations has its own CA.

6.2 Types of Peer Nodes

- Endorsing peer: is responsible for simulating and endorsing transactions by executing chaincode, checking the endorsement policy, and signing the transaction proposal responses.
- Leader peer: is elected among the endorsing peers within an organization to communicate with the orderer service for receiving new blocks and distributing them among other peers in the organization. The leader peer is responsible for ensuring that all other peers in the organization receive new blocks and stay in sync with the network.
- Anchor peer: serves as a communication bridge between different organizations in the network. It enables the discovery of other peers in the network and helps maintain the overall connectivity of the network.
- Peer: verifies that the transactions have been endorsed correctly, and then updates the ledger.They do not endorse transactions or execute chaincode.

6.3 Endorsement policies

Endorsement policies are the set of rules that define the conditions that must be met for a transaction to be considered valid in a Hyperledger Fabric network. It specifies which organizations endorsing peers must endorse (approve) a transaction before it can be considered valid and committed to the ledger. $\Longrightarrow$ defines which peers are endorsing peers. Each SC may have a different Endorsement policy.

6.4 Interactions

In this section we will analyse the interactions between each type of nodes.

(1) For each node under an organization, there needs to be a certificate issued by the Certificate Authority

(2) Orderer peers communicate to reach consensus in a decentralized manner

(3) Orderer peers communicate with each organizations Leader Peers to transmit new blocks added to the BC

(4) Endorsing peers communicate to the orderer the new transactions

— Anchor peers are available publicly, so that communication between organizations is possible

- - - Leader peers use the gossip protocol to transmit the new blocks received to all other peers in the organization

Each item in this list refers to a specific arrow label in the figure 3.

7 Architecture

In this section we will propose a system framework where different organizations within an urban context may connect themselves and share data securely and with confidence.
Having in mind all technologies that were briefly introduced in this document, the use cases as well as the requirements seen in section 4, we will build the Architecture step by step.

7.1 Communication channel

resiliance plutot que reliability

Our choice of communication broker solves requirements (1), (2) and (3).

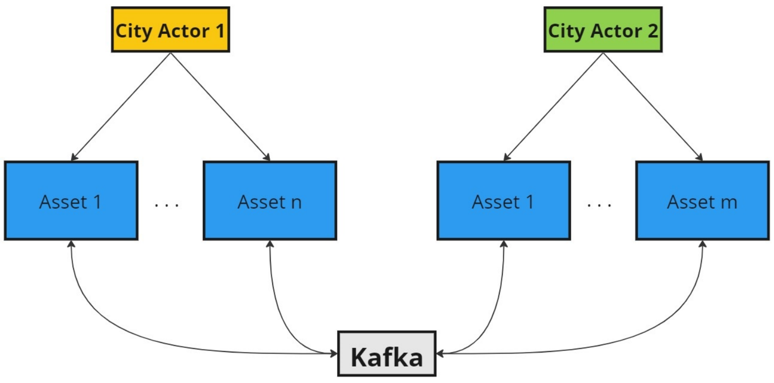

Figure 4: Communication channel

Due to Kafka's decentralized architecture, it is constantly available (1) and can easily scale to accommodate the needs of various organizations.
If an asset were to lose connection momentarily, it would still be able to consume missed messages, since they are stored (2) in the Kafka topic.
Kafka also enables group communication, every entity can be both a producer as well as a consumer (3).

However, if an attacker were to gain unauthorized access to the topic he would be able to read every message, breaking requirement (4). Additionally, Assets won't be able to self-authenticate to gain access to the kafka topic, meaning we'd still rely in a TTP for authentication that would need to be available at all times to uphold requirement (1).

7.2 Identification

Our proposed scheme will be based on HFB, which doesn't natively support SSI integration. While there are projects like Hyperledger Indy that support SSI, they are quite limited for Use Cases that need more functionalities such as transaction traceability, as described in section 3.3.

To overcome this limitation, we will utilize SC within HFB, which will enable the implementation of SSI.
HFB requires the use of Certificate X.509 based on PKI, that are issued by Certificate Authorities(CAs) for maintaining its functionalities such as the gossip protocol. Each organization manages a CA, meaning only organizations can issue certificates for their nodes
Unfortunately, these certificates still won't allow assets to self authenticate to a service provider. This can be a problem if the verification entity were to be unavailable, or compromised.

To achieve self authentication we will add a Decentralized Identifier (DID) to the certificate, decentralizing identity management. The DID will be the hash of the asset's public key.
This adds a new requirement, Kafka's access proxy must be able to see the ledger to verify identities. Figure 5 bellow shows the whole process:

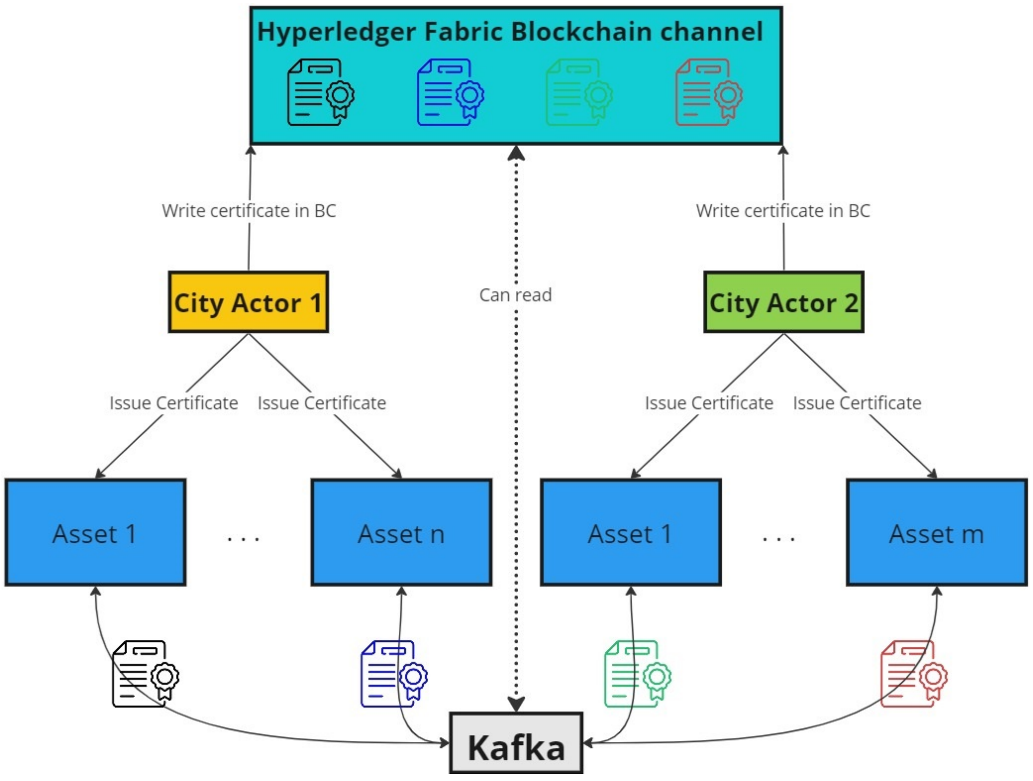

Figure 5: Communication channel with self authentication

By adding Kafka as an organization that can access the BC, we introduce a new challenge: which organization will "own/maintain" the Kafka node that can access the ledger?

Moreover, to integrate the scheme presented in figured 5 into the general architecture shown in figure 3. We will represent assets as peer nodes in HFB, but for the purpose of this document, they are the same thing:

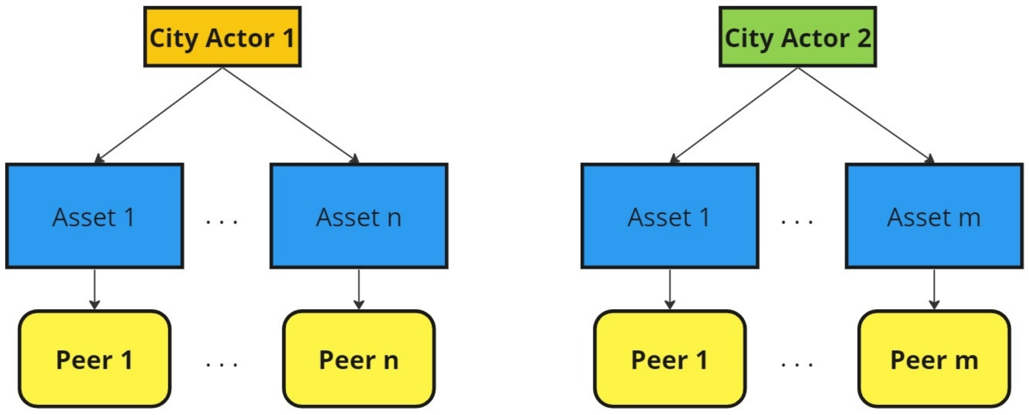

Figure 6: Assets transformation to Peer Nodes

They will represent Peer nodes of various types, which were introduced in section 6.2.

7.3 Federation

To address the previously introduced challenge, we propose the creation of a new organization, the Federation.
The Federation doesn't have any power over the BC or Kafka, its primary purpose is to maintain both infrastructures. I won't be able to have authority over the various organizations, it will only offer a platform, through which they can share information. With Figure 3 in mind, we propose the following architecture with the Federation shown in figure

8.
For simplicity some notations were modified:

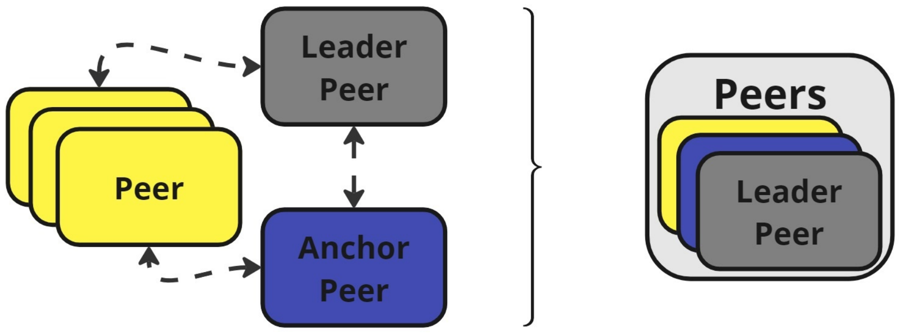

Figure 7: New peer nodes notation

Communication links were also removed, assume that all necessary communication links are present.

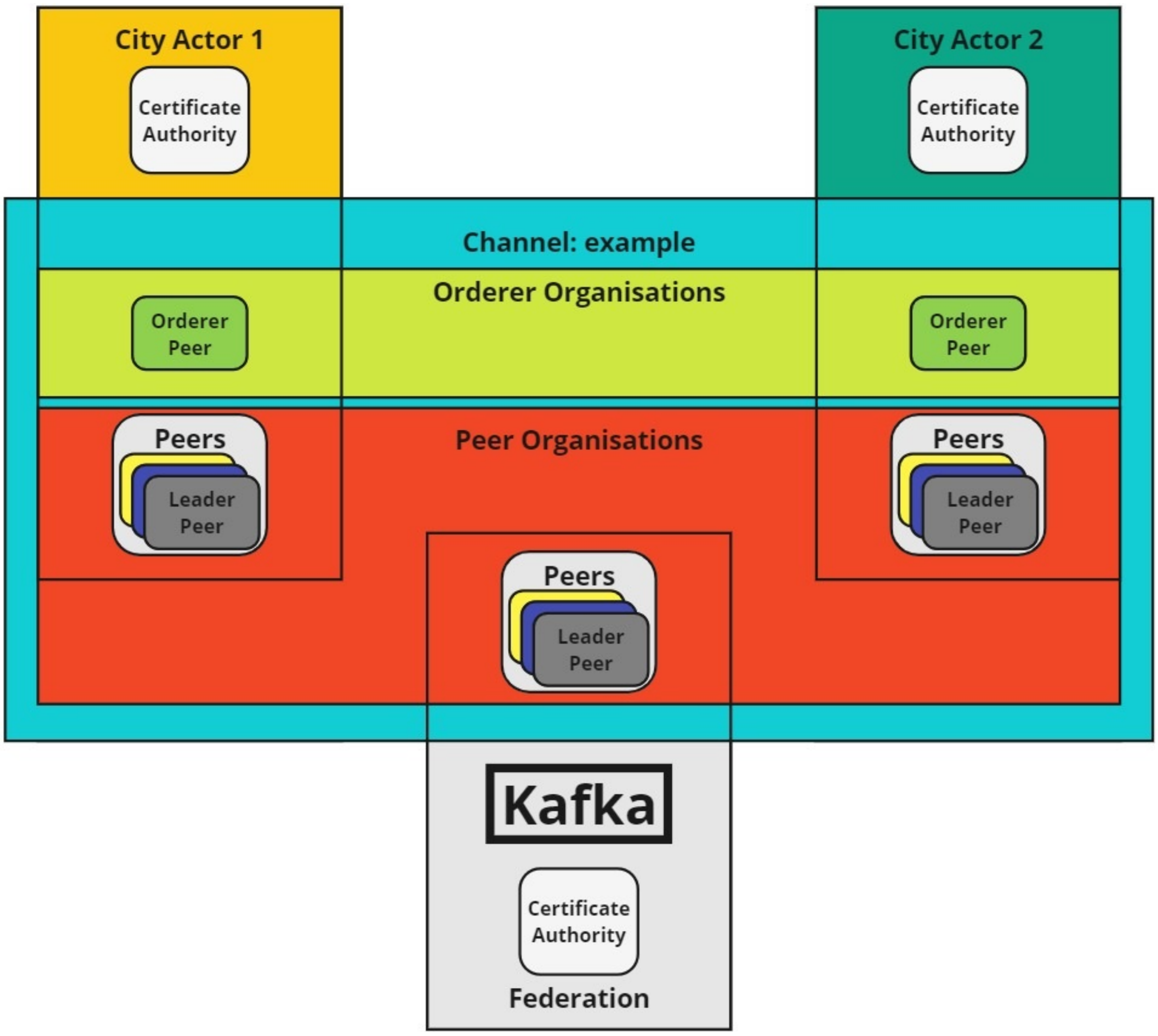

Figure 8: Hyperledger Foundation Architecture with Federation

The federation owns a CA, and maintains various peer nodes, as required for the well functioning of HLF. In the proposed scheme it does not own an Orderer Peer, but it would be possible.
Finally, the federation owns the Kafka servers, through which Assets will communicate.

7.4 Confidentiality

In this section we will go through the process of assuring confidentiality, a requirement announced in section 4.

7.4.1 Setting

As described in 7.2, identity certificates will be registered to the BC by the various City Actors. For the remainder of this section we will refer to the Use Case 1 described in section 3.1, using the provided example. Figure 9 illustrates how an asset, its identity, and its public key are visualized:

Asset
A
Identity
A
Key
A

Figure 9: Left: Asset A - Middle: Asset A's Identity registered - Right: Asset A's Public Key

City actor 1, a delivering company owns 2 assets (delivery trucks), called A and B. City actor 2, a company that manages shops, also owns 2 assets (shops) called C and D. As shown in the figure bellow:

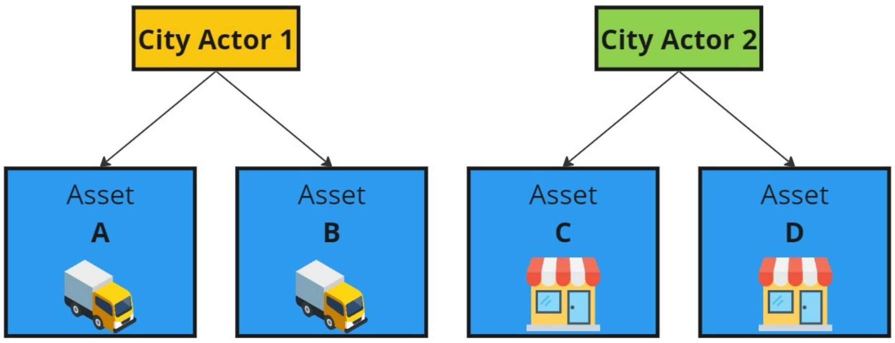

Figure 10: Scenario Use Case 1

It is important to note that these assets will ultimately be represented as Peer nodes of various types in HFB.

7.4.2 Identity Issuance and registration

In HFB all the assets will be represented as peer nodes, and each will own an unique Certificate X.509 based on PKI issued by the organization's CA, with the addition of an unique DID. Following Certificate issuance, it must be registered in the HFB, as depicted in the figure bellow:

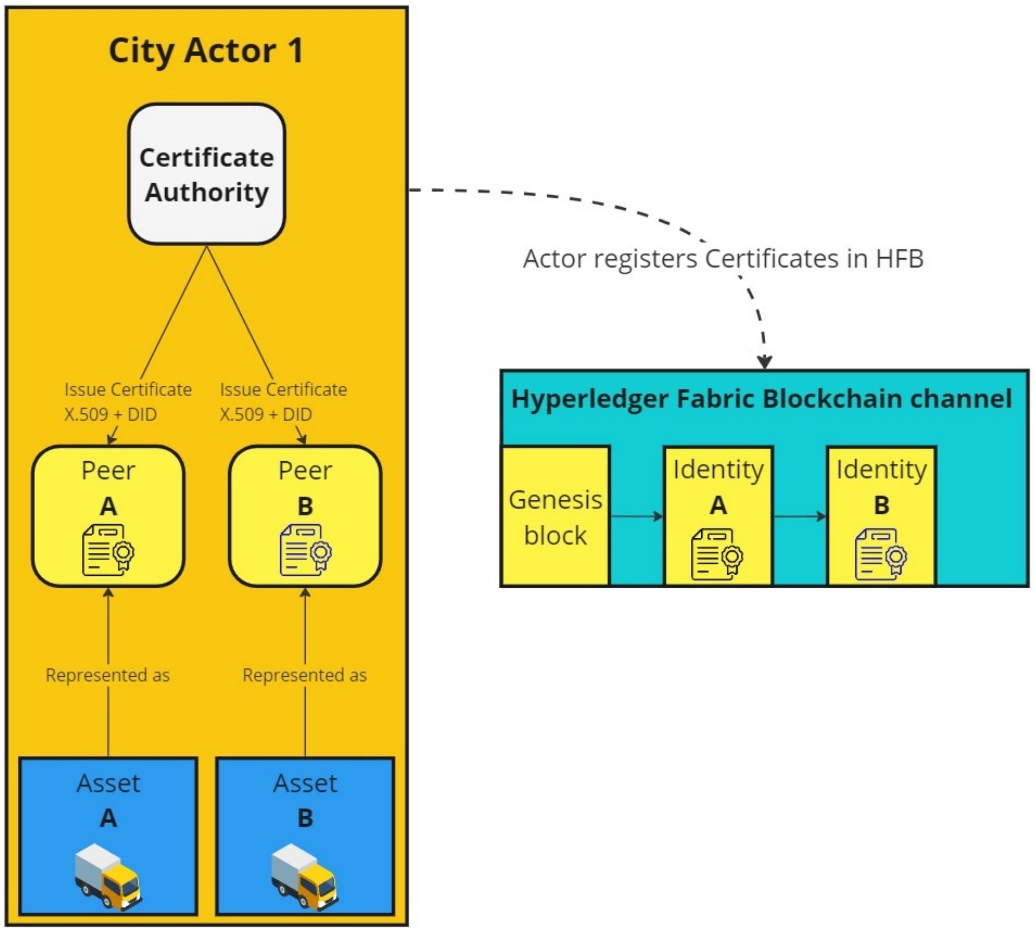
City Actor 1
Certificate Authority
Issue Certificate X.509 + DID
Issue Certificate X.509 + DID
Peer A
Peer B
Represented as
Represented as
Asset A
Asset B
Actor registers Certificates in HFB
Hyperledger Fabric Blockchain channel
Genesis block
Identity A
Identity B

Figure 11: Scenario: Certificate issuance and registration in Hyperledger Fabric Blockchain

The certificate has many fields, one of them is a Public key, this public key has a private key counterpart only known by the asset itself. The city actor who owns the asset, does not control the private key!

The explanation of how Asset A retrieves the public key of Asset B was kept at a high level for simplicity.

7.4.3 Common key computation

Lets assume both City Actors registered their Asset's identities. This is what the BC would look like:

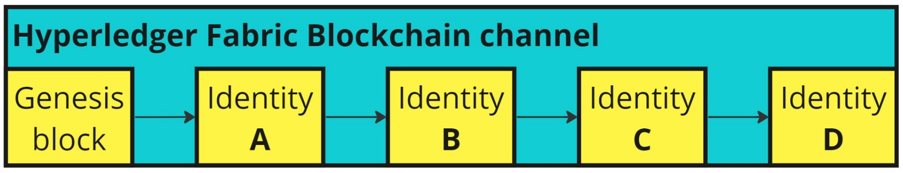

Figure 12: Blockchain status after identity registration

We want to use Kafka as an efficient group communication channel. Additionally, we want to be able to encrypt and decrypt with an unique secret key. Every asset registered in the HFB must be able to compute the common secrete key. In this section we will analyse an algorithm that allows every asset registered in the HFB to compute a common secret key.

The algorithm leverages the property of peer nodes in a BC network, where each node has an exact copy of the blockchain in local storage, with all blocks in the same order.

Before we dive into the algorithm, it's essential to understand what a binary tree is. A binary tree is a tree data structure in which each node is either a leaf (terminal node) or has exactly two children nodes. It is impossible for a node to be the parent of a single node. The figure bellow shows a generic binary tree of depth 3 with 7 total nodes, from which 4 are leafs (yellow), 2 intermediary nodes (grey), and the root (green).

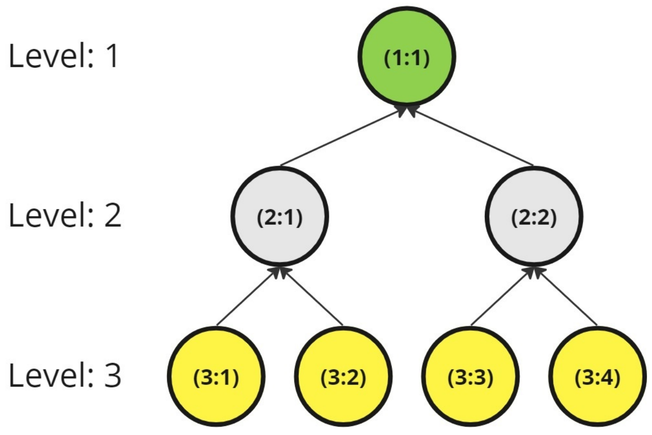

Figure 13: Generic binary tree of depth = 3

Each node has an unique name: (depth from the top: position from the left). Node (2:2) is on level 2, and is the second node from the left in that level.
Every node, apart from the root, has a sibling node. Sibling nodes share the same parent node.

We will use a binary tree to represent the registered identities in the blockchain. The binary tree is deterministically computed from the blockchain. The tree's leaf nodes represent registered identity in the BC.
The figure bellow shows the binary tree that can be deterministically computed from figure 12.

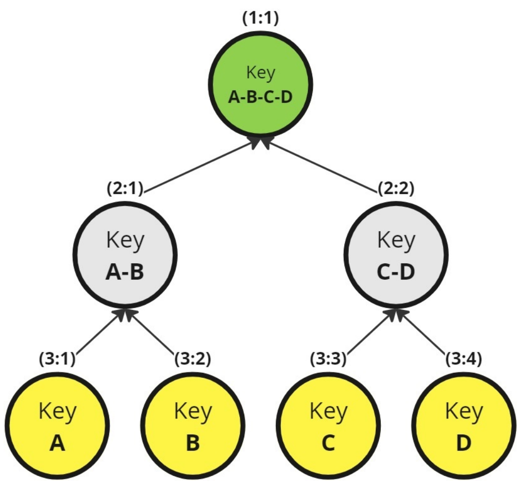

Figure 14: Binary tree state

Every leaf node in the tree represents an Asset registered in the BC. Therefore, each leaf node owns a private/public key pair. Every other node's private/public key pair is unknown, but can easily be computed from their children's public/private key pairs.
For example Node's (2:1) private key can only be computed by node (3:1) and node (3:2).

Lets run the algorithm assuming you are node (3:1) which represents Asset A.
You are the only one to know your private key, and you know your sibling's public key, since it is registered in your locally stored BC.

Step 1: Compute node's (2:1) private key using your own private key, and your sibling's public key.

Step 2: Compute node's (2:1) public key from its private key.

Step 3 a: Share node's (2:1) public key with nodes that can't compute it themselves, in this case nodes (3:3) and (3:4).

Step 3 b: While you computed node's (2:1) private/public key pair. Nodes (3:3) and (3:4) did the same thing for their parent node (2:2). They share with you node's (2:2) public key.

Step 4: Now that you know node's (2:1) private key, and node's (2:2) public key, you can compute node's (1:1) private key.

The private key of node (1:1) represents the commonly computed secret key (k).
While the underlying formulas and computations for steps 1, 2 and 4 are crucial to the algorithm's functioning, I have chosen not to delve into the details in this explanation. This is to maintain a focus on the high-level understanding of the algorithm and its overall process.

At this point, everyone can use key (k) to encrypt and decrypt messages. Assuming three messages encrypted with key (k) are produced for the topic, this would be the state of the Kafka topic:

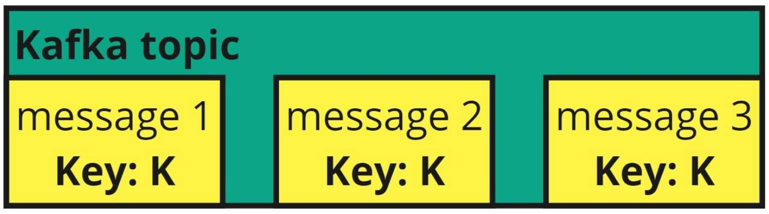

Figure 15: Kafka state after 3 messages sent encrypted with key k

7.4.4 New Identity registered

Let's assume we transition from Use Case 1 to Use Case 2, as described in sections 3.1 and 3.2. A new organization with 1 asset, called E, joins the private BC. As shown in the figure bellow:

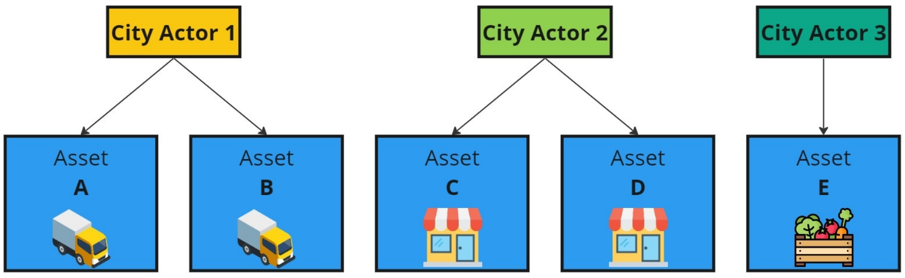

Figure 16: Scenario Use Case 2

This new identity must be registered to the BC. Let's inspect the state of the BC at this stage:

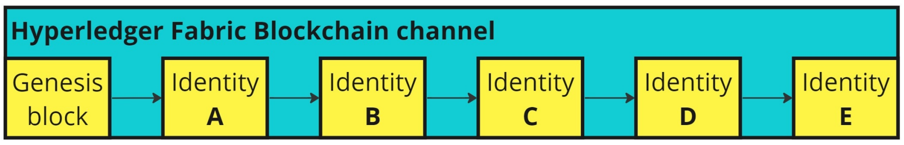

Figure 17: Blockchain status after new identity registration

As soon as every peer in the network receives and verifies the new block with the new identity, a new common key (k') must be computed. The binary tree will now look like this:

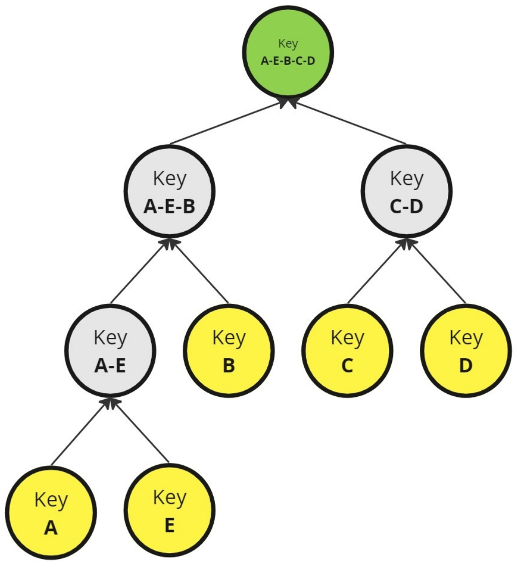

Figure 18: Binary tree state with additional identity

The algorithm defining the binary tree computation handles new member additions in a deterministic way, ensuring every peer in the network computes the same binary tree. Additionally, the new Peer will have access to the BC therefore having the ability to compute the same binary tree.

With the new common key (k') a new message is produced to the topic. Let's inspect the new state of the Kafka topic:

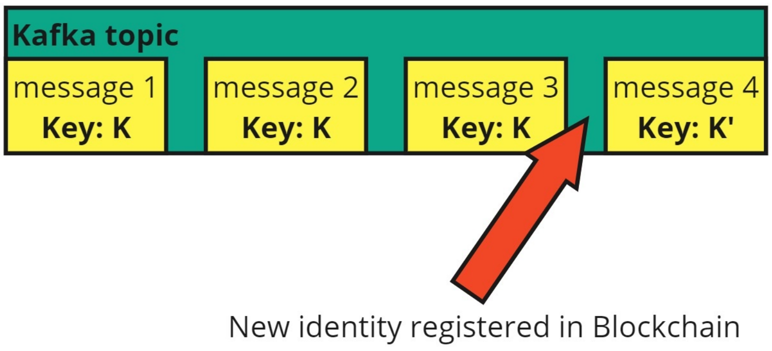

Figure 19: Kafka state after 4 messages sent encrypted with key (k) and key (k')

While the new peer can consume the previous 3 messages in the topic, it will never be able to decrypt them, since he doesn't have access to key (k), and will never be able to compute it.
To compute any common key using this algorithm, your own private/public key pairs must be one of the leafs in the initial binary tree.

7.4.5 Identity removed

As we were able to add new City Actors and new assets to our BC. We also want to be able to remove them. Due to the property of immutability of the BC, it is not possible to directly remove the block that registered an identity. Instead, we can address this issue by adding a new block that invalidates the corresponding identity registration block. Visually it would look like this:

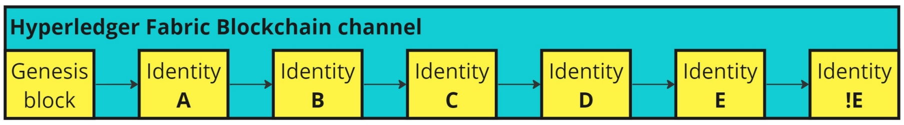

Figure 20: Blockchain status after identity removal - !E

As illustrated in the figure above, the last block informs all peers in the network that the identity registered as "E" is no longer valid.
This approach allows the architecture to handle the revocation of certificates.

7.4.6 Key freshness

In the previous three sections the state of the ledger changed quite a bit. We started at section 7.4.3 and computed common key (k). Following that, a new identity was registered in section 7.4.4 and common key (k') was computed. Finally in section 7.4.5, we invalidated the previously registered identity.
After this sequence of events, the identities that are still registered and valid in the BC are the original 4 assets of city actors 1 and 2 as described in section 7.4.3.

At this point, if we were to compute the common secrete key using the same algorithm as before, we would end up with the exact same secrete key (k). This is due to the fact that the common key (k) depends solely on the registered and valid identities in the BC at the time of computation.

This is an undesirable result. Even if no new asset were to be registered to our BC, we would still want to change common secret key (k) often enough. Assuring key freshness is vital for confidentiality, as it makes it more difficult to mount know-plaintext attacks.

To address this issue, a new random state (R) is introduced to the BC, which could be implemented via a simple SC. The SC would execute under certain conditions, such as an identity being registered as invalid, or reaching the set number of maximum messages sent using the same key (k). If executed, the SC would generate a private/public key pair, publish the public key as a new block transaction in the BC, and delete its private counterpart. This SC could be owned by any of the organizations in the BC perimeter. This is what the BC would look like with this random state (R) at the initial use case 1:

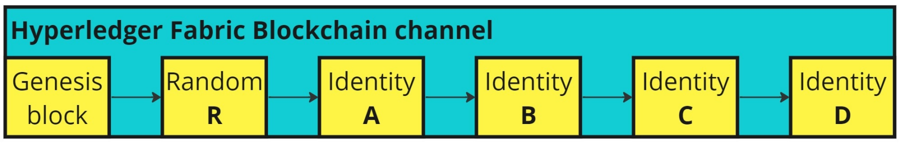

Figure 21: Blockchain status with random state (R)

And its corresponding binary tree:

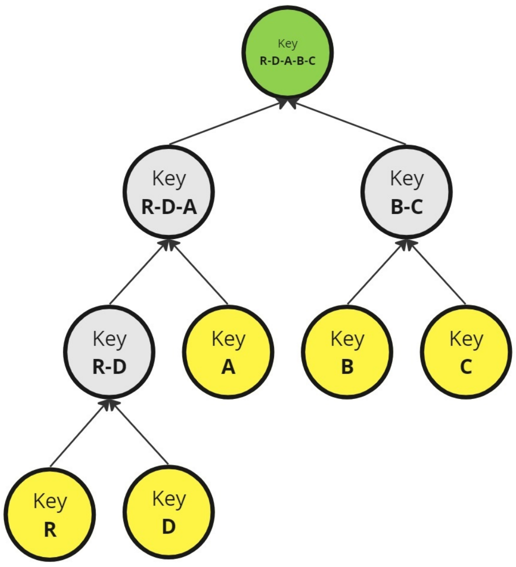

Figure 22: Binary tree state with random state (R)

Adding and invalidating identity E, would fulfil the preconditions for the SC execution, which would result in a new random state (R') being registered in the BC. As shown in the figure below:

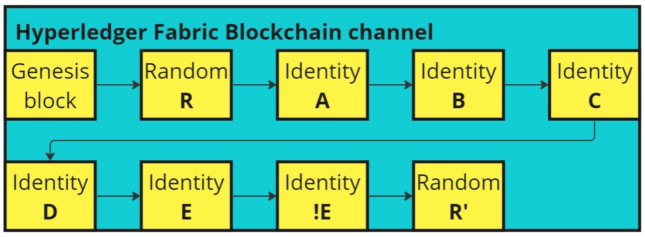

Figure 23: Blockchain status with random state (R') after invalidation of E

The following figure shows the state of the equivalent binary trees before and after the invalidation of identity E and the new random state (R'):

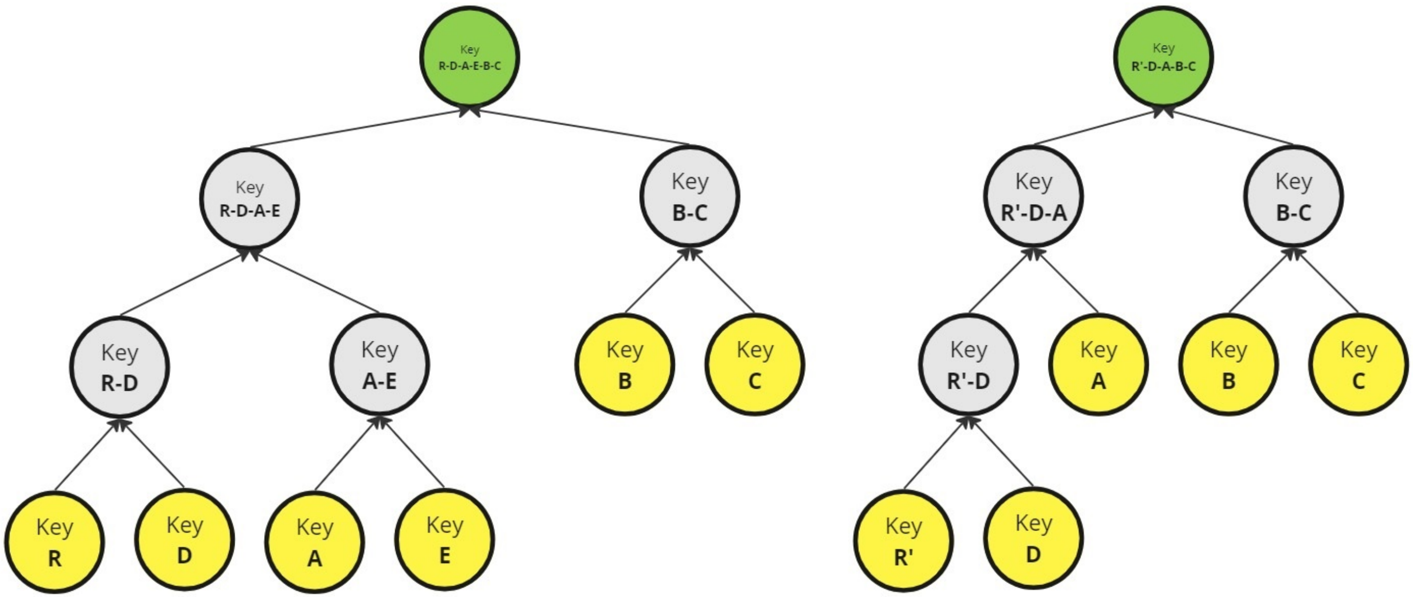

Figure 24: Left: Binary tree state with identity E and random state (R) - Right: Binary tree state with invalidated E and random state (R')

It is important to note, that if a new random state (R') is registered to the ledger, the resulting binary tree will not have both (R) and (R') states, rather, it will replace the old state (R) with new state (R').

Thanks to the update to the random state from (R) to (R') after invalidation of identity E. We assure that the commonly computed key (k) from the binary tree in 22, is different from the commonly computed key (k') from the binary tree in 24 (right side). Even though in both trees the exact same identities are present.

7.5 Integrity and Provenance

So far the proposed architecture has proposed a solution for the requirements described in section 4 (1) through (4). In this section we will focus in requirements (5) and (6).
Both integrity and provenance can be can be addressed by having the producer of a message sign the hash of the message that was written. The figure bellow illustrates this process:

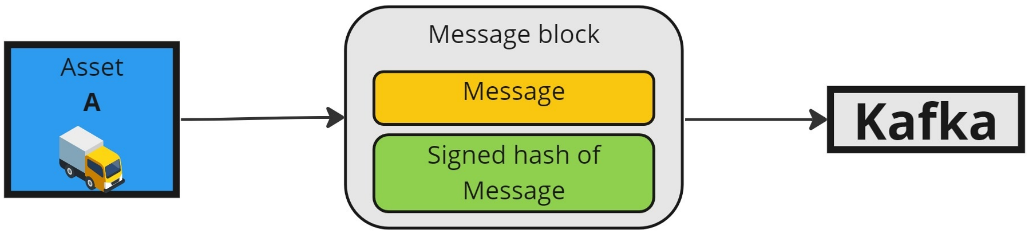

Figure 25: Hash of message signed

The federation, who controls the Kafka topics, could in theory delete messages produced to the topic. Meaning that the federation holds the power to silence Assets at will, and simply signing the messages won't solve this problem. To improve our solution and avoid this problem, we can add to the message the hash of the previous message produced, creating a virtual chain of messages. This way each consumer can be confident that no messages were removed from the topic as illustrated by the figure below:

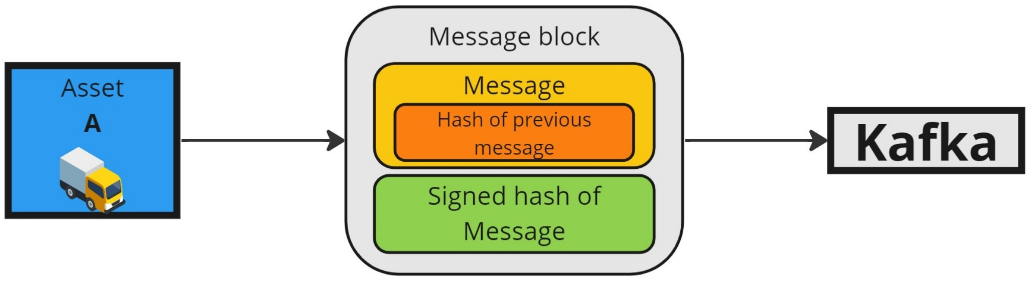

Figure 26: Hash of message with hash of previous message signed

With the addition of this new security measure, the architecture is compliant with all the requirements announced in section 4.

7.6 Business Transactions

A typical business transaction usually involves three main steps: ordering a service, delivering the service, and receiving payment for the service. A service might take various forms.
In section 3.3 we describe a use case which requires additional functionalities such as business transaction traceability. Such functionalities enable organizations to manage ordering and payment for a service within the BC network.

These business transactions can be managed using SCs. As the execution of an SC is registered on the BC, traceability of every business transaction is ensured within the immutable BC. The following sequence diagram shows how 3 assets from 3 different organizations would interact with the SC to carry out the business transactions:

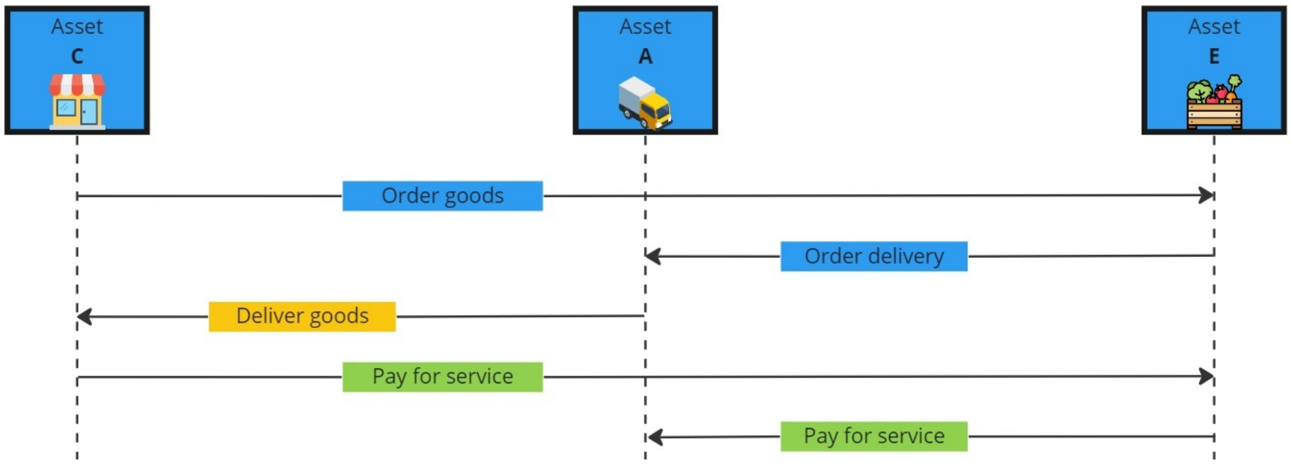

Figure 27: Sequence diagram of 2 business transactions between 3 organizations

Each of these business transactions can be implemented as SC functions that are executed on the BC. The BC registers the execution of these functions as transactions, ensuring traceability and immutability. The BC may even hold payment after a service is ordered, until it is delivered, assuring payment.
This is what the BC would look like after a transaction:

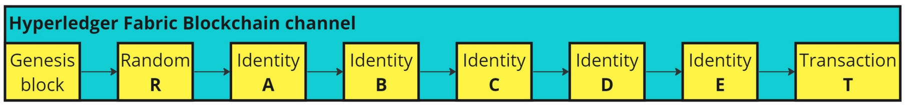

Figure 28: Blockchain status after a business transaction

7.7 Types of blocks

Throughout the construction of this architecture, various types of data blocks have been registered in the BC. Here, we summarize and provide more detail on each type of block:

(1) **Genesis block**: The first block in the blockchain, which serves as the foundation for subsequent blocks to build upon.

(2) **Federation identity block**: This block represents the registration of identities for the Peer nodes managed by the federation. These blocks would be ignored in the common key generation process described in section 7.4.3, preventing the federation Peers from decrypting messages produced in Kafka topics.

(3) **Identity block**: Represents the registration of asset identities on the BC. These blocks correspond the identities of Assets that can exchange information through Kafka topic.

(4) **Identity revocation blocks**: As described in section 7.4.5, this blocks represent the invalidation of a previously registered identity. It ensures that revoked assets cannot access secure communications.

(5) **Random blocks**: Assure key freshness, as described in section 7.4.6. These blocks introduce randomness into the computation of the secrete common key (k) used to encrypt messages produced to Kafka topics.

(6) **Transaction blocks**: Represent business transactions between organizations on the BC, providing traceability as described in section 7.6.

The following figure illustrates the BC with all possible types of blocks:

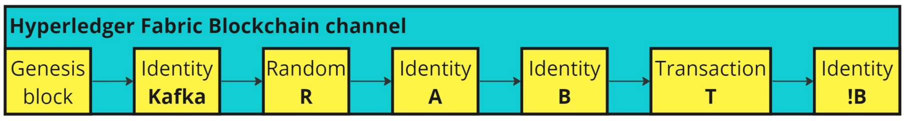

Figure 29: Blockchain with every type of blocks

8 Multiple Channels

So far we haven't explored the utility of having the ability of creating and joining as many private channels as we may require.
Assume you are the farmer, and you want to deliver your product to stores, but don't want to share the cost of the delivery with the store you are delivering to, as sharing that information might be damaging to your business. HFB allows you to create multiple channels of secure communication. The figure bellow illustrates this phenomena:

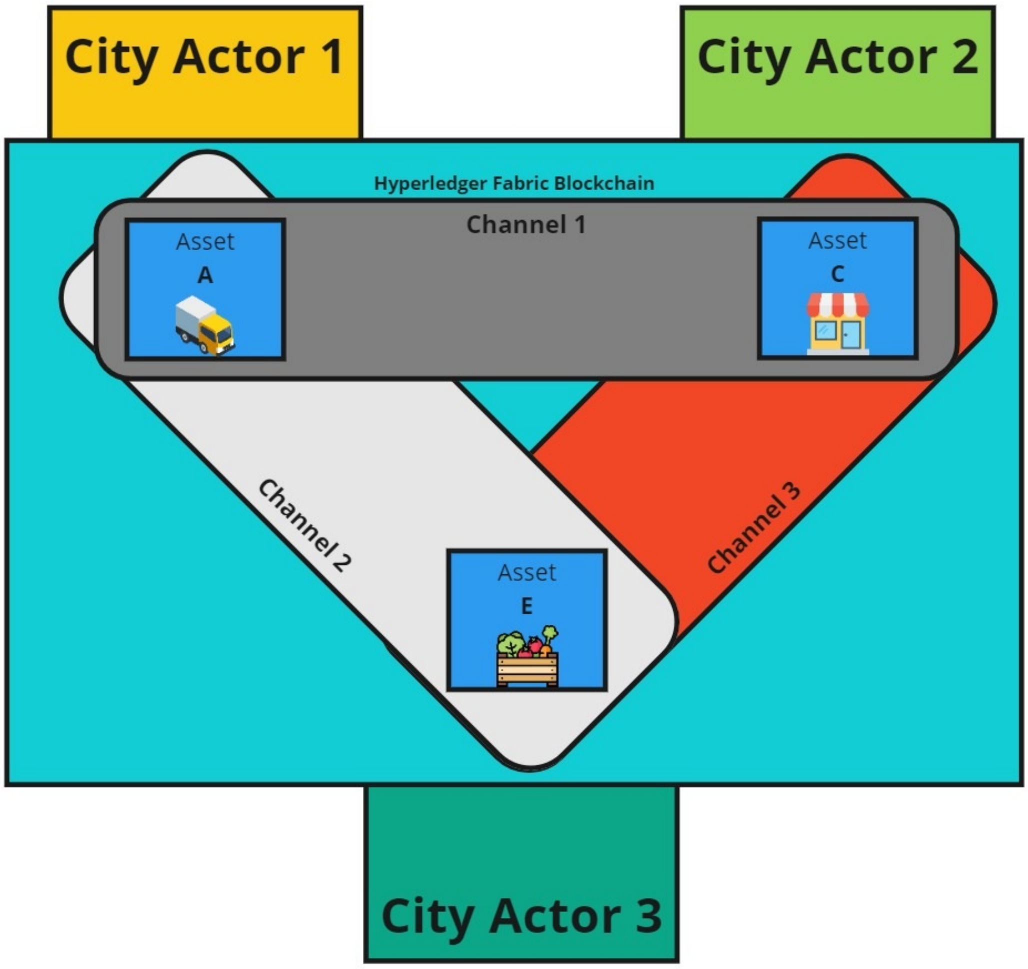

Figure 30: Hyperledger Fabric Blockchain with multiple channels - Federation was abstracted for simplicity

limitations =>

changer l endroit ou sont les limitations (random key R)
~(suppression de message avec message hash)
-multiple channels for lightweight (business transactions vs identity)

uma: opaque -- acces central --