UNIVERSITÉ DE GENÈVE

MASTER THESIS
16INFOTRAV

Cybersecurity solutions for Urban Digital Twins: A decentralised framework

Teacher Supervisor: Solana Eduardo

External Supervisor: Martin Florent

Author: João Quinta

August 2023

UNIVERSITÉ DE GENÈVE

FACULTÉ DES SCIENCES
Département d'informatique

Contents

1 Introduction

With the rise of digitization and the increasing complexity of urban environments, there has been a growing need for sustainable urban development, the concept of Smart Cities has emerged as a promising solution. Smart Cities leverage cutting-edge technology to optimize various aspects of urban planning, including water consumption, transportation, waste management, and overall quality of life. The cornerstone of Smart Cities development is the concept of Digital Twins (DTs), a virtual replica of physical assets and systems that enable real-time monitoring, analysis and optimization.

The heart of DT lies in its ability to capture, monitor and optimize the performance of its physical replica. Urban Digital Twins (UDTs) can seamlessly combine data points from multiple DTs, therefore providing invaluable insights into urban infrastructure. UDTs are full of potential, while each individual DT offers insights into a specific urban asset or system, when integrated into UDTs, they create a synergy. This collective insight offers a comprehensive, interconnected view of urban infrastructure, enabling a holistic approach to city management that is more complete and impactful than considering each DT in isolation. This wealth of information empowers decision-makers to optimize resource allocation, reduce environmental impact, enhance operational efficiency and improve overall livability of cities.

As Smart Cities continue to evolve, DTs are becoming increasingly prevalent across multiple domains such as transportation, healthcare and infrastructure. From optimizing traffic flow to enabling personalized healthcare services, the applications of DTs in the urban context are vast and transformative. With these vast opportunities also come challenges, while the potential benefits of UDTs in urban planning, infrastructure development, and public service optimization are undeniable, Smart Cities platforms involve many actors and raise challenges such as data ownership, data authenticity, interoperability, and overall cyber-security. As we venture into an era where data becomes the new currency, there is an evident need for a robust digital identity system that ensures data sovereignty and security. Self-Sovereign Identity (SSI) emerges as a crucial solution in this context, enabling individuals to control their personal data, ensuring its authenticity and reducing dependencies on centralized authorities. Ushering in the idea of SSI provides not just security but also empowerment in the digital domain.

This thesis dives deep into the realm of DTs, their urban counterpart, the intricate workings of SSI, and the potential of Distributed Ledger Technology (DLT), particularly the Hyperledger Fabric Blockchain (HFB). We will further explore the theoretical foundations, analyze use cases, and culminate with the presentation of a novel architectural decentralized framework that seeks to leverage these technologies for enhancing UDTs. Our primary objective is to provide insights into how blockchain (BC) and SSI can address the challenges of UDTs, potentially paving the way for a smarter, safer and more sustainable urban future.

The following chapters will illuminate on related works in the field, the fundamental theoretical principles, the detailed architecture of the proposed solution, and a thorough evaluation of the same. Through this research, we aim to provide a comprehensive understanding of the intertwined future of UDTs, BC and SSI. This intersection not only dictates how we efficiently manage and optimize our cities but also how we ensure that such digital transformations are secure, transparent, and put citizens safety at the heart of this evolution. Embracing this relationship now will shape the resilience and adaptability of our urban futures, making it imperative for current research and policy directions.

2 Related Work

This section reviews the extensive body of literature that serves as the foundation for this research. Our focus has narrowed down to exploring DTs and UDTs. We will review each subject separately, discussing its development, real-world applications, challenges, and key research studies. This will allows us to identify gaps in the existing knowledge and establish the areas where our research will contribute.

2.1 Digital Twin (DT)

First of all we need to define what is a DT, and which challenges lay ahead in the research of this field.

2.1.1 Introduction to DT

A DT can be described as a virtual instance of a Physical Twin (PT), the PT can be an object, system or even process, enabling real-time monitoring, analysis and optimization throughout the PT's life cycle. It is important to note that the definition of a DT may vary somewhat depending on the context of application. Nonetheless, the core concept of a virtual instance that mirrors its physical counterpart for context-dependent purposes remains true across these variations [1].

In [1, 2] the authors mention common misconceptions and highlight the different between digital models, digital shadows and digital twins. Figure 1 illustrates these distinctions:

- **Digital Model** or mirror model is a digital representation of a pre-existing physical object. However, if the PT were to evolve, the digital model won't change accordingly, this is due to the fact that there is no automatic data exchange between them;
- **Digital Shadow** goes one step further, by adding a one-way data flow between the physical and digital object. This means that the digital object is capable of evolving along side the physical object;
- **Digital Twin** is characterized by a bidirectional data flow between the physical and virtual object. This two-way integrated data flow, allows both twins to evolve side-by-side.

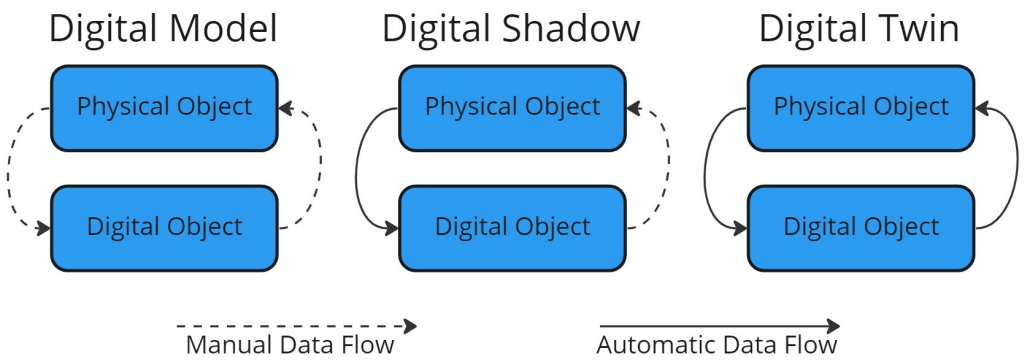


Figure 1: Difference between digital Model, Digital Shadow and Digital Twin

In section 2.1.2, we will address the security issues specifically associated with each model, and discuss potential solutions to overcome these challenges.

The models described above implement a three-dimension architecture, the physical entity its virtual model, and connection between them. In [3] the author extends it into a five-dimension digital twin (5D-DT) architecture, adding data and services dimensions. Figure 2 illustrates the five dimensions.

- **Physical Entity**: Consists of various functional subsystems that perform various tasks, as well as sensors that collect the states of the subsystems;
- **Virtual Entity**: Represents high fidelity integration of the data gathered from the sensors in the physical entity;
- **Services**: Refers to the functionalities the DT provides, such as simulation, optimization, and visualization;

- **Data**: Represents the contextual information of the DT, including real-time data, historical data, and predicted data;
- **Connection**: Is the bi-directional link between both realms, it is thanks to this connection that the Physical Object is able to feed data to the respective Digital Object and the various services.

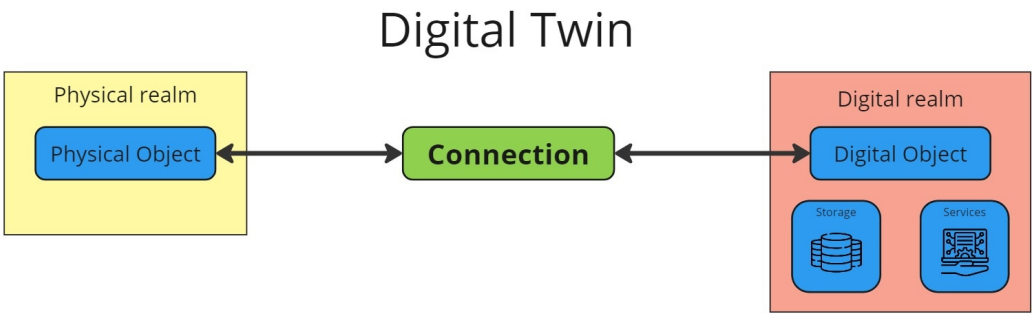


Figure 2: Five-dimension Digital Twin

The 5D-DT framework inherently aligns with the principals of microservice architecture [4], which has become more relevant when developing distributed software applications as it assures **scalability** and **maintainability**. The idea is to create applications by combining small independent software services, each focusing on a single functionality. These services are composed to form a cohesive and functional application [5], using either orchestration or choreography. Orchestration represents a centralized approach, while choreography is a decentralized. It is possible to use both compositions together, depending of the specific application.

Inter-service communication represents another crucial design decision in microservice architecture. Communication can be established using asynchronous or synchronous request-reply patterns, or event-driven asynchronous message exchange. In microservice architecture it is preferred to use event-driven communication as it promotes decoupling of the different services. A commonly used event-driven asynchronous communication solution is Apache Kafka, it offers fault-tolerant, scalable, and stream-based messaging.

By leveraging the modular and flexible nature of microservice architecture, the 5D-DT framework can integrate various technologies and functionalities, making it suitable for designing and implementing DTs, or even Digital Twin Networks (DTN) which we'll introduce in section 2.2.

2.1.2 DT Challenges

By looking at figure 1, we can see that the three different models progressively add layers of functionality and interactivity within the system. However, these new functionalities introduce new security challenges:

- **Digital Model**: The primary challenge is to ensure that all sensors used to capture data of the PT are accurate and reliable. Data inaccuracy can lead to discrepancies between the digital model and its physical counterpart, which might impact any analysis and decisions made based on the digital model;
- **Digital Shadow**: In addition to the accuracy challenges faced in digital models, digital shadows require a secure and constant channel for continuous transmission of data from the sensors incorporated in the PT. Ensuring that the data flow is accurate, timely and protected from potential attacks is critical for maintaining the integrity of the digital shadow. This secure channel will be referred as the Physical-to-Virtual (P2V), in section 2.2.2 we will discuss the security requirements for this communication channel;
- **Digital Twin**: Since the flow of data is bi-directional, the virtual model will be able to modify the state of the PT, this increases the potential risks in the system if an attacker compromises the DT. In such a scenario, the attacker could manipulate the PT, potentially endangering human lives. The flow of data from the DT to the PT, will go through the same P2V channel.

In [6], the author describes how a computer system made a decision based on sensor data, the data was gathered by a single sensor. Because the sensor was faulty, the data it fed didn't accurately describe the current state of the PT. The decision made based on the faulty data resulted in the loss of multiple lives. In this system there were no backup sensors, and no verification was performed on the quality of the data.

Finally, Fuller et al. in [1], highlight additional challenges in the field of DT research. One of them is the lack of a single objective within the research community, resulting in dispersed research efforts, this can hinder the speed at which more people adopt this technology. Another challenge is the lack of standardization within the research of DT, which can delay the progress of DT technologies across different domains.

2.2 Urban Digital Twin (UDT)

2.2.1 Introduction to UDT

DT establish a one-to-one mapping relationship between a physical and virtual space, whereas DTN is characterized by a many-to-many mapping relationship [2]. An UDT is an extension of the DT concept, applied to entire urban environment or city-scale systems, figure 4 is an illustration of potentially involved City Actors. This means that an UDT is essentially a DTN applied to a specific urban context. UDTs enable real-time monitoring, analysis and optimization of various functions and domains, thus playing a crucial role in the development of Smart Cities [7].

In recent years, researchers have investigated various UDT potential applications in various urban related domain. Here are some examples:

- **Visualization**: Enhancing the visualization of the city can bring better understanding of the urban environment, which can lead to fewer city design errors. UDTs can utilize and integrate various visualization tools the improve the current experience. For example in [8] the authors developed a DT of a city, and were able to visualize it almost seamlessly across all scales with the use of virtual and augmented reality;
- **Situational awareness**: Improve the visibility of the city as well as the analysis of urban events and operations. This will be achieved by harvesting multiple data points from the various City Actors. For instance, monitor noise pollution in the city [9] or analyze energy consumption patterns [10];
- **Simulation and prediction**: By analyzing past and present data, UDTs can provide valuable insights into future patterns and plans, enabling optimization of city functions. Some examples include optimizing electric vehicle charging locations [11] and even conducting flooding scenario analysis [12];
- **Integration and collaboration**: UDTs facilitate the integration of complex city elements and promote collaboration amongst City Actors, the goal being the co-development of the city. To that end, two avenues have been proposed. The first one argues that developing multiple DTs for the city may be more achievable than developing a single DT for the whole city [13]. The second one suggests the creation of a collaborative platform, used by the various City Actors. This platform would be used for data sharing amongst City Actors [14].

UDTs are bound to play a large role on how cities will evolve and be designed in the future. By leveraging the power of DT at a city scale, the various City Actors will be able to collaborate and share resources as they never were before, with the common goal of making data-driven decisions dedicated to solve complex challenges of modern cities.

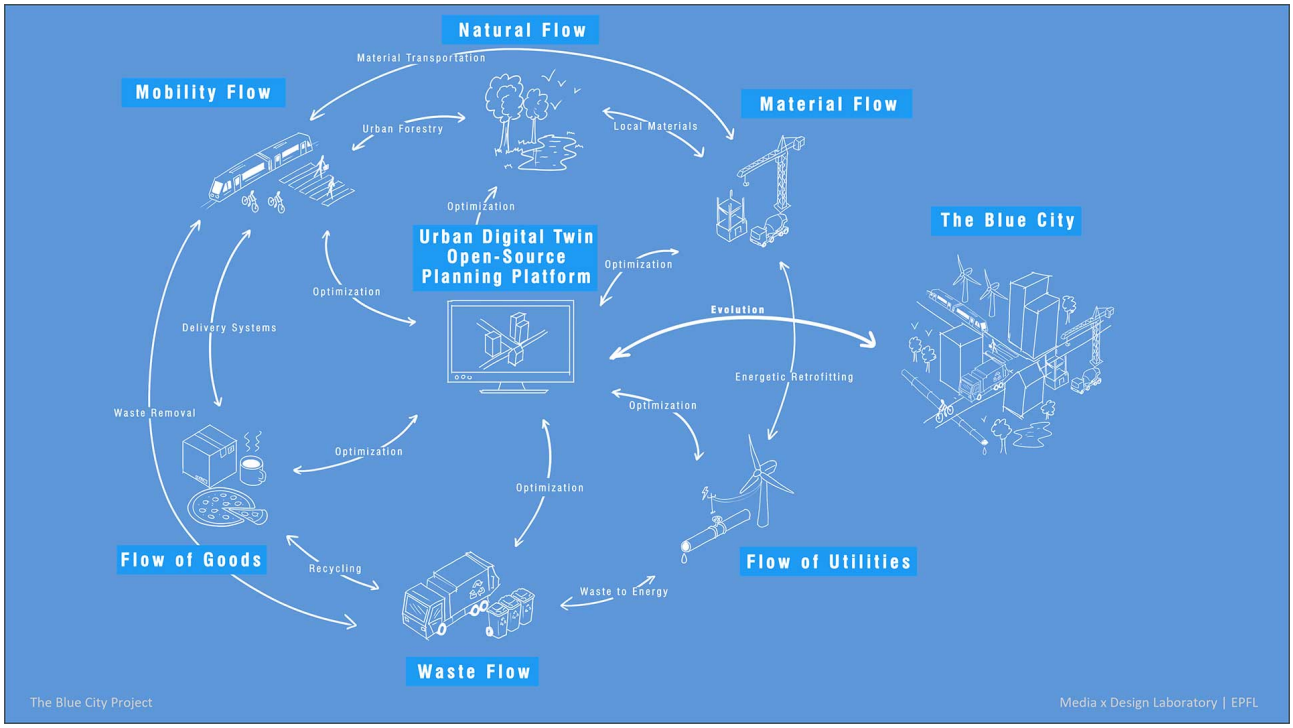


Figure 3: Example of City Actors involved in Urban Digital Twin [15]

2.2.2 UDT Challenges

UDTs face not only the challenges associated with DTs described in section 2.1.2, but also additional complexities resulting from the interconnected nature of the network. In [2] the author highlights a number of these new challenges, which we summarize below:

- **Data Management**: The data generated by the various City Actors present in a UDT will be both vast and of different formats, hence the ability to manage and process city data is important. To deal with heterogeneous data [16] proposes the use of the ontological approach, which was suggested to improve the semantic interoperability and secure future data expansions. Developing data standardization and data-sharing frameworks is fundamental to enhance data exchange in the UDT environment;

- **Physical-to-Virtual Communications**: To ensure accurate and reliable system performance, robust and reliable communication channels between the DT and the PT are required. The author suggests using Machine Learning (ML) algorithms to anticipate instructions from the DTs, thereby reducing the impact of communication related issues. Several factors could increase the overall latency in the system, including sensor latency, data processing latency, network latency, and feedback latency. Some applications may require ultra-low latency communication. A combination of edge and cloud computing will allow smaller amounts of data to be transmitted to the DT, better networks technology will decrease network latency. Finally, the communication scheme must be reliable against power outages or software deployment errors;

- **Virtual-to-Virtual communications**: Communication between DTs, have different requirements compared to P2V channels. Data integration from different sensors is vital. To increase the usability of the shared data, it should be treated as a product by the data-creators, and sharing it must be seen as a service [17]. The channel must be high fault tolerant, to prevent small errors causing large impact in the overall model;

- **Physical data processing**: Accurate modeling in UDT requires reduction and expansion of data dimension. DTs depend on constant data being streamed from the PT. However, data might not arrive due to the various factors mentioned in this very section. DTs should be designed to continue functioning correctly even if such an event occurs. Implementation of data visualization tools will be challenging due to the large-scale networks that generate data at an enormously large rate;

- **Digital Twin modeling**: Modeling and simulation in DTN can achieve high-fidelity virtual models of the physical world. Nonetheless, several factors may hinder the overall fidelity of the system. These include, lack of standardization framework to accommodate different generation standards, currently, no framework that can

meet the demands of an entire UDT exists. High-precision modeling, traditional DT modeling won't be able to handle the multidimensional and advance modeling required for an UDT, and the need for continuous model updates, which rely on accurate data, and sufficient computing power;

- **Computing**: Real-Time data processing in DTs is enabled by edge and cloud computing. UDT requires intensive computing, although cloud and edge computing will help, there is a need for standard architecture to support edge computing in UDT operation. Resource limitations, such as storage and computing power, pose challenges that could be addressed by implementing a migration task strategy to reduce stress on edge nodes. Environmental dynamics may disrupt communication between PT and DT, making it essential to optimize computing resource allocation.

2.3 Research Focus

While the challenge identified in section 2.2.2 were numerous and complex, this thesis will focus on a more confined problem area. Our primary objective is to propose a standard framework that facilitates secure data sharing amongst DTs of various City Actors in a decentralized world. Addressing key considerations such as data integrity, authentication, confidentiality, as well as constraints related to storage, computing power and network reliability.

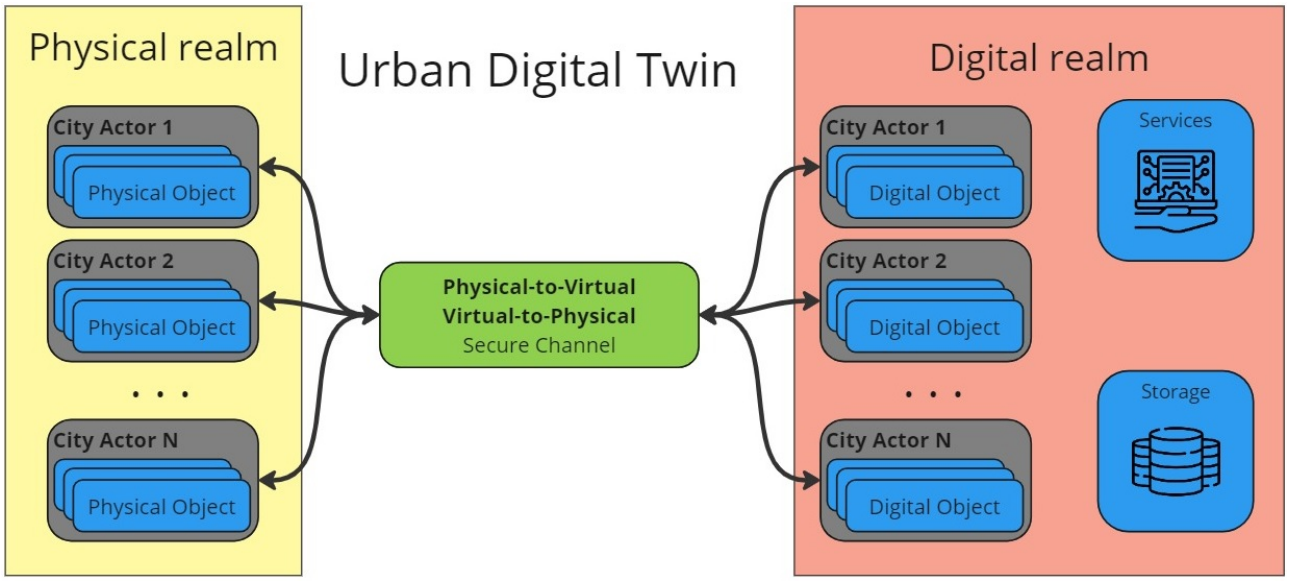


Figure 4: Research Focus: Development of Bi-directional Secure Channel

To accomplish this, we will integrate various existing technologies, to build a coherent ecosystem within UDT. The subsequent sections will introduce and elaborate on these technologies and their roles within our proposed framework.

3 Theoretical Foundations

In this next section, we will first establish the concept of **Self-Sovereign Identity (SSI)** and its relevance to our framework, leading us to explore the principles of **Distributed Ledger Technology (DLT)** more specifically **Blockchain (BC)**, and **Smart Contracts (SC)**. Each of these technologies plays a significant role in addressing the specific challenges of secure data sharing in UDT.

3.1 Self-Sovereign Identity (SSI)

SSI is an approach where the user holds the information necessary to identify himself on the internet. But before diving deeper into it, we should talk about the commonly used approaches nowadays such as the use of **Identity Providers (IdP)**, and **Passwordless Authentication**.

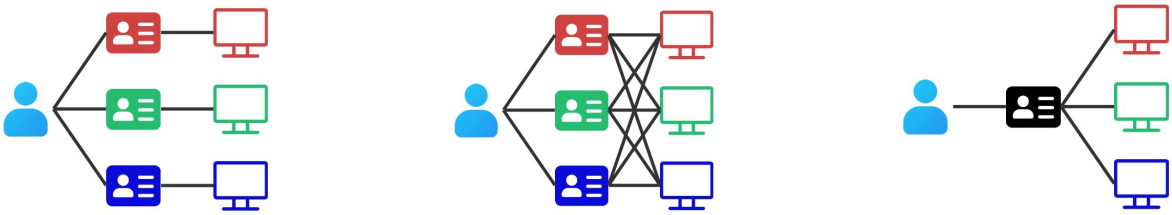
3.1.1 Identity Providers (IdP)

On the internet there is no way of knowing which unique person is browsing the web, however some **Service Providers (SP)** wish to tailor their services differently to each user, for example, once you check Google mail, Google will only show e-mails that were sent specifically to you, and won't show you e-mails sent to other persons. This seems obvious, but to achieve this, Google had to create a barrier of access to their service, where you as a human, are able to authenticate yourself. This is usually achieved via a secrete Username/Password tuple, to prove your identity.

There are three methods of user authentication which were analysed in [18], the first method is known as **Centralized Identity**, where each SP runs his own IdP, meaning for each service, there will be a tuple Username/Password, that are independent from one another. This method can be cumbersome, users either always use the same Username/Password tuple for each IdP, which is not secure, or they use a different password for each IdP, and manage all of them through a third party service like a password manager.
To make this process more manageable for the user, many services synchronized their IdPs, allowing users to identify themselves to a SP through a different IdP, for example, most SP allow authentication through Google Account, meaning that one secure account gives you access to various services, this method is know as **Federated Identity**. Most SP still offer their own IdP, but is up to the user which one he uses.
Recently, a third method has emerged, referred to as **User-Centric Identity**, where multiple SP use the same IdP, however no other IdP exists to access these specific services.

For all of these methods, it is possible to use some form of Two Factor Authentication (2FA), which requires some sort of action in a second, already logged-in, device. For example, when you attempt to connect to your Google account in a new device, it requires you to confirm this action in your phone that is already logged-in.

Each SP is free to choose a different IdP method, this choice is dependant on how detrimental it would be to the user if an unauthorized person were to gain access to their account. For example, a bank will usually use its own IdP that requires a strong password coupled with 2FA at each connection, to protect your bank account.



(a) Centralized identity (b) Federated Identity (c) User-centric Identity

Figure 5: Different Models of Digital Identity

A brief overview of all the three methods is shown in 5 above. Each of these methods is "centric" towards some component in the chain.

- **Centralized identity** is SP centric, since each SP implements a different IdP, making it so the user requires multiple identities;
- **Federated Identity** is IdP centric, because the same IdP allows the same user to connect to various SP, but it is up to the user which IdP he uses;

- **User-centric Identity**, as the name says, is user centric, the same user connects to various SP with an unique identity, which is the only available option.

3.1.2 Passwordless Authentication

As we explore other methods of user authentication, one prominent option is FIDO2, a set of open standards developed collaboratively by the FIDO Alliance and the **World Wide Web Consortium (W3C)**. The FIDO2 framework offers a compelling alternative to traditional password systems, leveraging public key cryptography to deliver a more secure and seamless authentication experience.

FIDO2 consists of two integral components: **Web Authentication (WebAuth)** and **Client-to-Authenticator Protocol (CTAP)**. WebAuth is a web standard that facilitates passwordless authentication on the web. It provides a robust API enabling web applications to authenticate users in a manner that is both secure and resistant to phishing. CTAP, on the other hand, establishes a standard that allows external devices like mobile phones or security keys to function as authenticators through the WebAuth framework. FIDO2 can simplify the user authentication process, with users only needing to perform a straightforward gesture, such as a fingerprint scan or facial recognition, to authenticate themselves.

However, the adoption of FIDO2 is not without its challenges. User familiarity with the system, comprehensibility of the technology, and consistent support across platforms and websites are crucial factors that determine the success of this promising technology [19].

3.1.3 Introduction to Self-Sovereign Identity (SSI)

SSI's goal is to give the power of authentication to the user, by achieving self authentication we would be able to remove the IdP all together. This makes it so the user doesn't have to manage a large amount of identities to connect to the various desired SPs, doesn't need to disclose private information to third parties, and doesn't rely on the availability of an IdP to gain access to a SP. To achieve this, we can work from the previously seen **User-Centric Identity** in figure 5, but with direct user **sovereignty** instead of delegating this task to an IdP. Meaning that the user is fully responsible for all his identities and is able to authenticate himself to a SP without the use of an IdP. This method is illustrated in the image bellow:

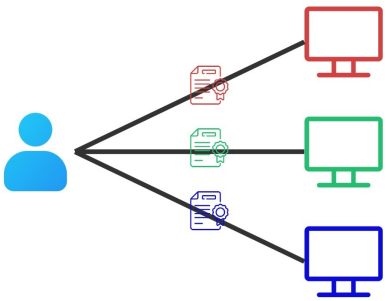


Figure 6: Self-Sovereign Identity

In the following sections, drawing on the work of [20], we will explore the fundamental architecture and key components that empower SSI. These include **Decentralized Networks**, **Decentralized Identifiers (DIDs)**, **Verifiable Credentials (VCs)**, the **Trust Model** and the **Verifiable Data and Status Registry (VDSR)**. Each of these elements plays a crucial role in enabling the user to manage, authenticate, and maintain control over their identity in a decentralized system.

3.1.4 Decentralized Networks

Decentralized networks are systems where components communicate with each other directly and distribute resources, rather than communicating through a central server. In the context of SSI, decentralized networks are leveraged to enable peer-to-peer identity verification and authentication, improving privacy and control over personal information.

3.1.5 Decentralized Identifiers (DIDs)

A DID is a string capable uniquely identifying a user of some decentralized network, they are fully under the control of the DID subject, or owner of the DID. The same user may manage several DIDs, however, a single DID is always linked to an unique user. Therefore DIDs can be used as a tool for self-authentication in a decentralized network. A DID consists of three parts: the DID prefix, a DID method, and a method-specific string, each part separated by a " : ". The figure bellow illustrates an example of a DID:

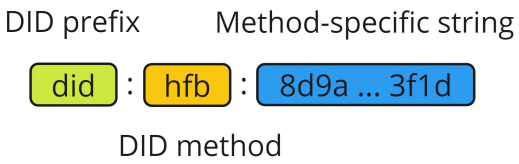


Figure 7: Decentralized Identifier (DID) example

Each DID is associated with a standardized DID Document. It carries information such as the user's Public Key, this information allows the DID subject to prove ownership over the DID, as well as providing information on how to reach the DID subject (ip-address, port, authentication protocol...).

3.1.6 Verifiable Credentials (VCs)

VCs are a set of claims made by an issuer about a subject. They follow a standard format, and are a digital representation of physical credentials, such as drivers licences or university degrees. Credentials are **issued** by issuers to subjects, and can be **presented** and **verified** online. VCs allow users to present them to verifiers in a privacy-preserving way through **Zero-Knowledge Proofs (ZKP)**.
It is important to note that some VCs should be revocable, for example, a drivers licence may have a expiration date. This is a major challenge in the SSI ecosystem.

3.1.7 Trust Model

There are three core roles possible for a SSI user in a decentralized network:

- **Issuer**: usually, they are organizations that issue VCs to other users;
- **Holder**: is the user who holds VCs and presents them to prove, for example, their identity;
- **Verifier**: is a user that verifies the validity of presented VCs, using **Public Key Infrastructure (PKI)** signatures.

It is important to note that the same user may have multiple roles.
An additional Role is the **Governance Authority** who is responsible for ensuring standardization and compatibility between different SSI ecosystems. Although this role is not mandatory for an SSI network, it is crucial to assure its well-functioning.

3.1.8 Verifiable Data and Status Registry (VDSR)

The VDSR is a critical component within the SSI ecosystem. It empowers the **holder** of VCs to control how they are shared, providing the **verifiers** with assurances about the claim's authenticity and integrity, typically achieved via PKI. A VDSR is commonly implemented utilizing **Distributed Ledger Technology (DLT)** with we will discuss in the next section 3.2.

3.2 Distributed Ledger Technology (DLT)

In this section, we dive into the exploration of DLT within the context of SSI. Numerous studies, such as [21, 22], provide comprehensive overviews of the various types of DLT. However, for the purpose of this work, we will mainly focus on a particular type of DLT – the Blockchain.

3.2.1 Blockchain (BC)

Satoshi Nakamoto introduced the BC in 2008, its first application was in the world of finance, namely Bitcoin. He proposed to distribute electronic transactions rather than being dependant on a centralized entity. Since then the BC technology has been applied outside the financial world [23].

A BC is a continuously-growing chain of blocks, where each block represents a single transaction. For each new transaction, a new block is added to the chain, hence the name. Before a new block to be added into the chain, it needs to be validated by a consensus algorithm. The use of such consensus algorithms eliminates the need for **Trusted Third Party (TTP)**, thus removing associated costs [24]. Each block in the chain contains a cryptographic hash of the previous block, a timestamp of when it was created and any transaction information. Cryptographic hashes help ensure the integrity and immutability of the BC, figure 8 helps visualize this concept. The larger the chain is, the more protected it is [24, 25, 26]. BCs can be seen as a public ledgers where no transaction can be falsified.

The consensus algorithm may change depending on which BC platform. Nevertheless, the general idea is that each peer node connected to the BC participates in the validation of each new block, this can be achieved via a voting system amongst the peer nodes. In [27] the author completes a review on various existing consensus algorithms, with some of the most used are **Proof of Work (PoW)** and **Proof of Stake (PoS)**.

There are three types of BCs: **public**, **private** and **federated**. Public BCs allow any anonymous entity to join the BC and become a node with voting power. Whereas Private BC are controlled by a limited number of pre-authorized participants. Federated BCs are a combination of the previous two, they assign a leader that verifies each transaction. The choice on which type to use is context dependant [28].

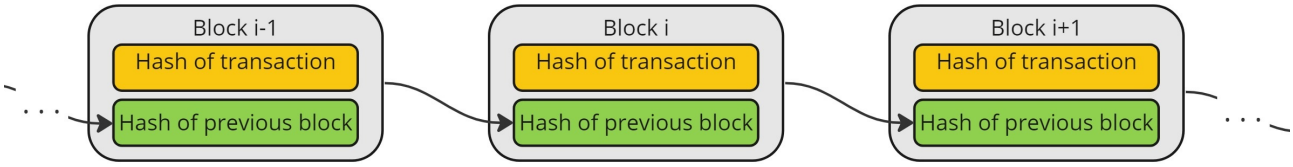


Figure 8: Visualization of simplified blockchain

3.2.2 Blockchain (BC) as Verifiable Data and Status Registry (VDSR)

Thanks to the immutability of the BC serves as an excellent candidate for a VDSR within the SSI ecosystem. A BC can maintain a permanent, verifiable record of all transactions, which in the context of SSI may represent interactions involving VCs, issuance, revocation, presentation or verification. In the following section 3.3 we will analyse how BCs facilitate SSI operations through the use of SCs for automated transaction management.

3.3 Smart Contract (SC)

As we further explore the decentralized nature of SSI and the use of BCs as a VDSR, another powerful tool emerges that enhances the automation and trustless operations within the system: SCs. SCs are decentralized, self-executing agreements coded onto a blockchain. They automate processes according to predefined rules, making transactions traceable, transparent, and irreversible, which aligns well with the principles of SSI. SCs can manage the issuance, verification, revocation and presentation of VCs in an SSI context, effectively decentralizing trust and removing the need for a central authority. In the following section, we will explore the concept of SCs and discuss how they can contribute to the operation and reliability of the SSI ecosystem.

3.3.1 Introduction to Smart Contract (SC)

Smart contracts were first introduced in 1997 by Szabo Nick. In his paper, he argues that the digital revolution has enabled the development of new institutions that can secure relationships in cyberspace. He describes them as a type of protocol that reduce transaction costs and facilitate all phases of the contract development and enforcement. They are capable of formalizing secure digital relationships, making them more functional than their paper-based counterparts [29].

3.3.2 Life Cycle

BC technology enables the use of SC, the execution of each contractual transaction is recorded in the BC that, as we saw in the previous section, is immutable which will be a key factor in ensuring the integrity of SC transactions [24].

To better understand how SC function, we can examine the four stages of SC life cycle identified in [24]:

- **Creation** of SC: The creation process involves collaboration among various experts, and multiple steps. First, parties involved in the contract must agree to the terms, obligations and rights. Secondly an actual physical contract is drafted by actual lawyers. Afterwards, software developers must convert the agreement into a SC using programming languages, declarative languages and logic-based languages are preferred. Finally, the SC must be developed, which includes design, implementation and validation. This process can be quite lengthy and iterative, as multiple rounds of negotiation may occur;
- **Deployment** of SC: Once the SC is thoroughly validated, it can be deployed on top of a BC platform. After a contract is deployed on BC, it cannot be modified, any modifications will require a whole new SC. All parties have access to the SC through the BC. Moreover, digital assets of both parties are locked by freezing the corresponding digital wallets. Parties involved in the SC can be identified by their digital wallets;
- **Execution** of SC: After deployment, contractual clauses are monitored. Once all pre-conditions for execution are met, the contractual transactions are automatically executed. Each transaction that is executed is validated by the various nodes in the BC and will be committed to the BC as a new block;
- **Completion** of SC: Once execution of the contract is done, involved parties states are update in the BC. Digital assets are transferred and digital wallets are unfrozen.

These four stages complete the life cycle of SC, illustrating the process from creation to completion. It is worth noting that at all stages but the first, the SC writes to the BC. Challenges associated with each stage will be discussed in section 3.3.4.

3.3.3 Applications

SCs have a broad field of applications, in [24] the author categorizes six different types of SC applications, we will go over them and defend the choices with various examples:

- **Internet of Things (IoT)**: Integration of IoT can support various applications such as supply chain management, inventory control systems, access control. For example, devices may obtain hashes from SC for swifter firmware update process, this will reduce the cost for maintaining central servers, in addition it enables automate **Peer-to-Peer (P2P)** business trading, which lowers TTP fees, and speed up the overall process [30];
- **Distributed system security**: **Distributed Denial-of-Service (DDoS)** attacks are often used to great success, these attacks try to overload systems to try and interrupt the services proposed by the system, in [31] a SC-based solution for DDoS attacks was proposed, the main idea was to use the BC to track attackers IP addresses. Cloud computing is a service where the provider sells access to computing and storage power, to verify the trustfulness of a **Cloud Service Provider (CSP)**, Dong et al. [32] proposes a solution based on SC;
- **Finance**: SC can reduce financial risks, cut down on administration costs. In commercial and retail banking SC can bring benefits to the mortgage loan industry through automation of the mortgage process as shown in [33]. In the world of insurance, AXA has implemented solutions based on SC, their idea was to automate flight insurance reimbursements, when any passenger decides to buy a plane ticket with AXA insurance, a SC will automatically be signed and launched to the BC, this contract connects to the global air traffic database, if there is a flight delay of over two hours, the passenger will immediately be reimbursed via the SC [34], this improves efficiency, and reduces claim process costs;
- **Data provenance**: The ability of ensuring the authenticity of data is very important, some solutions propose to store meta-data records, current implementation from Trusted Platform Module assures data provenance at the cost of privacy [35]. SC might just be the solution to the problem of data provenance while preserving user privacy as proposed by Liang et al. in [36];
- **Sharing economy**: Represents the idea of reducing costs for consumers, by putting in place a platform for underutilized resource sharing such as cars, homes, objects. However, most sharing economy platforms are suffering from high transaction costs, privacy exposure and unreliable TTP. SC can revolutionize these platforms, by reducing transaction costs, enforcing agreed upon transactions, protecting user privacy and assuring trust between the parties without TTP, as shown by Bogner et al. in [37];

- **Public sector**: The use of BC can essentially prevent data fraudulence and provide transparency. Currently e-voting faces challenges in identity verification as well as preservation of voter anonymity. SC offers the solution for e-voting as shown by McCorry et al. in [38]. SC can also establish personal digital identity, users are able to protect their private information via SC that grant access permission to other users [39].

These examples showcase the potential of SC across various sectors. Their design and implementation are made possible by underlying platforms tailored to support their execution. In the following section we will go through the major platforms that enable SC, their characteristics, and their influence in the adoption of this technology.

3.3.4 Smart Contract (SC) Challenges

While the notion of SC may be applied in multiple fields, they all share the same challenges. The challenges associated with SCs can be split between the various life cycle stages defined in section 3.3.2, starting from their creation, deployment to their execution and eventual completion. In each phase, unique issues related to readability, functional complexities, deployment intricacies, execution efficiency, privacy and security concerns emerge. This section seeks to explore these challenges:

- **Creation Challenges**: This phase is associated with lack of readability and functional issues. The source code of SCs is typically written in programming languages like Solidity, Go, Kotlin, and Java, is compiled and executed in various forms. Ensuring readability across these forms is a significant challenge. Alongside lack of readability, functional issues such as re-entrancy, block randomness, and overcharging due to under-optimization pose substantial problems to overcome;

- **Deployment Challenges**: Once the deployment stage is reached, it becomes nearly impossible to revise a SC. Therefore, ensuring the correctness of the contract prior to deployment is critical, some SC platforms enable versions in SC, allowing them to be upgraded. Alongside correctness, managing the dynamic control flow of SCs, particularly during interactions with other SCs, requires meticulous design and raises difficulties in predicting the behavior of a contract;

- **Execution Challenges**: They may require real-world data necessitating the role of an oracle, establishing a trustworthy oracle is crucial. However, ensuring the reliability of data from the oracle is complex. Adding to this, the bifurcation in blockchain branches can randomize operation execution order, leading to potential incorrect outcomes. Lastly, execution efficiency is hampered as SCs are executed serially by miners, limiting their performance;

- **Completion Challenges**: As SC reach completion, they encounter privacy and security-related challenges. Existing SC and BC technologies often lack robust privacy-preserving mechanisms, resulting in potential information leaks through transaction analysis. In addition, security is compromised due to the susceptibility of BC to malicious attacks, causing delays in message broadcasting. In the era of rampant digital fraud, the ability to detect scams in the realm of SC is essential.

3.3.5 Platforms

The choice of the platform is crucial. It should be driven by the functionalities required for the whole framework, while having its advantages and disadvantages in mind.
However, given the broad range of existing platforms and the specific requirements of our proposed framework, an exhaustive review and comparison of all available platforms falls beyond the scope of this thesis. We will instead focus on key requirements when choosing a platform. In [24, 40] the authors complete extensive comparisons between the various available platforms.

While we do not intend to perform a comprehensive review of all available platforms, it is worth mentioning a few that stand out. Namely **Hyperledger Indy (HLI)** and **Hyperledger Fabric Blockchain (HFB)**.

- **HLI** is a distributed ledger built specifically for SSI. It provides tools, libraries, and reusable components for creating and using independent digital identities. HLI's main objective is to provide digital identities that are interoperable across administrative domains, applications, and any other silo which can lead to more trustworthy transactions;

- **HFB** is an open-source project intended as a foundation for developing applications or solutions with a modular architecture. Fabric allows components, such as consensus and membership services, to be plug-and-play, making it a versatile solution for various business scenarios.

HLI seems perfect at first, due to its native SSI integration. Unfortunately, it lacks the ability to deploy general purpose SCs. Whereas we can develop SCs that implement SSI and deploy them in **HFB**. This has been successfully attempted in [41, 42, 43].

In section 4, we do an in depth analysis of HFB architecture and its core components.

4 Hyperledger Fabric Blockchain (HFB)

In this section, we will provide an in-depth analysis of the default components required for the use of HFB, we will analyse the different types of organizations, types of peer nodes, endorsement policies, the flow of interactions between each component and finally some specifications. This analysis is based on the official Whitepaper for Hyperledger Fabbric Blockchain [44, 45], as well as other resources [46, 47].

4.1 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 responsibilities 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 a subset of the organizations in the channel, and the consensus would be achieve via RAFT or KAFKA protocols.
In figure 9 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 SCs in the network as well as endorsing transactions. In figure 9 organizations 1 and 2 manage various types of peer nodes.

Both organizations in figure 9 have a CA that is issuing a x.509 based on 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.

4.2 Types of Peer Nodes

- **Endorsing peer**: is responsible for simulating and endorsing transactions by executing SC, 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 SC.

4.3 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. Each SC may have a different Endorsement policy.

4.4 Interactions

In this section we will analyse the interactions between each type of nodes. Each item in this list refers to a specific arrow label in the figure 9.

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

(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.

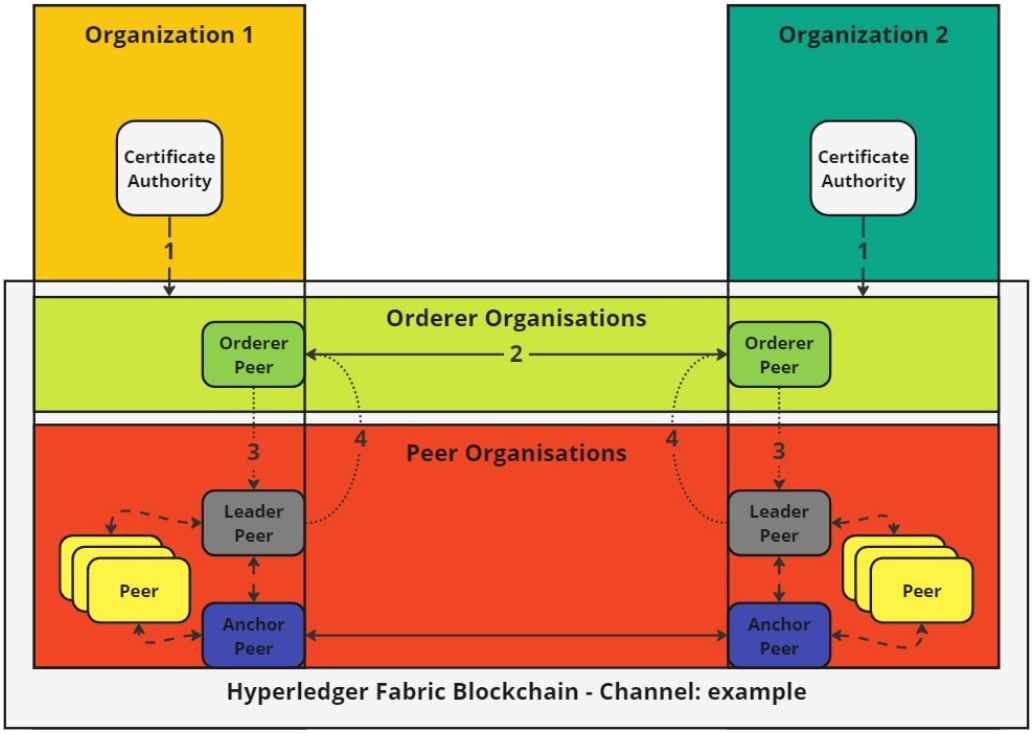


Figure 9: Hyperledger Fabric Blockchain general architecture between two organizations

4.5 Specifications

4.5.1 Type of Blockchain

In section 3.2.1, we introduced different types of BCs. Let's focus on the first two, **Public** and **Private** BCs.

Public BCs allow every entity to join, this can be very useful to ensure trust ans transparency to the general public, the downside of such BCs is that every new node can be an ill intended node, to overcome this challenge they are forced to use more robust consensus algorithms.

Private BCs allow only predefined entities to become a Peer Node in the network, it makes for a better overall performance, scalability and a finer Access control, by blocking the general public we are giving up Auditability. This allows us to employ simpler consensus mechanisms such as Practical Byzantine Fault Tolerance (PBFT) or Raft, which can offer better performance and scalability.

Ultimately, each approach comes with its own set of trade-offs. The choice on which type of BC we use should be Use Case dependant. HFB is a **Private** BC.

In the upcoming section, we will explore specific use cases for our framework, shedding light on why HFB, a **Private** BC, is the preferred choice. The use cases will demonstrate how we prioritize performance, scalability, and access control while still maintaining a necessary level of transparency and security.

4.5.2 Consensus algorithm

5 Framework Definitions, Use Cases, and Requirements

To provide a comprehensive understanding of the proposed system framework, this section introduces specific terminologies used in this document in section 5.1, outlines potential scenarios where the framework could be employed in section 5.2, and enumerates the key requirements necessary for the platform to function efficiently and securely in section 5.3.

5.1 Framework Definitions

5.1.1 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.

5.1.2 Asset

An asset represents a physical object or a service, that is owned by a specific city actor as illustrated in figure 10. The term Digital Twin (DT) is intentionally avoided, as it does not have an official definition. Using Asset as the primary term avoids potential confusion and misinterpretation associated with the term DT.

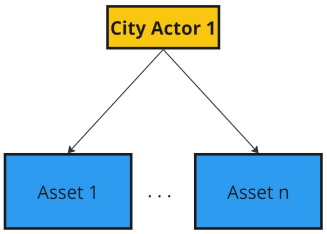


Figure 10: City Actor with n assets

5.1.3 Perimeter

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

5.1.4 Channel

In the context of our proposed framework utilizing Hyperledger Fabric Blockchain (HFB), a 'Channel' can be viewed as a private blockchain overlay within the larger HFB network. Each channel is an independent chain of transaction blocks containing only transactions specific to the entities participating in that channel. This means that transactions conducted within a channel are visible only to its participants, providing a layer of privacy and confidentiality.

5.1.5 Topic

In the Apache Kafka infrastructure, a topic can be seen as a private broadcast channel, where authorized assets can produce messages and subscribe to the topic in order to receive messages from others. Each topic is associated with a specific HFB channel.

5.2 Use Cases

To understand the thought process we first must understand the needs of the users (organizations).
The Use Case is based on a Swiss company, droople. The company's mission is to develop smart tools to improve the way people use, conserve and value water [48].

They developed smart taps, which can be installed at each water terminal point in a specific household. These smart taps are capable of collecting real-time water consumption data. By combining all the data of a specific household, one can perform computations on the data to calculate average usage, identify anomalies or even predict future demand. Unfortunately, extracting the full potential of this data involves collaborations with other organizations that own complementary datasets. These could include entities like utility companies, weather forecast organizations, local government bodies, or environmental conservation groups.

The definition of the Use Cases will be based on the IBM official template [49].

5.2.1 Use Case 1

Use-case field	Description
Use case name	Decentralized Data Sharing Framework
Subject area	Smart City Water Management
Business event	Real-time water consumption data collected from households using smart taps
Actors	Households (Data providers), City utilities department/Government bodies (Data consumers), and droople (Data collection tool providers)
Use case overview	Establishment of a secure, decentralized platform for various city actors to share and receive real-time water consumption data from households with smart taps
Preconditions	Households must have droople's smart taps installed and operational. Agreement from all participating entities to use the proposed decentralized data sharing platform
Termination outcome	Successful establishment and utilization of a decentralized platform for secure data sharing
Condition affecting termination outcome	Successful termination is dependent on the implementation and acceptance of the decentralized data sharing platform by all relevant parties
Use case description	Households with smart taps provide real-time water consumption data. The city utilities department and other participating entities access this data via the decentralized platform, maintaining the security and privacy of the data at all times
Input summary	Real-time water consumption data from households
Output summary	Access to secure, real-time water consumption data for various city actors
Usability index	This will be based on stakeholders' evaluation but could be quite high due to its importance for secure and decentralized data sharing
Use case notes	The system should ensure that all shared data is encrypted and user privacy is respected. The platform should also be scalable to handle increasing data as more households install smart taps

Table 1: Decentralized Data Sharing Framework Use Case 1

Figure 11 below illustrates an example scenario of Use Case 1, demonstrating the interactions between the household and the relevant government body.

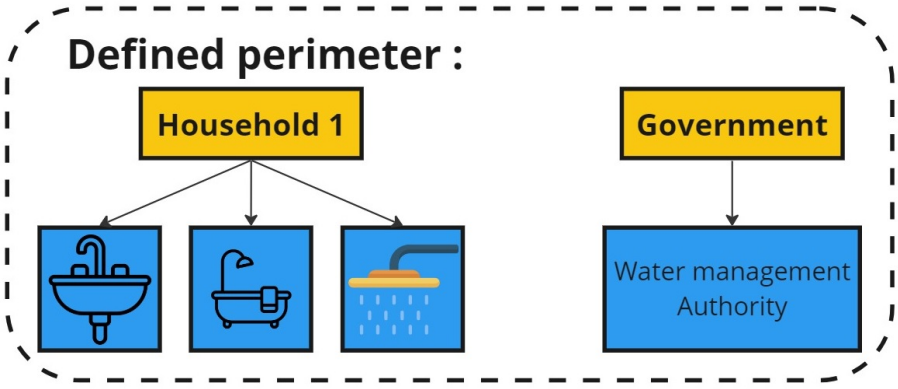


Figure 11: Use case 1

5.2.2 Use Case 2

Expanding on the foundations established in Use Case 1, we can introduce additional functionalities, leading to the following evolved specifications:

Use-case field	Description
Use case name	Secure Expansion of Data Sharing Framework
Use case overview	A new organization joins the decentralized platform for data sharing, with permissions set to only access future communications while ensuring the privacy and security of past communications
Actors	Addition of a New Organization (Future Data Consumer) to the existing actors: Households (Data providers), City utilities department/-Government bodies (Data consumers), and droople (Data collection tool providers)
Preconditions	All existing conditions from Use Case 1 plus the new organization must agree to the terms of the decentralized platform, including restricted access to past data
Use case description	A new organization joins the decentralized platform for data sharing. This organization can only access future water consumption data shared by households. All past communications remain secure and inaccessible to the new organization

Table 2: Decentralized Data Sharing Framework Use Case 2

Figure 12 below illustrates an example scenario of Use Case 2, demonstrating an additional actor in the defined perimeter.

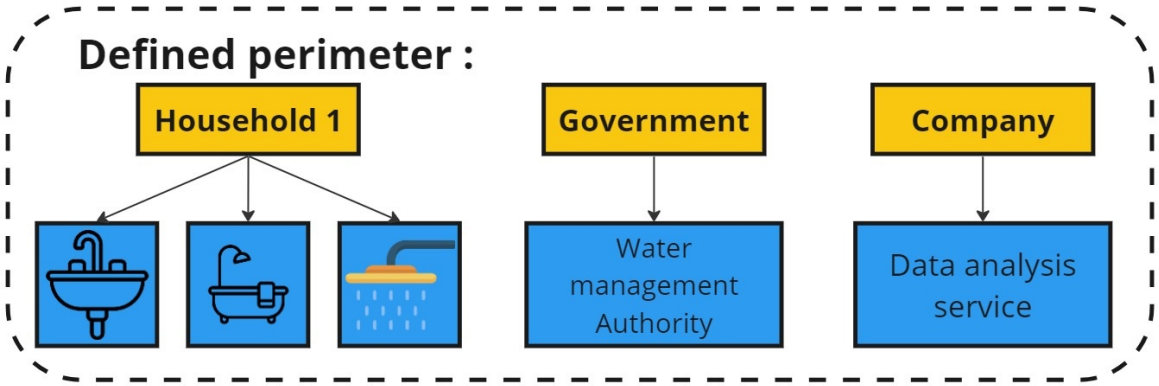


Figure 12: Use case 2

5.2.3 Use Case 3

Building on the advancements achieved in Use Case 2, we can further enhance the system's capabilities, resulting in the subsequent specifications:

Use-case field	Description
Use case name	Implementation of Smart Contracts for Traceability and Automated Payments
Use case overview	Organizations in the decentralized platform implement smart contracts to provide traceability for business transactions and automate payments without the need for additional Trusted Third Parties (TTP)
Actors	Existing actors: Households (Data providers), City utilities department/Government bodies, New Organization (Data consumers), and droople (Data collection tool providers) plus Payment Processors (Facilitated by Smart Contracts)
Preconditions	All existing conditions from Use Case 2 plus the organizations must agree to the terms of the smart contracts for transaction traceability and automated payments
Use case description	Organizations on the decentralized platform implement smart contracts. This provides traceability for business transactions and facilitates automated payments, eliminating the need for additional TTPs

Table 3: Decentralized Data Sharing Framework Use Case 3

5.3 Requirements

In [50] the author breaks down the general UDT architecture into six layers: **Business** and **Functional** layers, owned by the federation, as well as **Information**, **Data Processing**, **Communication** and **Integration** layers, owned the an independent City Actors. The figure below illustrates these 6 layers:

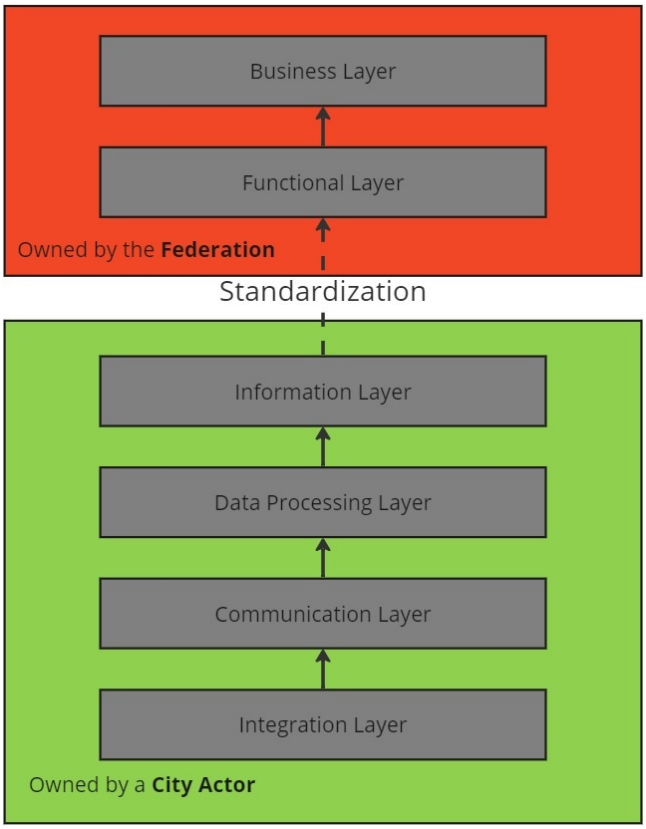


Figure 13: 6 Layer based Urban Digital Twin

The general idea is that each City Actor owns their Assets that are data generators, the data needs to be contextualized and standardized before going through the **Common Secure Channel**, where it will reach the Federated layers. The Federated layers are composed of digital services that consume the data generated by the Assets.

The data pipelines present in the layers owned by the City Actors have evolved independently from each other, which forms a heterogeneous system. Since the Federated layers will consume data from different City Actors, an additional abstract layer of **Standardization** is required. It is in this abstract layer that our **Physical-to-Virtual secured channel**, described in section 2.3, will sit.

The **Standardization** layer has the goal of rendering independent City Actor data pipelines interoperable. To that end we will create a framework, the **Physical-to-Virtual secured channel**, that must respect certain requirements related to the Layers around the **Standardization** Layer. These requirements concern the **Business**, **Functional** and **Information** Layers. In [50] the author defines an extensive list of requirements related to these three layers as well as **Non-Functional** requirements, the latter ones do not fall under any of the 6 layers illustrated in figure 13.

Respecting these requirements will allow us to create a capable and trusted platform for data sharing amongst City Actors. In total there are 35 individual and very granular requirements, at this stage we choose to keep a higher level definition and more global definition of these requirements, which are as follows:

(1) **Availability**: The platform must always be online;

(2) **Resilience**: 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;

(7) **Scalability**: The platform must be capable of handling the increasing load as more organizations join and the data shared grows in volume.

6 Architecture

In this section we will propose a system framework where different organizations within an urban context may connect themselves and share data securely. The framework will be built around the idea of *decentralization*.
In [50] the author tackles the same problem, however the proposed framework was built around a central entity of trust, which he calls the federation.
Decentralization over centralization allows for heightened privacy and security, enhances resilience, and diminishes the reliance on a single central authority, therefore resulting in a more robust trust environment.
Having in mind all technologies that were introduced in sections 3 and 4, the use cases as well as the requirements seen in section 5, we will build the framework step by step.

6.1 Message Broker

Our choice of communication broker aims to solve the following requirements: **Availability**, **Resilience**, **Efficiency** and **Scalability**.

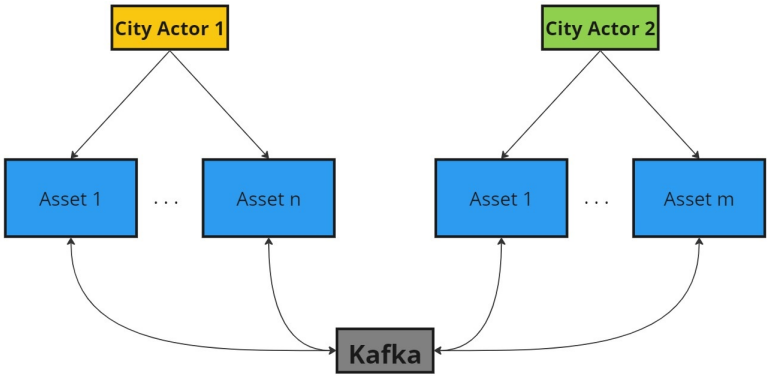


Figure 14: Communication platform

Due to Kafka's decentralized architecture, it is constantly available, and can easily scale to accommodate the needs of various organizations [51].
If an asset were to lose connection momentarily, it would still be able to consume missed messages, since they are stored in the Kafka topic.
Kafka also enables group communication, every entity can be both a producer as well as a consumer.

However, if an attacker were to gain unauthorized access to the topic he would be able to read every message, breaking the **Confidentiality** requirement. 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 the **Availability** requirement.

6.2 Authentication

The proposed platform will be based on SSI and HFB, which were described in sections 3.1 and 4. HFB doesn't natively support SSI integration, while there are projects such as HLI that support SSI, they are quite limited for Use Cases that need more functionalities such as transaction traceability, as described in section 5.2.3. To overcome this limitation, we will use SCs that will implement SSI in HFB.
Each Asset will be represented as an unique Peer Node in the HFB, each Peer Node will be issued a x.509 certificate by its organizations CA, they are used for maintaining HFB functionalities such as gossip protocol. This certificate carries a Public Key, to which the Peer Node owns the respective Private Key. Each organization manages a CA, meaning only organizations can issue certificates for their Peer Nodes.

Unfortunately, since it is built on PKI, Peer Nodes would still require a TTP to authenticate each other. This is where SSI comes in, we propose to integrate a DID into HFB x.509 certificates. The DID would simply be the cryptographic hash of the Peer Node's Public Key, this way we are certain that it will be unique across the various organizations. This same protocol has been suggested in [41], where the authors attempt to solve security issues related to Modbus Industrial Internet of Things (IIoT) protocol. Each Peer Node owns a x.509 certificate with the add-on DID. Each DID is then registered to the BC. Due to the assured synchronicity of the BC, each Peer Node keeps a local copy of its view of the

BC. Which means every Peer Node knows every DID registered in the BC, and can used it as a VDSR, this ultimately allows Peer Nodes to self authenticate, as intended.

This adds a new requirement, Kafka's access proxy must be able to see the ledger to verify identities. Figure 15 bellow shows the whole process:

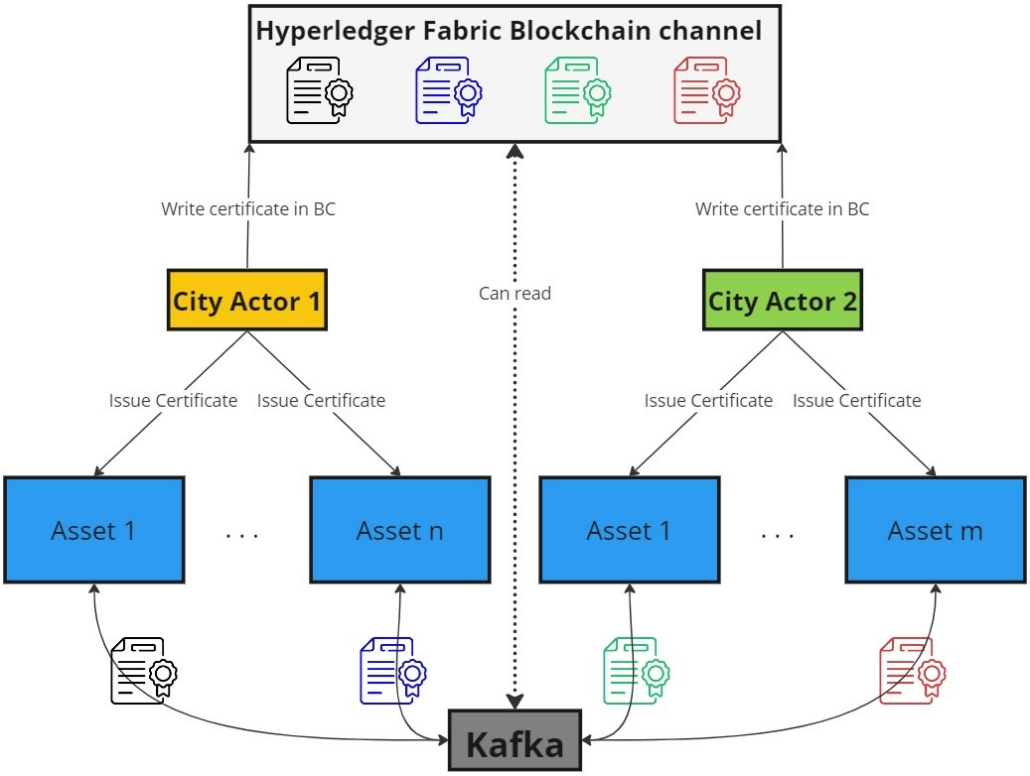


Figure 15: Communication channel with self authentication

Moreover, to integrate the scheme presented in figured 15 into the general architecture shown in figure 9. We will represent assets as peer nodes in HFB, but for the purpose of this document, they are the same thing, they will represent Peer nodes of various types, which were introduced in section 4.2:

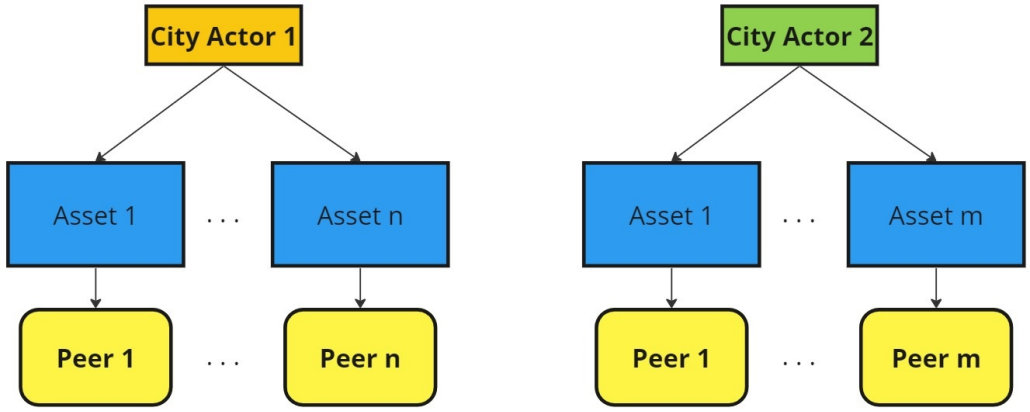


Figure 16: Assets transformation to Peer Nodes

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

6.3 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. It won't be able to have authority over the various organizations, it will only offer a platform, through which various organizations in the Urban context can share information. With Figure 9 in mind, we propose the following architecture with the addition of the Federation, shown in figure 18.
For simplicity some notations were modified:

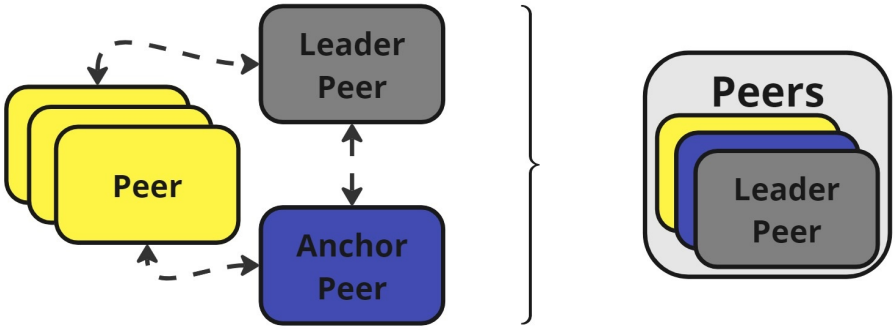


Figure 17: New Peer Nodes notation

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

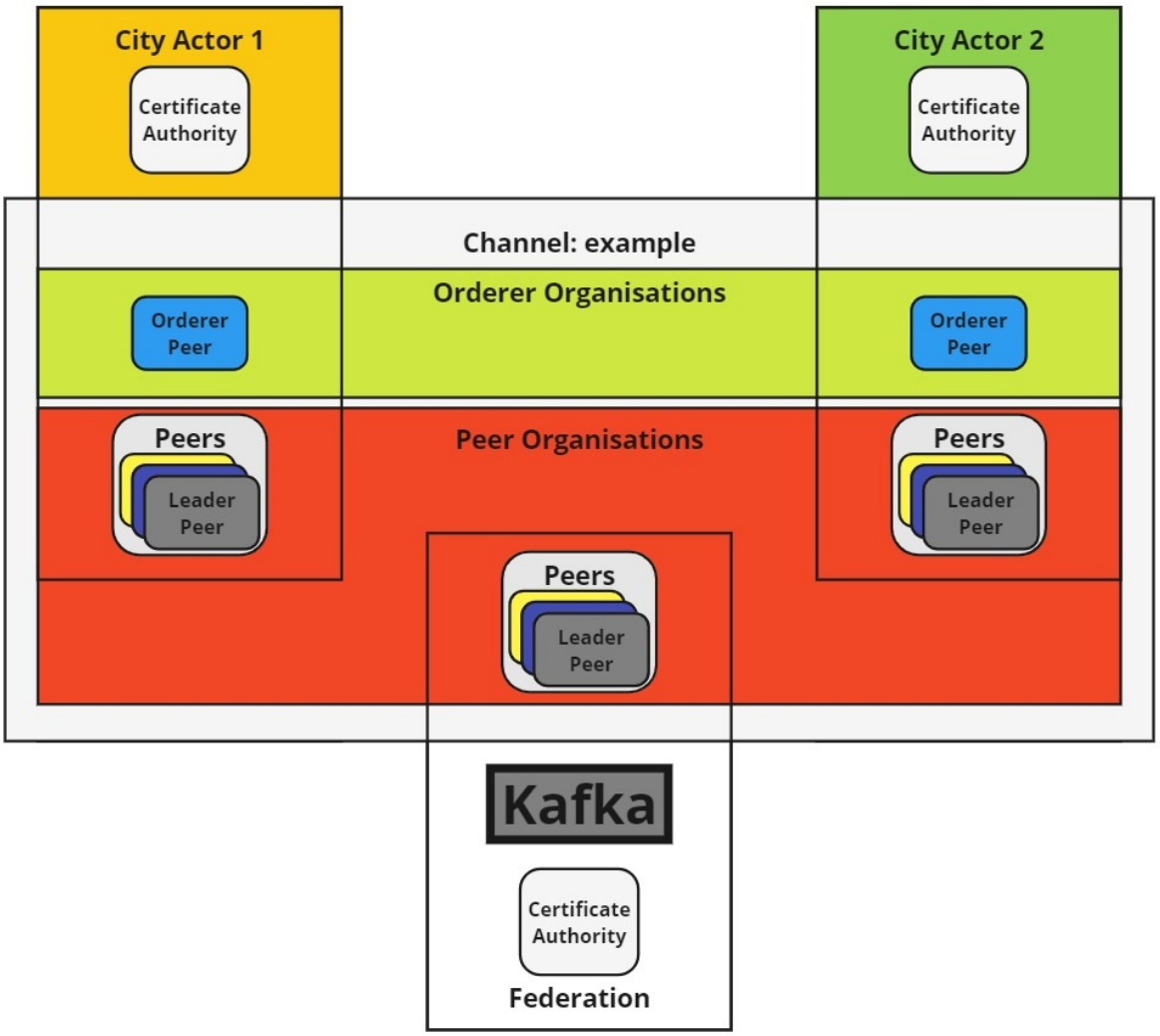


Figure 18: Hyperledger Fabric Architecture with Federation

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

6.4 Use Case 1: Confidentiality

In this section we will go through the process of assuring **Confidentiality**, a requirement announced in section 5.3.

6.4.1 Setting

As described in 6.2, DIDs 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 5.2.1, using the provided example. Figure 19 illustrates how an asset, its identity, and its public key are visualized:

Figure 19: Left: Asset A - Middle: Asset A's Identity registered - Right: Asset A's Public Key

City actor 1, a household owns 3 assets (smart taps), called A, B and C. City actor 2, a water management company, owns 1 asset called D. As shown in the figure bellow:

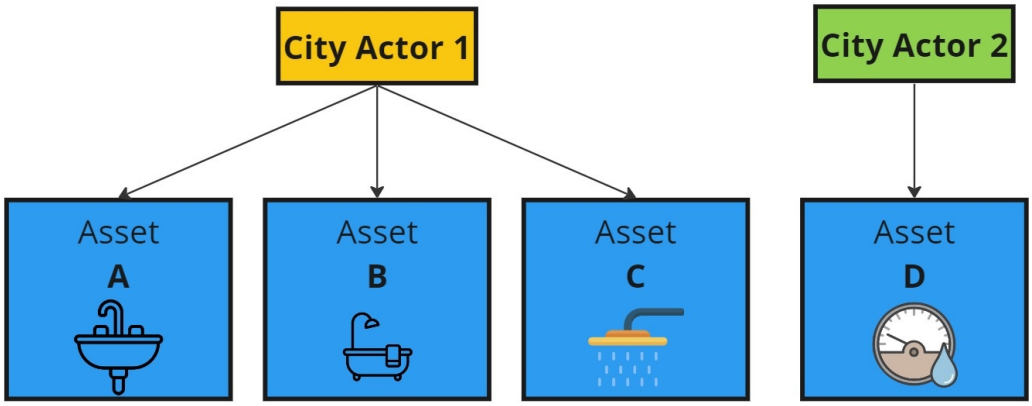


Figure 20: Two organizations with a total of 4 assets A,B,C and D

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

6.4.2 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 its organization's CA, with the addition of an unique DID. Following Certificate issuance, its DID must be registered in the HFB, as depicted in the figure bellow:

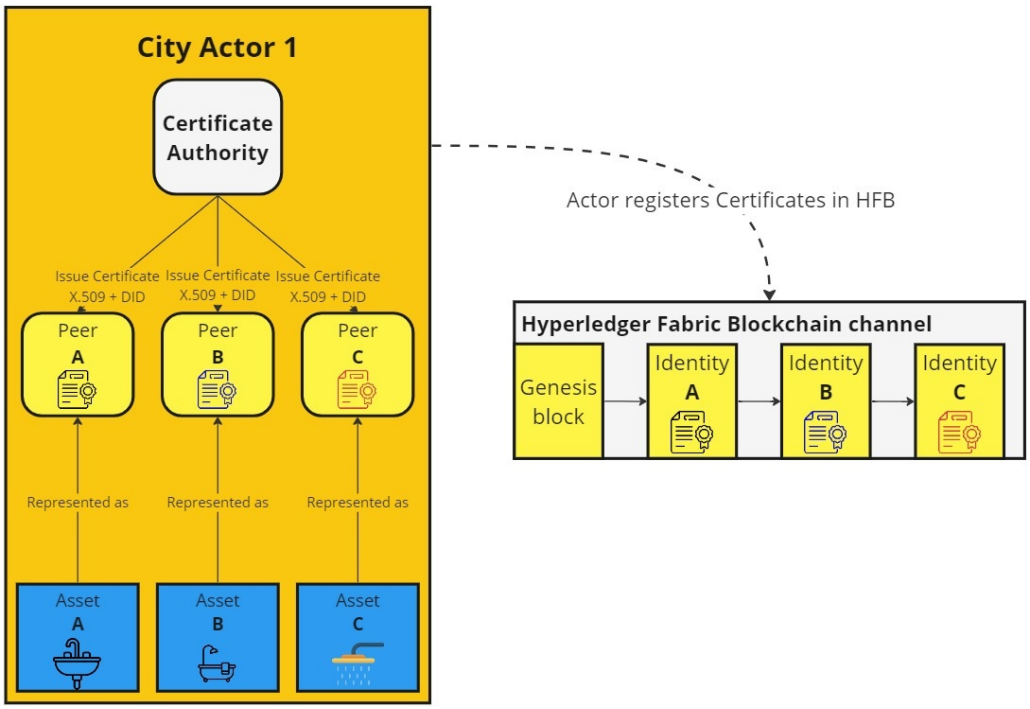


Figure 21: Scenario: Certificate issuance and DID registration in Hyperledger Fabric Blockchain

6.4.3 Common secrete key computation

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

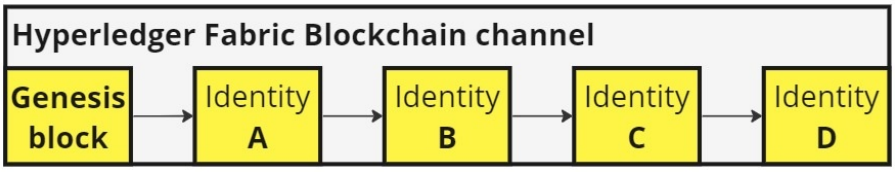


Figure 22: Blockchain status after DID registration

We want to use Kafka as a secure group communication channel. To uphold the **confidentiality** requirement, only assets registered in the HFB should be able encrypt and decrypt messages sent through the kafka broker. Every asset in the defined perimeter will send its generated data through the same kafka topic. There are two options to assure **confidentiality**, each possible pair of assets set up symmetric keys amongst themselves, for a total of $(m-1)!$ keys, and each message is sent $m-1$ times where m is the number of assets registered in the channel, or all assets set up an unique symmetric key, that will be known by every registered asset. The later is the best option for our intended use.

In this section we will analyse a protocol proposed in [52] that allows for secure communication for a dynamic group in which members are located in a distributed fashion and can join and leave the group/channel at any time.

This protocol blends **Binary Key Trees (BKTs)** with **Diffie-Hellman (DH)** key exchange protocol. A BKT is a tree data structure where each Node has either two children, referred to as the left child and right child, or none, known as a leaf Node, this means that it is impossible for a Node to have only one child. The top most Node is known as the root. The figure below shows a generic binary tree of depth 2 with 7 total Nodes, from which 4 are leafs (yellow), 2 intermediary Nodes (grey), and the root (green):

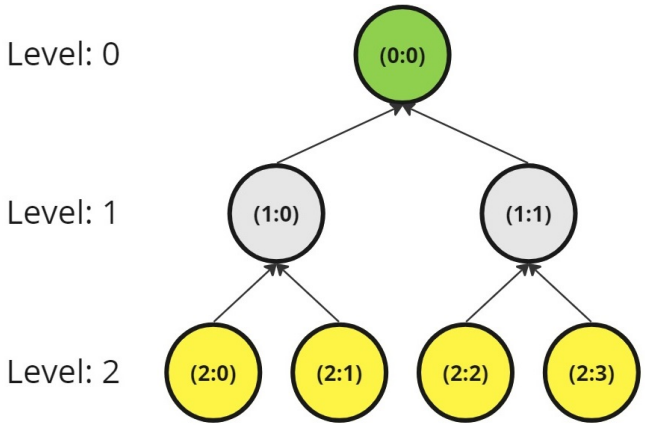


Figure 23: Generic binary tree of depth = 2

Nodes are denoted $< l : v >$, where $0 \leq v \leq 2^l - 1$. Each Node is associated with a secrete Key: $K_{<l:v>}$, and the Blinded Key (BK) which is public: $BK_{<l:v>} = f(K_{<l:v>})$ where function $f(\cdot)$ computes the $BK_{<l:v>}$ of the corresponding $K_{<l:v>}$. To compute the private key K associated with Node $< l : v >$, we need a combination of its children key pairs as follows:

$$\begin{aligned} K_{<l:v>} &= g(K_{<l+1:2v+1>}, BK_{<l+1:2v>}) \\ &= g(K_{<l+1:2v>}, BK_{<l+1:2v+1>}) \end{aligned}$$

Where $g(\cdot,\cdot)$ is the function used by Nodes $< l+1 : 2v+1 >$ and $< l+1 : 2v >$ to compute $K_{<l:v>}$. The formula above tells us that for every Node $< l : v >$, that is not a leaf, its private session key $K_{<l:v>}$ can only be computed by its children.

Assume that each leaf Node $< l : v >$ hosts the member M_i, therefore the Node $< l : v >$ has M_i's session key pair: $(K_{<l:v>}; BK_{<l:v>})$ that are pre-computed. Every member M_i hosted at a leaf node in the initial tree has the goal of computing the private session key: $K_{<0:0>}$.

In figure 23, if a member M_1 is hosted at node $< 2 : 0 >$, it follows the following steps to compute $K_{<0:0>}$:

(Step: 1) Compute its parents Key using:	$K_{<1:0>} = g(K_{<2:0>}, BK_{<2:1>})$	(Compute common key$K$)
(Step: 2) Compute the corresponding Bkey:	$BK_{<1:0>} = f(K_{<1:0>})$	
(Step: 3) Synchronization Step		
(Step: 4) Compute the root Node session key using:	$K_{<0:0>} = g(K_{<1:0>}, BK_{<1:1>})$	

As we can see from the algorithm, M_1 requires $BK_{<1:1>}$ to compute $K_{<0:0>}$. However he cannot compute $BK_{<1:1>} = f(K_{<1:1>})$ himself. This is what the $(Step: 3)$ is for, at this stage members M_i hosted in the leaves of the tree, communicate with each other to exchange any computed $BK_{<l:v>}$.

In figure 23, we have 4 Members = $\{M_1, M_2, M_3, M_4\}$ hosted at $< 2 : 0 >$, $< 2 : 1 >$, $< 2 : 2 >$, $< 2 : 3 >$ respectively. M_1 computes $BK_{<1:0>}$ and sends it to M_3 and M_4 hosted at $< 2 : 2 >$ and $< 2 : 3 >$ respectively. At the same time, M_3 computes $BK_{<1:1>}$, and sends it to M_1 and M_2 hosted at $< 2 : 1 >$.
This algorithm requires a Synchronization step for each intermediate layer in the binary tree. In section 7.1.5 we will analyze another algorithm, which allows Assets to compute the common Secrete Key K asynchronously.
$K_{<0:0>}$ represents the common secrete key.

We will use a binary tree to represent the registered identities in the blockchain. The binary tree can be 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 22:

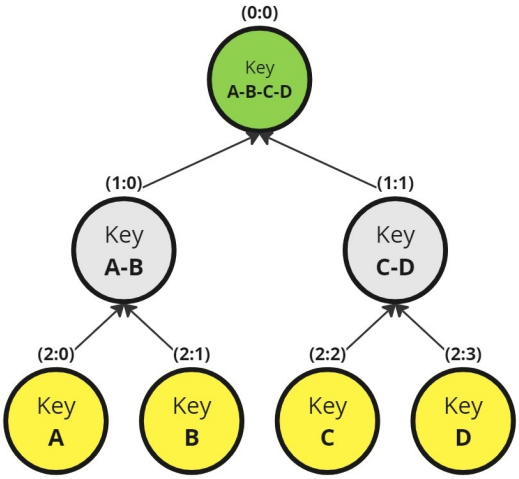


Figure 24: Binary tree state

Each leaf Node in the tree represents an Asset registered in the BC. Therefore, each leaf Node is associated with a Private/Public key pair, or $(K_{<l:v>}; BK_{<l:v>})$.

At this stage, everyone Asset can use $K_{<0:0>}$ to compute the symmetric key K that will be used to encrypt and decrypt messages. $K = hash_{256}(K_{<0:0>})$, whilst $K_{<0:0>}$ is secure, and could technically be used directly as the encryption and decryption key, it is advised to not used it directly, for an additional layer of security. Assuming three messages encrypted with key (K) are produced to the topic, this would be the state of the Kafka topic:

Figure 25: Kafka state after 3 messages sent encrypted with key K

6.5 Integrity and Provenance

So far the proposed architecture has proposed a solution for the requirements described in section 5.3 **Availability**, **Resilience**, **Efficiency** and **Confidentiality**. In this section we will focus in requirements **Integrity** and **Provenance**. Both integrity and provenance can be can be addressed by having the producer of a message sign the hash of the message that is to be sent [53]. The figure bellow illustrates this process:

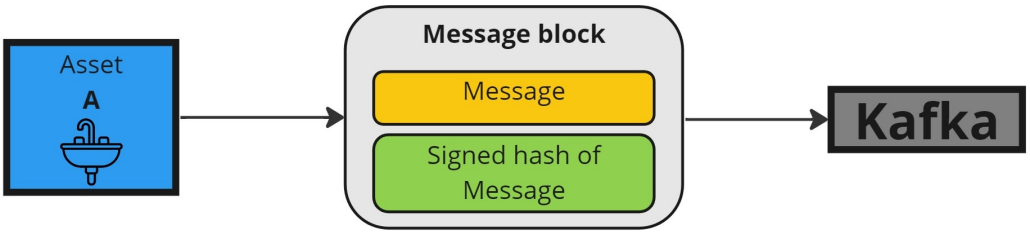


Figure 26: Hash of message signed

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:

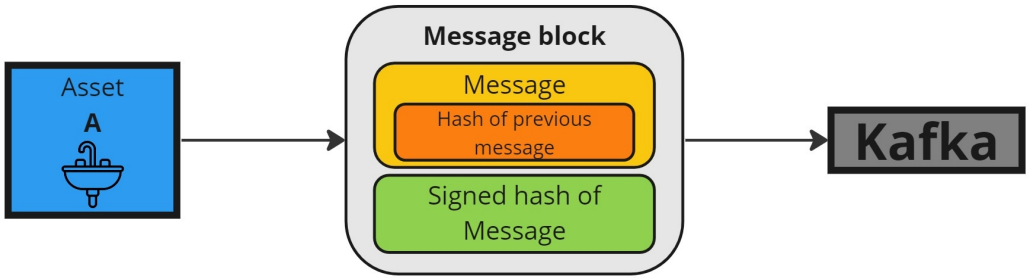


Figure 27: Hash of message with hash of previous message signed

With the addition of this new security measure, the architecture is compliant with all the requirements announced in section 5.3.

6.6 Use Case 3: 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 5.2.3 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.

In our specific use case, the ordering and delivering steps are continuous and not isolated events. Without the BC we required a TTP, usually a bank, to validate a payment. The workflow is as follows:

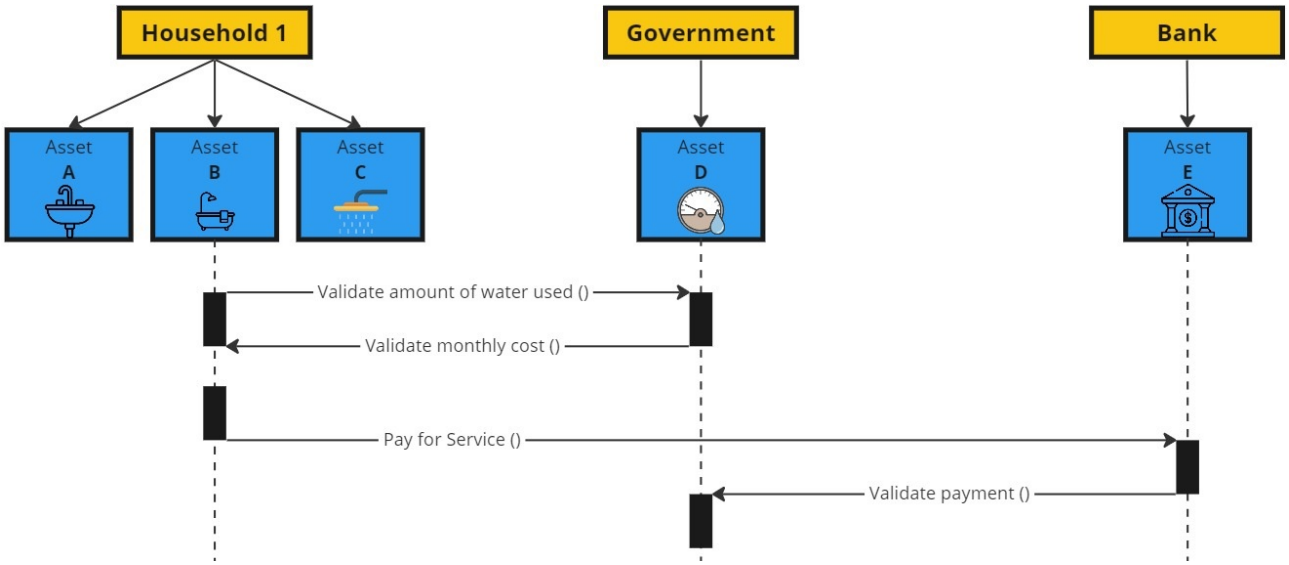


Figure 28: Sequence diagram of payment validation by TTP

By using the BC as our TTP, via SCs, we completely remove the bank from the transaction. 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:

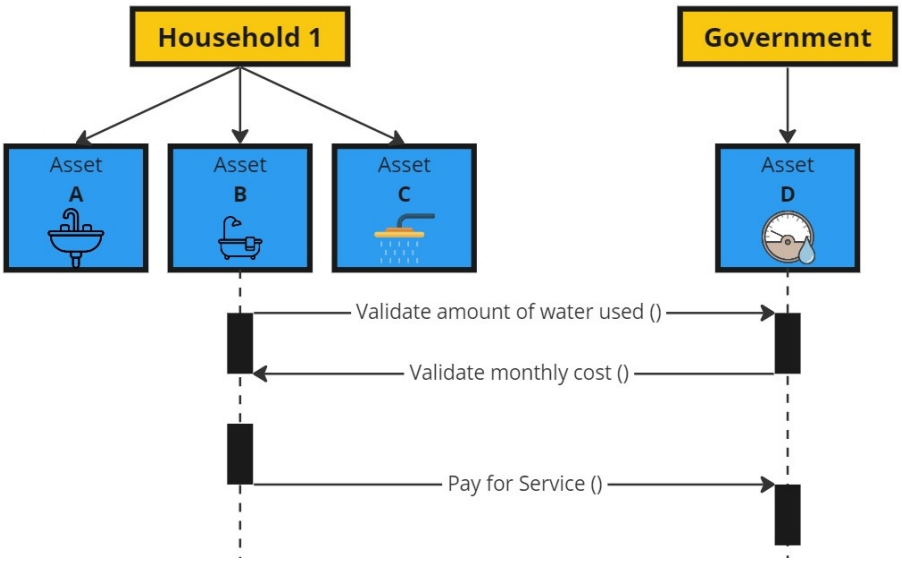


Figure 29: Sequence diagram of payment validation by BC

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:

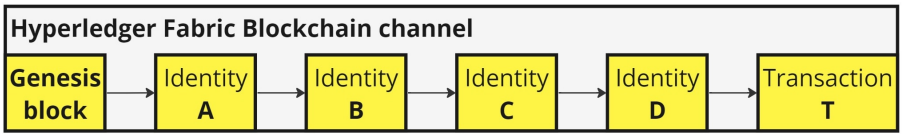


Figure 30: Blockchain status after a business transaction

7 Limitations

In this section we will address some limitations of the proposed architecture, whilst providing solutions.

7.1 Common key computation

In previous section 6.4.3 we saw how this architecture proposed to compute a common secrete key for symmetric encryption. Unfortunately the proposed solution brings some related problems which we will now analyse.

7.1.1 Use Case 2: New Identity registered

Let's assume we transition from Use Case 1 to Use Case 2, as described in sections 5.2.1 and 5.2.2. A new organization with 1 asset, called E, joins the private BC. This new identity must be registered to the BC. Let's inspect the state of the BC at this stage:

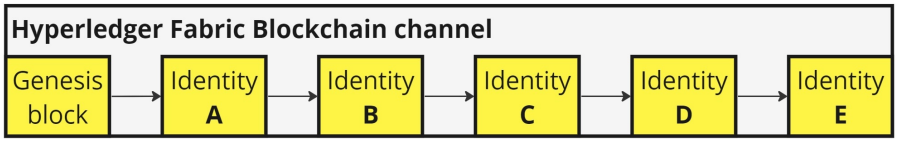


Figure 31: Blockchain status after new identity registration

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:

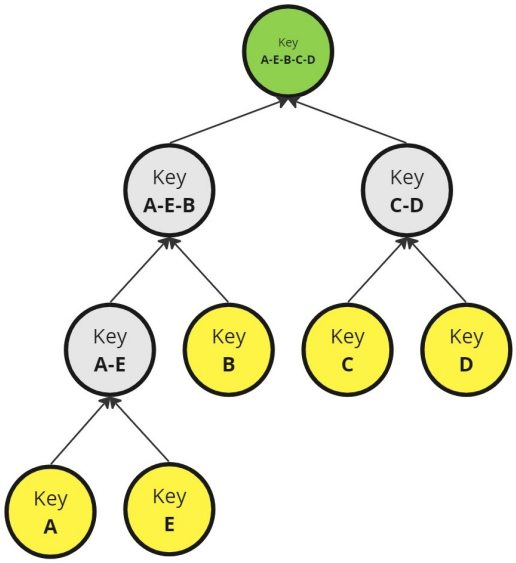


Figure 32: Binary tree state with additional identity

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:

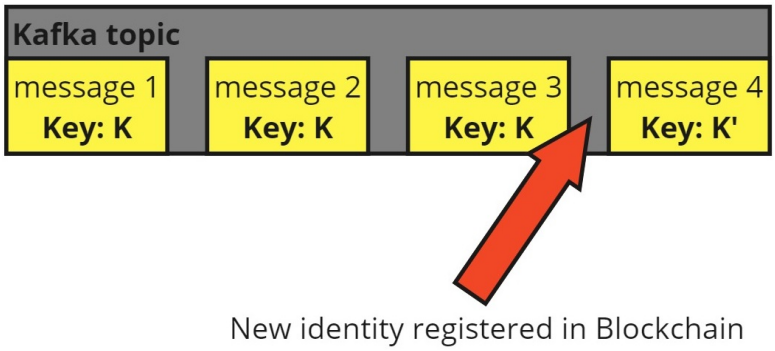


Figure 33: Kafka state after 4 messages sent encrypted with key (k) and key (k')

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.

7.1.2 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:

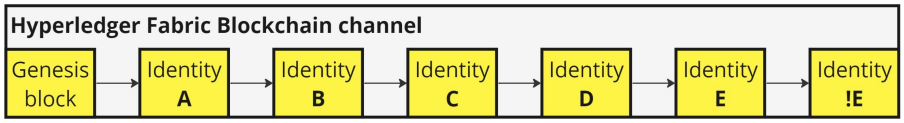


Figure 34: Blockchain status after identity removal - !E

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.

7.1.3 Key freshness

In the previous three sections the state of the ledger changed quite a bit. We started at section 6.4.3 and computed common key (k). Following that, a new identity was registered in section 7.1.1 and common key (k') was computed. Finally in section 7.1.2, 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 6.4.3.

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:

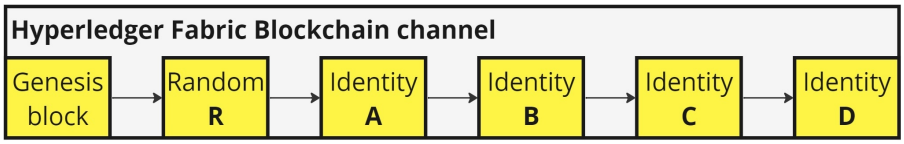


Figure 35: Blockchain status with random state (R)

And its corresponding binary tree:

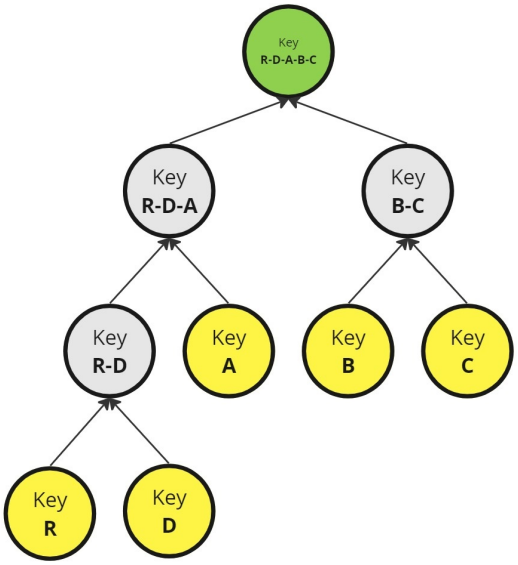


Figure 36: Binary tree state with random state (R)

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:

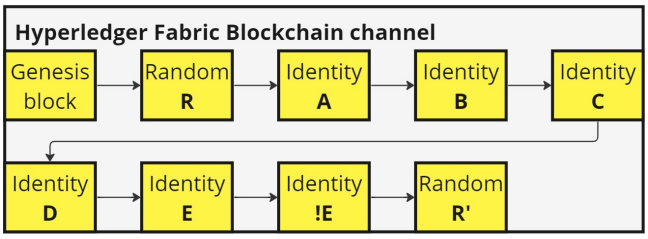


Figure 37: Blockchain status with random state (R') after invalidation of E

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'):

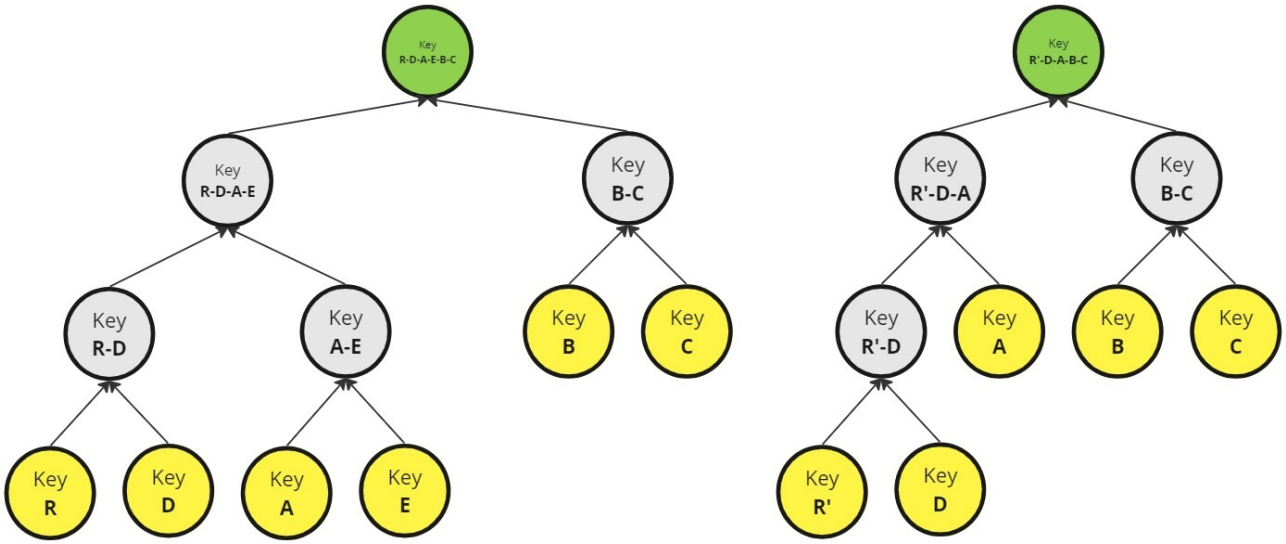


Figure 38: Left: Binary tree state with identity E and random state (R) - Right: Binary tree state with invalidated E and random state (R')

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 are assured that the commonly computed key (k) from the binary tree in figure 36, is different from the commonly computed key (k') from the binary tree in figure 38 (right side). Even though in both trees the exact same identities are registered.

7.1.4 Performance

It can become quite costly to compute the common secrete key, especially for small computing power units. To make this protocol lightweight we can compute the common secrete key using Elliptic Curve Diffie-Hellman [54], which is the default PKI for the x.509 certificates.

7.1.5 Synchronization step

Let's assume the binary tree hosts 4 Members = $\{M_1, M_2, M_3, M_4\}$, if M_1 creates the private channel, it cannot compute the common secrete Key K, until all the other Members M_i become online and start the computation process themselves. This is because $(Step: 3)$ of the algorithm described in (Compute common key$K$) is a blocking operation.

To avoid this synchronization step, the authors of [55] suggest a new algorithm. Let's suppose the binary tree hosts 4 Members = $\{M_1, M_2, M_3, M_4\}$, and M_1 is the only member online at the time of creating the channel. Instead of using the public keys of the other Members M_i, M_1 will use **prekeys** as well as **setup** keys. Let's define every component:

- the **setup** key is a DH (Private;Public) key pair: $(suk; SUK)$ generated by the channel initiator M_1. M_1 broadcasts SUK to every other Member M_i.
- the **prekey** were first introduced in [] for asynchronicity in messaging apps. They are a DH (Private;Public) ephemeral key pair: (ek_{M_i}, EK_{M_i}) $\forall M_i$. They are **pre** computed by each Member M_i, EK_{M_i} are sent to and stored by a untrusted intermediate server.
- the **identity** key is a long term DH (Private;Public) key pair: (ik_{M_i}, IK_{M_i}) for Member M_i, obtained from PKI.

→ The channel initiator, M_1, knows: $(suk; SUK)$, EK_{M_i} and IK_{M_i} $\forall M_i$.

The first step is for M_1 to setup a new (Private;Public) Key pair: $(\lambda_j^u; g^{\lambda_j^u})$, where u is the corresponding identity and j a key counter, for every other member M_i hosted in the tree:

Compute Private Key: $\lambda_1^{M_i} = KeyExcahange(ik_{M_1}, IK_{M_i}, suk, EK_{M_i})$ (Algorithm Initiator)
Compute Public Key: $g^{\lambda_1^{M_i}}$
new Key Pair for M_i: $(\lambda_1^{M_i}; g^{\lambda_1^{M_i}})$

This Key pair can only be computed by the initiator who holds suk and ik_{M_1} secrete, as well as by the corresponding Member M_i who holds both ik_{M_i} and ek_{M_i} secrete.

By knowing every (Private;Public) Key pair for every Member M_i hosted in the tree, M_1 is capable of computing every (Private;Public) Key pair associated with each Node in the tree, including the root Node.
The next step is for M_1 to broadcast every Public key of every intermediary Node to every other Member M_i.

As soon as any other member M_i is online, it is capable of computing:

Compute his own Private Key: $\lambda_1^{M_i} = KeyExcahange(ik_{M_i}, IK_{M_1}, ek_{M_1}, SUK)$ (Algorithm non-Initiator)
Compute Public Key: $g^{\lambda_1^{M_i}}$
new Key Pair for M_i: $(\lambda_1^{M_i}; g^{\lambda_1^{M_i}})$

Upon computing his own (Private;Public) Key pair, it can compute every (Private;Public) Key pair associated with every Node in its path up to the root, by using the public keys that were broadcasted by the initiator M_1.

One important property about the function $KeyExcahange()$ is that:

$$KeyExcahange(ik_{M_1}, IK_{M_i}, suk, EK_{M_i}) = KeyExcahange(ik_{M_i}, IK_{M_1}, ek_{M_1}, SUK),\ \forall\ i$$

At this point each M_i is able to compute a new $(\lambda_2^{M_i}; g^{\lambda_2^{M_i}})$. Recompute the new (Private/Public) Key pair associated with every Node in its path up to the root, and broadcast the new Public Keys.

Let's assume a scenario where M_1 creates such a channel and follows the algorithm described in Algorithm Initiator, with M_2 and M_3 that follow the algorithm described in Algorithm non-Initiator:

(1) M_1 computes the first session Key K_1 and sends a message m_1 encrypted with K_1: $encrypt(m_1, K_1)$

(2) M_2 comes online, computes K_1 to read m_1, creates a new (Private;Public) Key Pair for himself. Resulting in a new session Key K_2 and sends message m_2 encrypted with K_2: $encrypt(m_2, K_2)$

(3) M_3 comes online, computes K_1 to read m_1, computes K_2 to read m_2, creates a new (Private;Public) Key Pair for himself. Resulting in a new session Key K_3 and sends message m_3 encrypted with K_3: $encrypt(m_3, K_3)$

As we can see every Member M_i needs to compute every session Key when it comes online to read previously sent messages.

7.2 Business Transaction secrecy

In previous section 6.6 we saw how this architecture proposed to add traceability for business transactions. Unfortunately some problems were left unsolved.

These business transactions would be registered in the same BC as where City Actors are registering Assets identities. The number of Node peers could become quite large, as some of these Assets could be something as granular as sensors. Having each Node peer store a local copy of the BC might become increasingly hard, especially when adding blocks representing business transactions.

Secondly, organizations would be having these business transactions in a BC where one of the organizations in the perimeter is the Federation, as we can see in figure 18. The organizations behind these business transactions might want to keep them private.

To solve both these shortcomings, we suggest a simple solution: To create two channels.

The first channel would be dedicated for Asset identity registration, every Asset under each organization would be registered, as well as the Peer nodes under the control of the Federation. These would allow Peer nodes to still compute the common secrete key, used to exchanged in the Kafka topic, whilst keeping the storage load on these Assets as low as possible.
The second channel would only be for the two organizations, and a subset of their Assets. In this channel both organizations could conduct business transactions securely.

The figure below illustrates this solution:

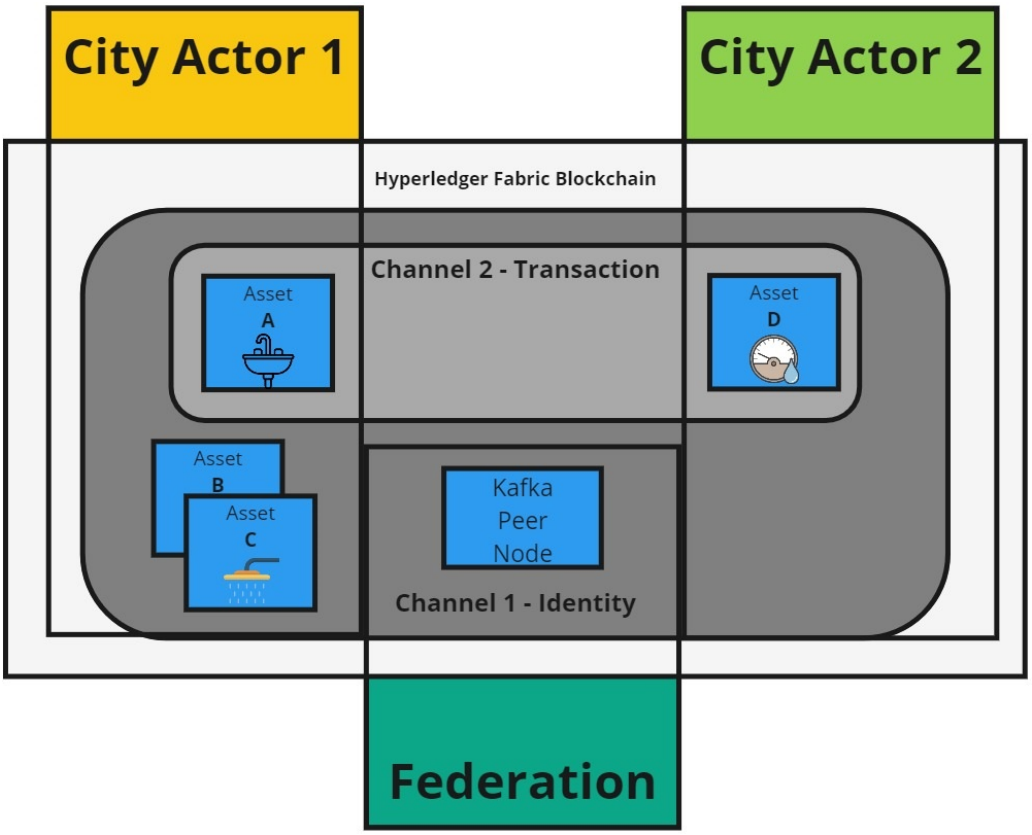


Figure 39: Hyperledger Fabric Blockchain with Identity channel and Transaction channel

Which Asset would join which channel would be entirely dependant on the organization itself.

8 Choice of technology

In this section we will discuss the choices of technology made in our architecture. There were 2 main choices made, the first was the communication platform, and the second was the Identity management and Access Control. One of the advantages of how the system was designed, is that both choices are completely independent, but work together to make sure only authorized entities have access to specific data.

8.1 Communication platform

The communication platform is a crucial aspect of the system, dictating how data is shared and accessed across various Peer nodes in the network. Kafka was selected based on several key factors, including its **availability**, **resilience** and **efficiency** which were some of the requirements stated in section 5.3.

8.1.1 Apache Kafka architecture

Kafka provides a real-time publish-subscribe solution. Kafka's is distributed in nature it typically consists of clusters that have multiple brokers [51]. To balance the load, topics are split into multiple partitions, which are stored in one or more brokers. The figure bellow illustrates the design of Kafka:

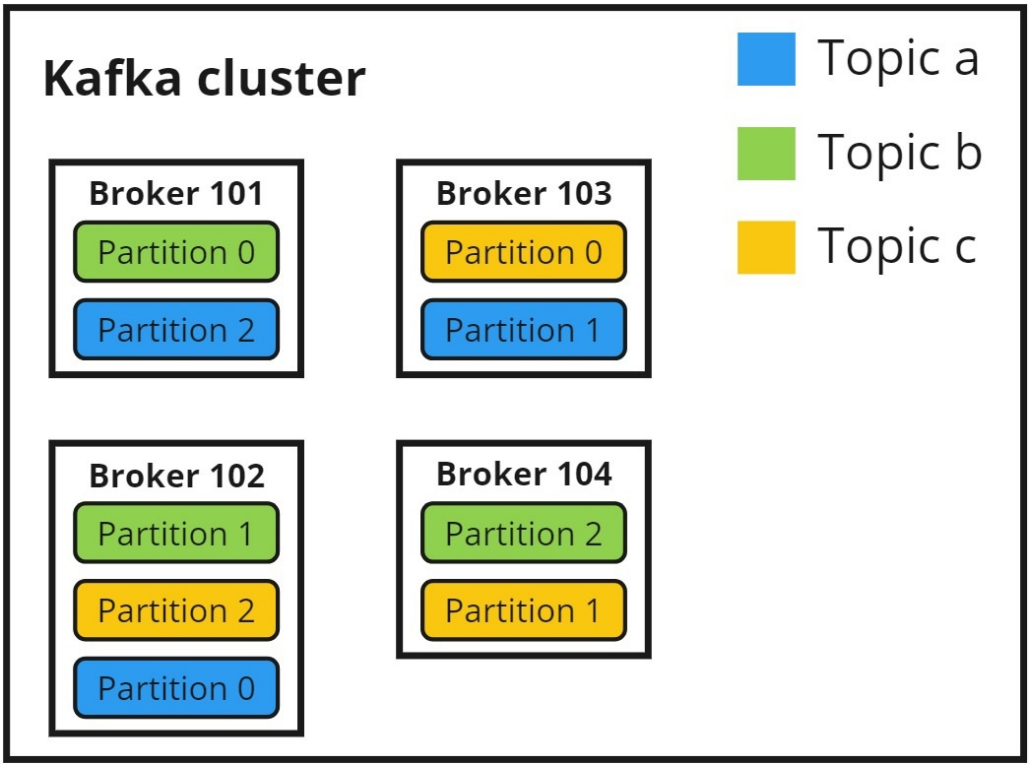


Figure 40: Example of Kafka cluster 4 brokers, each containing partitions of different topics

This architecture allows Kafka to scale as much as required. This means that on top of offering **availability**, **resilience** and **efficiency**, it is also able to **scale** in environments with multiple producers and consumers, whilst maintaining availability.
In [56] the authors proposed a system that combines Apache Kafka, for online and offline consumption of messages, and Apache Spark to stream and process the data provided in real time by IoT devices.

Kafka's distributed nature coupled with its architecture design are what allows it to scale so well and avoid a single point of failure.
This is why we believe the communication platform should ideally be decentralized by nature, so it maintains its **availability**, **resilience**, **efficiency** and **scalability**. While we've chosen Apache Kafka for our system due to its strengths, it's worth noting that other technologies with similar characteristics could also be suitable replacements, provided they meet the requirements, and have the same functionalities.

However, it is worth noting that it is not its decentralized nature that assures **confidentiality**, **integrity** and **provenance**, the other requirements mentioned in section 5.3. Those are achieved by design as explained in sections 6.4 and 6.5.

Following that we will go over the choices made for Identity management, and Access control, we will discuss them separately.

8.2 Identity management and Access Control

8.2.1 Identity management

The identity management platform's goal is to give an unique form of identity to each individual Asset. By doing so we can have Assets sign their messages or even transactions. Ultimately this will help with the overall data **provenance** problem, as well as transaction **traceability**.

In the proposed architecture we decided to use the BC technology to deal with the lack of trust amongst the various City Actors in each other. We choose to implement our Identity management through HFB as described in section 6.2. HFB uses x.509 certificates for identity based on PKI. These certificates are issued by a CA, the advantage being that each organization within the network controls their own CA, which allows for a more decentralized trust model.

If we'd want to avoid using the BC, to avoid related computational costs, we could look at protocols such as Certificate Transparency Logs(CTL) described in [57]. They rely on a trusting original Certificate Authority, who owns a root certificate. The root certificate would be controlled by a TTP, who would issue intermediate certificate to the various City Actors participating in the network. These intermediate certificates, would be restrict to a certificate domain owned by the specific City Actor, implying that they could only issue certificates in their own domain. Each certificate issued is added to a append-only log, this provides transparency in the system. The figure bellow illustrates the different hierarchy of trust for both certificate protocols:

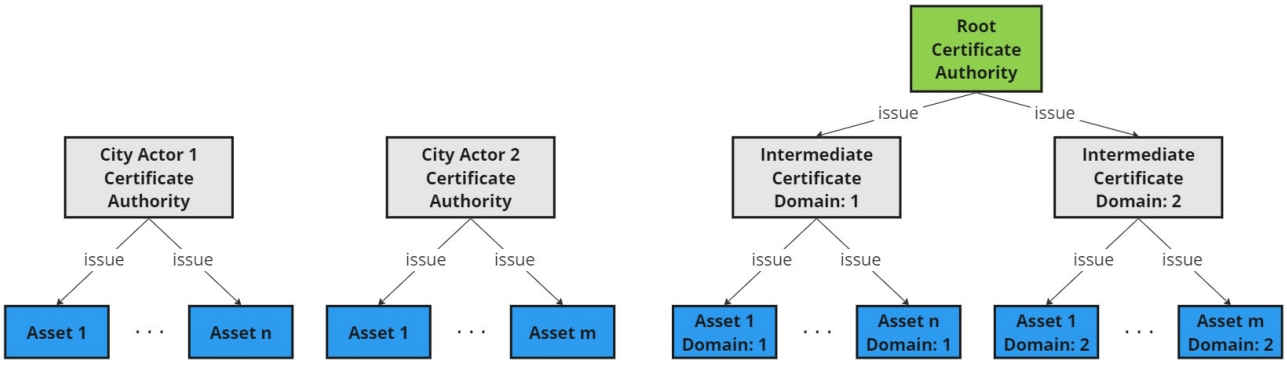


Figure 41: Left: Certificate Issuance in Hyperledger Fabric - Right: Certificate Issuance in Certificate Transparency Logs

As we can see, in the CTL there exists a Root CA, which represents single point of failure.

8.2.2 Access control

Access control is the way the data creator or data owner decides which entity has access to the data they share. In HFB this is managed through the concept of private channels. Channels are by design private, only visible to allowed organizations, therefore they can serve as a communication setup channel between specific network participants.

This contrasts with User-Managed Access(UMA) which puts the data creator in direct control of authorizing data sharing with online services, UMA offers a high degree of user control. However, in [58] the author identifies some of its pain points:

- **Availability:** UMA relies on a centralized authorization server, which introduces a single point of failure;
- **Transparency and Traceability:** In a typical third-party access control service, the authorization process cannot be traced, which makes it difficult for users to have visibility on the actions of the system;

- **Maintainability:** Once deployed, updating the system to a new-version can be costly and time-consuming, this impacts the upgradability of the system.

The author goes on to implement UMA in a BC(B-UMA) environment to try and solve the identified pain points. The results were promising, however he does recommend considering the limitations of complexity and economy when designing B-UMA systems.

9 Proof of Concept (PoC)

9.1 Introduction

9.1.1 Purpose

The purpose of this PoC is to validate the feasibility and functionality of a decentralized platform for secure data sharing among different organizations, described in detail in section 6, based on the use case described in 5.2.1.

9.1.2 Background and Context

This PoC is built in the context of smart cities and smart water grids, with an emphasis on secure data sharing. The rapid development of digital infrastructure has created opportunities for advanced water management through smart water grids.

9.1.3 Scope

This PoC focuses on the deployment of a HFB network with two organizations, each having one peer node. It also involves the creation and use of SCs for identity registration and generation of random public keys.

9.1.4 Success criteria

The PoC will be considered a success if we are able to create a secure group channel as described in section 2.3.

9.2 Implementation

For the PoC to be successful we need to combine many different technologies. In this section we will go over the process of setting up each component of the whole architecture.

9.2.1 Setup

First lets go over version check.
Everything was ran in a Virtual Machine, using Oracle VM VirtualBox Manager [59]. The Operating System (OS) is Linux, Ubuntu distribution version 22.04.2 [60].

Parameter	**Specification**
Operating System	Ubuntu 22.04.2
Processor	6 cores
RAM	16 GB
Storage	50 GB

Table 4: System Specifications

9.2.2 Hyperledger Fabric

For our network, we used HFB's official Test Network [61].
After each installation step these were the versions of every software:

Software	Version	Description
git	2.34.1	A distributed version control system used to clone and manage the versions of the software repositories.
curl	7.81.0	A command-line tool for transferring data using various protocols, used for interacting with REST APIs, downloading files, and testing network connections.
docker-compose	1.29.2	A tool for defining and running multi-container Docker applications. It defines the HFB network architecture and services.
docker-py	5.0.3	A Python library to interact with Docker's remote API, used to programmatically manage Docker containers.
docker	20.10.21	A platform to develop, ship, and run applications within containers. Used to containerize and isolate the HFB network services.
go	go1.20.5	A statically-typed compiled language used for writing and managing Smart Contracts.
jq	jq-1.6	A lightweight and flexible command-line JSON processor, used for parsing and manipulating JSON data.
fabric binaries	v2.5.2	Essential command-line tools provided by the Hyperledger Fabric project to setup and manage the network.
gradle	8.1.1	A build automation tool used to compile and build Java projects such as client applications.
java (openjdk)	11.0.19	Used to develop client applications that interact with our HFB network.

Table 5: Software Tools and their Descriptions

At this point it is possible to easily deploy the test network, its configuration is two different organizations, each with a single peer Node. Each organization has its own CA, however as it is a test environment, there is a single Node Raft ordering service. This is not an advisable production network setup.

9.2.3 Hyperledger Explorer

To help with the visualization of the BC, Hyperledger Foundation developed a tool called Hyperledger Explorer [62]. This is not a mandatory tool, it is a read only visualization of the ledger, it is great to see if SC were installed correctly as well as if they were invoked as intended.

After launching the test-network, Hyperledger Explorer gives us this view:

				Ledger Height		
Peer Name	Request Url	Peer Type	MSPID	High	Low	Unsigned
peer0.org1.example.com:7051	peer0.org1.example.com:7051	PEER	Org1MSP	0	3	true
peer0.org2.example.com:9051	peer0.org2.example.com:9051	PEER	Org2MSP	0	3	true
orderer.example.com:7050	orderer.example.com:7050	ORDERER	OrdererMSP	-	-	-

Figure 42: Blockchain state after deployment of test-network

In the figure above we can see which Peers are in the defined perimeter of the network, first the single orderer Peer which composes the ordering service. After that we have both Peer Nodes, one for each organization.

9.2.4 Smart Contracts

We wrote two SCs for this PoC, both of which are written in the Go language [63].

1. **Register SC:** This smart contract functions as a ledger recorder. It records a key-value data pair to the ledger as a block. Each peer node in our network uses this SC to register their own Digital Identity (DID) and DID Document. In simpler terms, a peer node 'announces' its presence and credentials to the rest of the network using this SC. It does

so by creating a unique identifier (the DID), based on its own public key. The Register SC then records this DID, along with the peer's public key, onto the blockchain. Once this happens, the peer node can be recognized and authenticated by other nodes in the network.

2. **Random SC:** This SC works similarly to the Register SC, but instead of recording real, existing keys, it generates and records a brand new key pair. After generating a new pair of keys, it calculates a DID just like the Register SC, and records the DID and the corresponding public key to the blockchain, the private key is immediately deleted. This SC is used to create a temporary, random entity in the network, which is useful to have key freshness, as thoroughly explained in section 7.1.3.

Smart Contract	Key	Value
Register	$DID = hash_{256}(Public\ Key)$	$Public\ Key$ [1]
Random	$DID = hash_{256}(Public\ Key)$	Random: $Public\ Key$ [2]

Table 6: Smart Contracts: Register and Random

Although their function is the same, once invoked, thanks to them being two different SCs, Peer Nodes in the network can easily detect which SC was executed, and process it accordingly.

It is important to note that neither SC implements the business logic themselves. It is fully implemented before the invocation. This is a desirable property since each validating peer must execute the SC operation to validate it, by keeping the SC simple, we keep the computations by each peer light, whilst keeping traceability to each transaction.

Both SC have to be individually installed in each Peer Node, by installing they are effectively accepting its deployment to the BC. After every Peer Node has installed the SC, one of them can deploy it to the SC. Meaning that there are three operations per SC, as wee can see in the figure below:

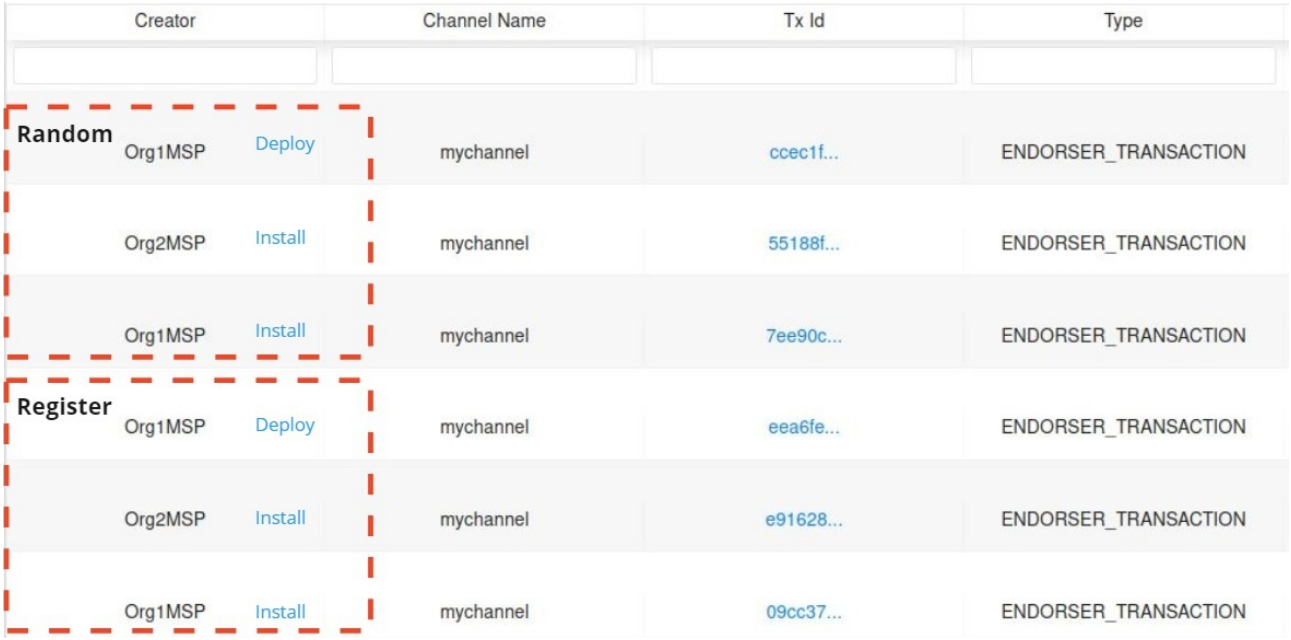


Figure 43: Smart Contract Install and Deploy operations

9.2.5 Java SDK

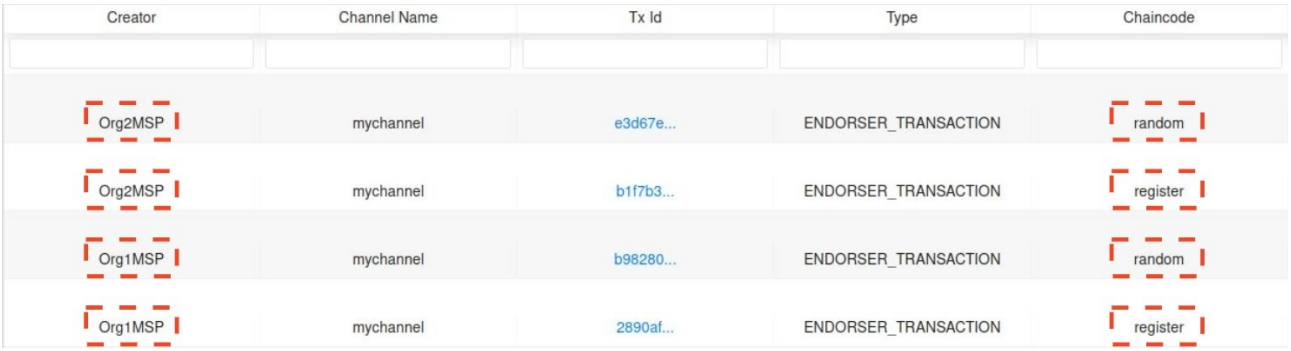
The Java SDK tool facilitates interaction with the ledger. There are two main Java processes.

1. **Event Listener:** This component monitors the blockchain for any new transactions. Whenever it detects a new transaction, it triggers a Python script execution. This script logs comprehensive transaction details to a .log file.

[1] The protocol described in 6.4.3 uses $Blinded\ Key$ rather than $Public\ Key$, but they represent the same thing
[2] The protocol described in 6.4.3 uses $Blinded\ Key$ rather than $Public\ Key$, but they represent the same thing

2. **Smart Contract Invoker:** This process is responsible for invoking the SCs. It uses the invoking Peer's crypto information, to be able to connect to the ledger, and proceeds to use the SC with costum values. These values are computed with the help with business knowledge implemented in the Java scripts.

In our PoC we want each Peer Node to invoke each SC once, resulting in the transactions we see in the figure below:



Creator	Channel Name	Tx Id	Type	Chaincode
Org2MSP	mychannel	e3d67e...	ENDORSER_TRANSACTION	random
Org2MSP	mychannel	b1f7b3...	ENDORSER_TRANSACTION	register
Org1MSP	mychannel	b98280...	ENDORSER_TRANSACTION	random
Org1MSP	mychannel	2890af...	ENDORSER_TRANSACTION	register

Figure 44: Smart Contract Invocation by each Peer Node

We can take a closer look at a transaction in the figure below:

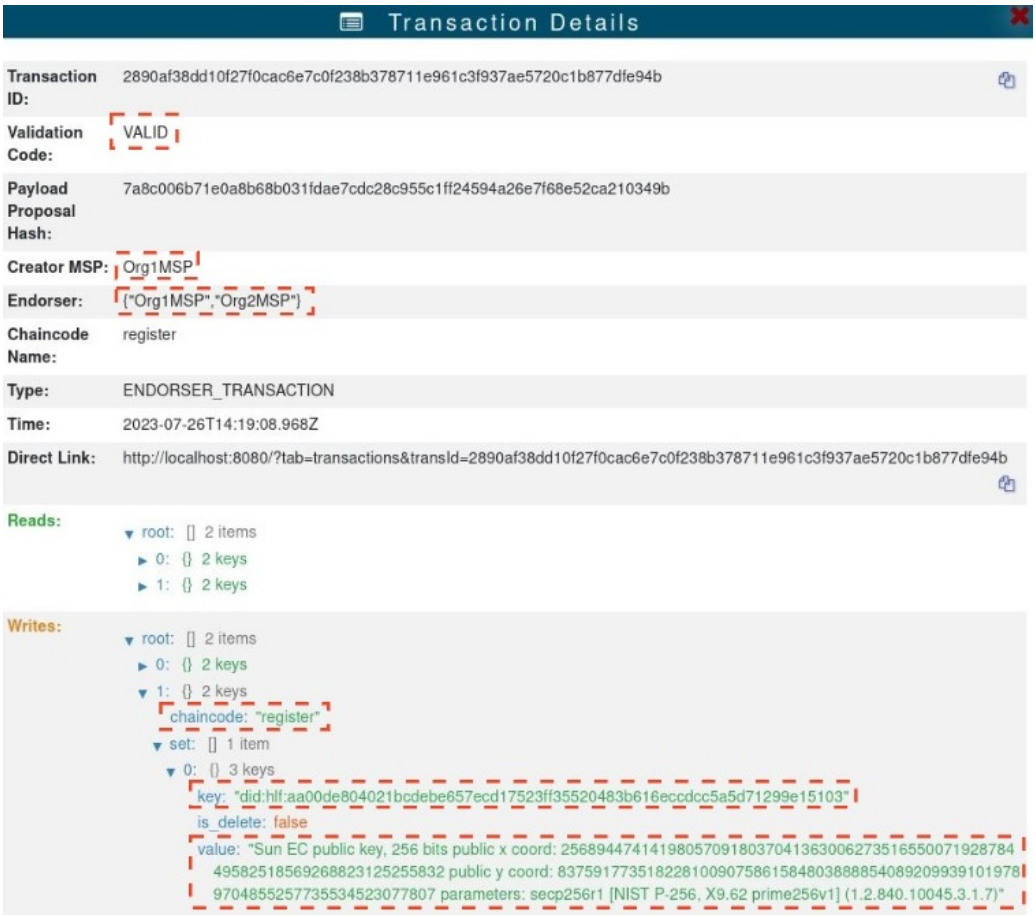


Figure 45: Register Smart Contract transaction details

We can see the creator of the transaction, the endorsers, the SC name, and finally the uploaded values, the Key, which is the DID, and the $Value$ which is the Public Key.
Thanks to the Java event listener, all this information is logged and can be accessible by a python script.

9.2.6 Binary Tree Group Diffie-Hellman

From the list of transactions shown in figure 44, there are four registered identities in the BC. Two real identities created with the register SC = $\{A, B\}$, and two fake ones created with the random SC = $\{R, R'\}$. For our PoC we choose to create a binary tree from these four identities, each registered identity will be represented as a leaf node in the binary tree as explained in section 6.4.3. As shown in the image below:

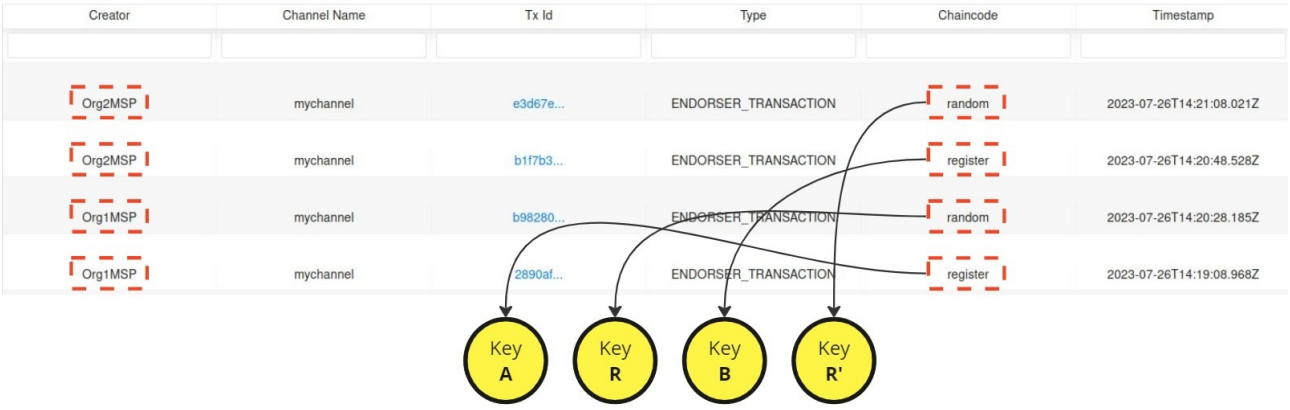


Figure 46: Registered Identities represented as leaf Nodes

It is important to note that we do not have access to the private keys for the two random identities, so for the protocol to work we need to alternate the real and random identities. This way the Node that hosts a real identity is able to combine his own K with his sibling's BK, which is registered in the BC, to compute their parent's K / BK Key pair. This results in the following binary tree:

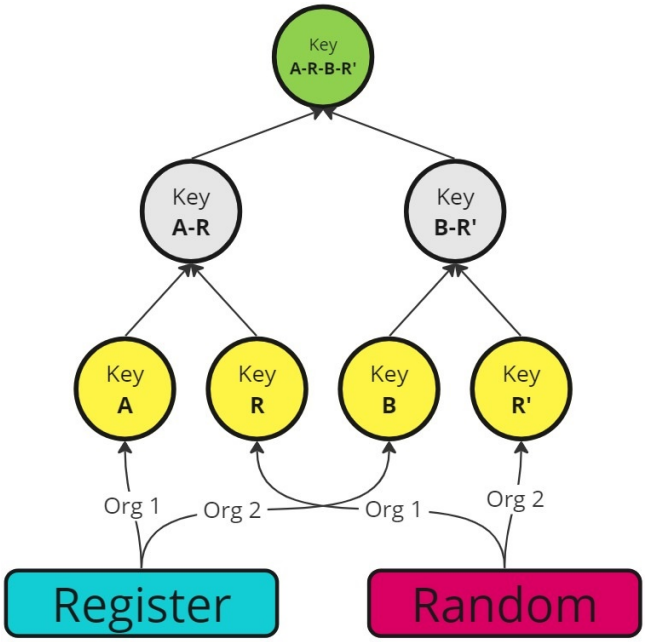


Figure 47: Binary tree

As we can see the root Node of the tree (in green) is represented by a key which is computed from the four original keys.

Finally, we need to execute two python scripts in parallel, one of them will represent the first real identity, and the second one represents the second real identity. Each of these create an instance of the binary tree. In the figure bellow we can see the step by step process of computing the common secrete key from the point of view of Identity A:

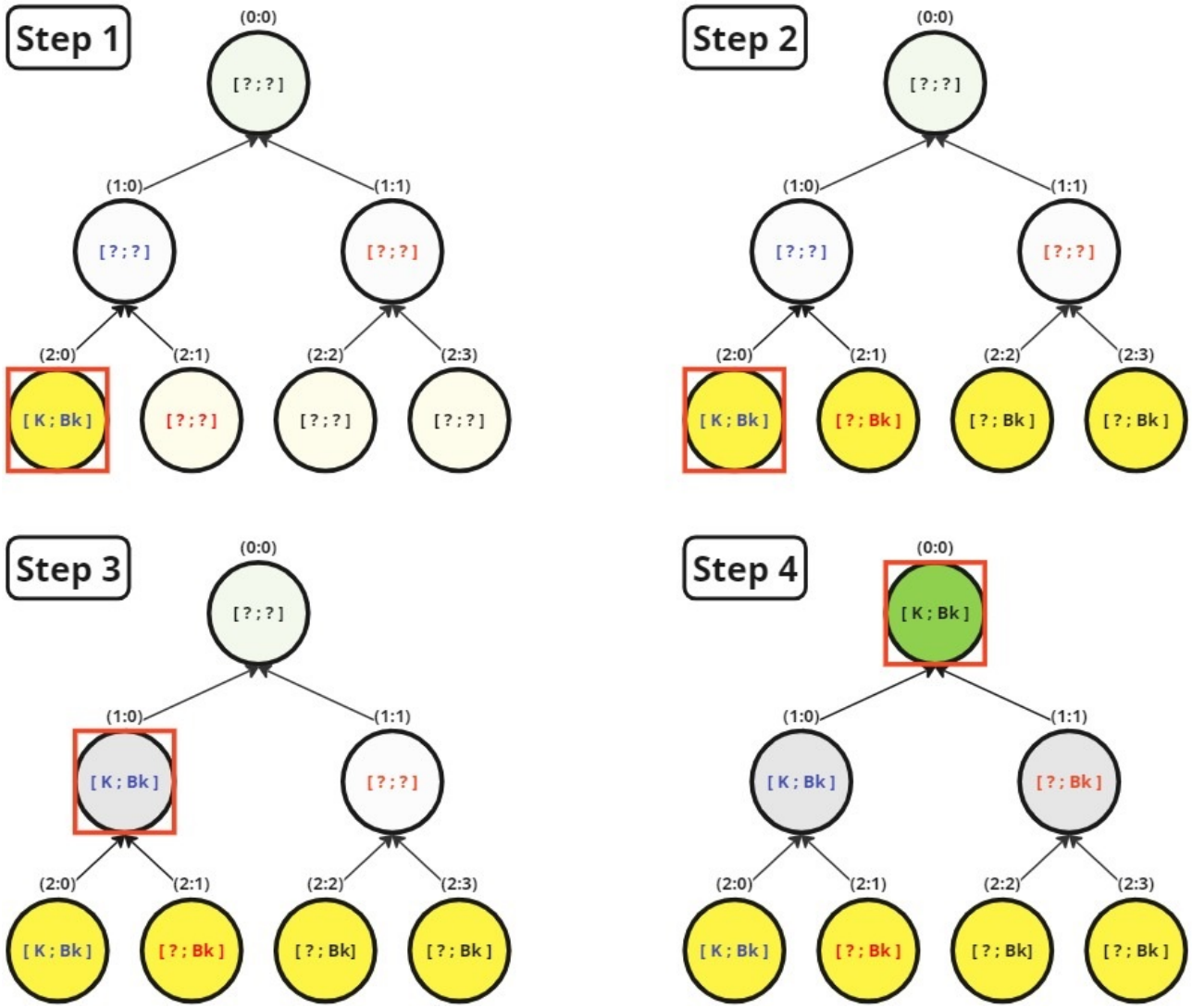


Figure 48: Step by step computation of common secrete key

Let's analyze each step of the process depicted in the figure above. The python script, let's call it process A, starts by generating the binary tree, which depends on how many Identities were registered to the BC.

Step 1 (top left) it knows only its own private and public key $[K,\ Bk]$, hosted at node $< 2:0 >$ (red square);

Step 2 (top right) it loads all the information about the other Identities to its respective leaf Node $\{< 2:1 >, < 2:2 >, < 2:3 >\}$, only their public keys are known, $[?,\ Bk]$, as they were registered to the BC via the SCs;

Step 3 (bottom left) Using its own secrete key $K_{<2:0>}$ and its sibling public key $Bk_{<2:1>}$, it can compute the parent's key pair $[K,\ Bk]$ hosted at node $< 1:0 >$ (red square);

Step k As process A started, a second process, process B, also started. However, it started at node $< 2:2 >$. This step represents the synchronization step described in algorithm Compute common key$K$ between Process A and Process B. The implementation of this synchronization step will be described in the next section 9.2.7;

Step 4 (bottom right) with the knowledge of $Bk_{<1:1>}$, process A can compute the root Node key $[K,\ Bk]_{<0:0>}$ (red square). Note that process B is also able to compute the root node key, that's why it represents the common secrete key.

9.2.7 Kafka

For a swift setup of the kafka server, we used a bitname docker image [64], with just a few modifications.
The kafka service is deployed at the same time as the network. As soon as both the Process A and process B described

in the previous section 9.2.6, are blocked, they connect to a specific topic, and broadcast their respective public keys, and listen to messages. This protocol is capable of functioning even if both processes don't connect at the same time to the kafka server, or have any connectivity issues. This step sits between the steps 3 and 4 depicted in figure 48.

In the figure below we can see both processes side by side:

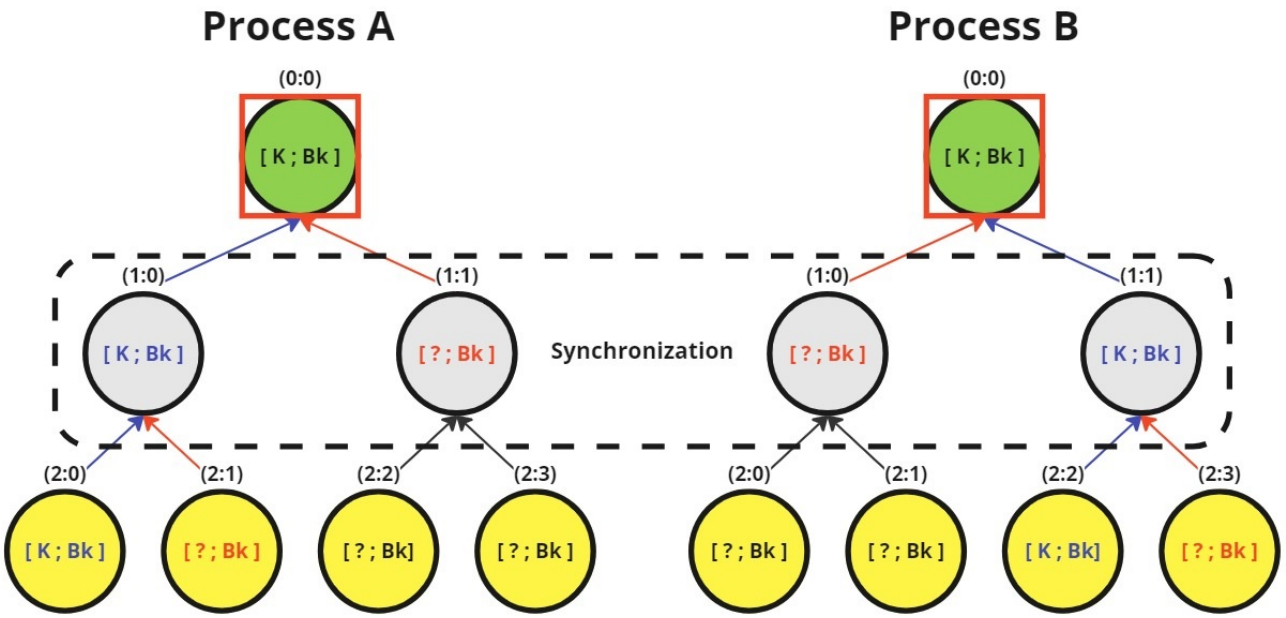


Figure 49: Common key computation - Left: Process A – Right: Process B

The "Synchronization" banner in the figure above, symbolizes the Kafka server, and the exchange of respective public keys between the processes. We can see that process A requires $Bk_{<1:1>}$ and process B requires $Bk_{<1:0>}$. As they connect to the Kafka server, and broadcast this complementary information, they will unblock each other.

9.3 Conclusion

The PoC was considered a success, as the required success criteria described in section 9.1.4 was met. We were able to setup a BC where two different organizations registered the digital Identities of their Assets. Through the registered identities, the Assets computed a secrete group key to exchange information through a secure channel.

10 Analysis

The architecture presented in section 6 was successfully validated by our PoC implementation described in section 9. However, it's crucial to acknowledge that the PoC may not entirely simulate the complexities and challenges inherent in a real-world deployment of our system.
In this section we will address key factors including identity revocation, security, performance, cost. These factors will validate the real-world applicability of the proposed solution.

10.1 Identity Revocation

There are multiple scenarios in which we'd like to revoke a registered Identity.
Although we didn't mention what revocation is in our overview of SSI in section 3.1, it is an important function of the overall SSI environment. Revocation consists of the removal of any issued VC.

In our system when an organization registers an Asset, it is essentially issuing a VC that validates the Asset's identity. If for any reason an organization would like to revoke such VC, it should be able to. This could happen if, for example, the Asset no longer physically existed. Based on our use case, described in section 5.2, we can easily imagine a scenario where one of the water terminals is physically removed from the house. Logically, its digital Identity should also be removed.

To implement this desired functionality, we propose the design and deployment of a new SC that flags previously registered identities as revoked. The basic idea was described in section 7.1.2. However, some key elements were left out. For such a SC to function correctly, we need to modify the endorsement policy, obviously, the concerned Asset cannot be asked to verify the transaction that invalidates its own identity. This adds a new dimension of difficulty to the various challenges involved in the development of SC identified in section 3.3.4.

There are however scenarios where revocation is more complicated, in the event an Asset is hacked, the revocation of it's identity becomes even more important. In such a scenario there are additional challenges, the first one is identifying that the Asset was compromised, this can only be achieved by the organizations themselves. The Federation that is maintaining the BC and the Kafka server does not, by design, have the power of block an asset, and does not have access to the messages being sent to perform data quality control. The second challenge is for the new transaction to be distributed to every Peer Node in the network. In the sequence diagram below we see the events following the compromise of an asset:

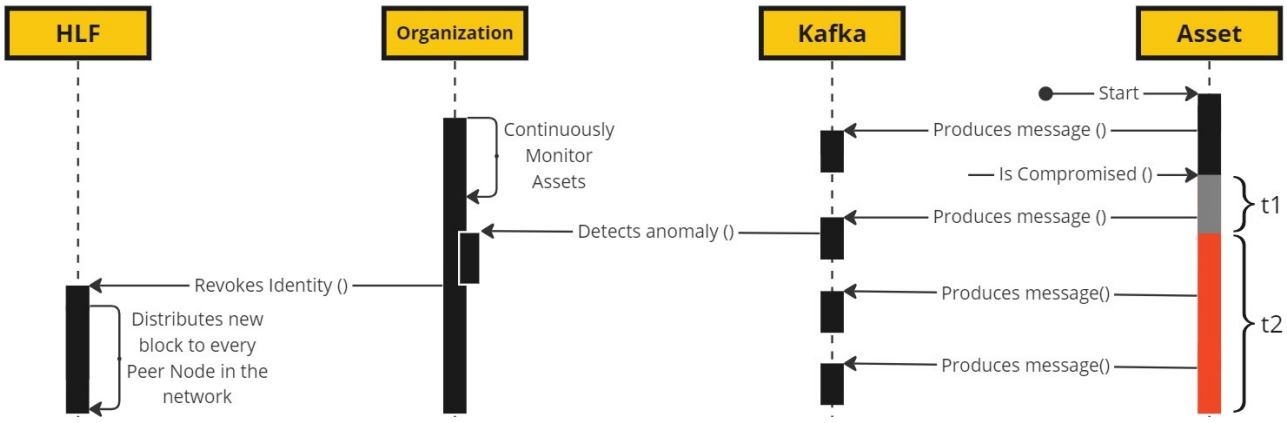


Figure 50: Sequence diagram - Process of Identity revocation

In this sequence diagram we see an Asset that is producing messages to the Kafka topic, when it is compromised. At this point it continues to produce messages to messages to the Kafka topic, at this point, the organization who is continuously monitoring its Assets, detects an anomaly, and declares the Asset to be compromised. This detection can take many shapes, and its completely under the responsibility of each organization. It invokes the revocation SC and afterwards the Network is responsible for distributing the new block to every Peer Node.

Let's set two variables, $t1$ = time it takes for an organization to detect its Asset has been compromised, $t2$ = time it takes for HFB to distribute a new transaction to every Peer Node. After an Asset is compromised, during a time $t = t1 + t2$ the attacker can both control which messages are produced, and consume every message produced by

other Assets.

There are improvements we could add to our system to avoid this issue, we propose to add a timer t_{delay} to each message produced to the Kafka topic, this timer would define a delay between when a message is produced and when the Kafka would allow other assets to consume it. Obviously the organizations maintaining each Asset would need to consume the messages as soon as possible. With this system we are making it so each organization is validating each message their assets produce before it is consumed. This adds a layer of difficulty to the overall system, as well as it delays the information, meaning that the Digital representation of the Physical object would essentially be living in the past.

10.2 Security

By looking at the overall architecture described in section 6, we can categorize potential security threats into two main categories. These categories are, Attacks on organization's infrastructure and Attacks on Federation's infrastructure.

Attacks on organization's infrastructure revolve around the Assets themselves, the organization that maintains each Asset, is solely responsible for their security. We identified two main challenges present throughout the Assets lifetime, the first challenge is avoiding man-in-the-middle (MITM) attacks when first issuing the x.509 certificate to the Asset. MITM attack is a cyber attack where the attacker C, sits in between two entities A and B, making himself pass as A to B, and as B to A. Fortunately, SSI infrastructure is designed with several layers of protection to prevent such attacks in runtime, the authors of [65] proposed an additional method of detecting and mitigation MITM attacks. The third challenge is making sure Assets aren't compromised in runtime, as mentioned in the previous section 10.1.

Attacks on Federation's infrastructure, mostly involve Denial of Service (DoS) attacks, these attacks are designed to take down infrastructure, making it unavailable for its intended use. These could be directed at both the Kafka broker as well as the HFB Network. Fortunately, they are resilient to these kind of attacks, as both lack an evident single point of failure, thanks to their distributed nature as previously mentioned in sections 3.2 and 8.1.1.

We can see that the most dangerous attacks fall under the first category. Meaning that it is up to the various organizations in the network to apply good security measures.

10.3 Performance and Cost

These two properties go hand in hand, when improving the performance of a certain system, for example by optimizing an algorithm to be faster, then you are probably reducing the cost of the overall system. Unfortunately, when adding security lawyers to a system, its overall performance will be worse. Ultimately this means that each security measure we implemented in our framework, slowed the overall performance, therefore increasing its final cost, weather it be initial cost, development cost, maintenance cost or running cost.

In the context of our use case and ultimate goal of having a DTN of an entire city, we need to think about the overall scalability of the framework. To analyse the performance we need to explore possible bottlenecks. Just as in the previous section 10.2, there are two categories of bottlenecks, Bottlenecks to the Organization's infrastructure and Bottlenecks to the Federation's infrastructure.

Organization's infrastructure is composed of many devices, in the use case we describe in 5.2, the devices would be the smart taps. These smart taps need to be both cost effective and can't take up much space, this means that they will most likely have low computing capabilities. This represents the first bottleneck, in our PoC we described a scenario where two assets A and B derive a common secrete key, as they need to advance through the binary tree at the same time, as described in section 9.2.6, if one of the two Assets possesses higher computational power than the other, he still won't be able to compute the common secrete key faster, since the Assets need to cooperate at each step. Additionally, after computing the common secrete key, each Asset still needs to be able to produce and consume messages to the Kafka server, each message requiring at least encryption or decryption.

Federation's infrastructure is composed of two main systems, HFB and Kafka. It is crucial that both function in a timely manner independently of the amount of users. In [66] the authors have successfully scaled HFB number of transactions from 3000 per second to 20000 per second. This was accomplished by implementing a series of independent optimizations focusing on I/O, caching, parallelism, and efficient data access. Moreover, Kafka has been proven in various IoT implementations that broadcast sensor data [67, 68]. One important note, as mentioned in section 8.1.1, Kafka can be swapped later if any other better options are available.

11 Future Work

In this section we will suggest some possible next steps that can be taken to further validate the capabilities of the proposed architecture. Our PoC was very rudimentary, it was just a basic setup to prove the link between the technologies was possible. With this in mind, we suggest that the future work should concentrate in proving the feasibility of the project in a more realistic scenario.

There are four important technologies in our system, the computation of the common secrete key, the SCs execution, the Kafka server and the HFB network described in the sections 6.4.3, 9.2.4, 8.1.1 and 4 respectively. In future work, it will be important to further test each technology in a scenario that is increasingly more realistic.

11.1 More Identities

Our PoC described in section 9 was basic in its design, it aimed to show that the integration of the technologies was indeed feasible. With this base established, we suggest that the next course of action should focus on demonstrating the practicality of the project in a more real-world context.

An interesting next step is to perform the same exercise with an increased number of identities registered in the network. In our PoC we registered only 4 Identities. It would make sense to attempt this process again but with 2^i different Identities, where $i = \{2,3,4,5,...\}$. To do this in the current HFB setup, which consists of two organizations with one Peer Node each, we need a new SC, let's call it **Dummy Identity SC**. This SC would be for test purposes only, it would do the same tasks as the **Random SC** described in section 9.2.4, however, it'd need to publish both the Private Key and the Public Key. This way each block in the BC would be one complete identity.

Once the SC is invoked 2^i times, there will be 2^i complete Identities registered in the BC. From there, execute a Python script that would first create a read only binary tree representation containing every Identities Public Key, let's call this tree *Global Binary Tree*, afterwards it would create 2^i threads, using existing libraries such as *threading* [69]. Each thread would be assigned a specific Identity, and would receive the corresponding Private Key, as well as receive access to the *Global Binary Tree*. Having access to the tree view, and a specific Private Key, each thread could represent one Identity and carry out its own process of computing the common secrete key. To do so they would follow the protocol detailed in section 6.4.3, and for the synchronization step they would join a Kafka server and start broadcasting as well as listening for their siblings Public Keys.

It should be noted that the current Python code couldn't support this type of use. For starters the current generation of the binary tree is mostly *hard-coded* and would need to be modified to accommodate any amount of Identities. In the current script, when the processes are connected to the Kafka topic, they accept the first message that wasn't sent by themselves, because if it wasn't them, then it was their sibling, as per explained in section 9.2.7, however, in a scenario where multiple threads are connecting themselves to the same Kafka topic, each process needs to listen to the broadcast of its own sibling. Additionally, when two siblings compute their parents Private and Public Key pair, only one of them should publish the Public Key in the Kafka topic, to not overload the server with duplicate messages, of course they both need to subscribe to the topic to receive the next Public Key they required to advance in the common key computation. These are only some of the adjustments that would need to be done.

Once all the code has been correctly modified, there could be further testing to the common key computation protocol as well as to the Kafka topic, to verify how it would handle an increasingly large number of identities. We assume that in such an environment we won't be able to overload Kafka server, the machine running the Python script will most likely be the bottleneck, as the number of concurrent threads will be limited. Although this standalone step won't be capable of further validating the architecture's ability to scale, it is a step in the right path.

11.2 Smart Contract development

The current iteration of SC used to register real and fake identities to the BC are rather simple. As described in section 9.2.4, they only post a key:$value$ object to the BC. It doesn't verify the validity of the values that are being posted. A possible next iteration to the SC would be to implement the business logic directly onto the SC.

Although this step adds considerable complexity to the overall SC development, it will ensure the validity of the posted Identities.

11.3 Parallelization

In the previous section, we identified the need to adapt the Python script to accommodate an arbitrary number of Identities. Unfortunately, we anticipate a potential bottleneck due to the limitations of the machine executing the Python script. To address this, we recommend distributing the processes across multiple machines.

By gaining access to a larger amount of machines, further testing to the common key computation with an ever-growing number of identities can be performed. This approach would also allow realistic testing to Kafka's performance in a scenario that mirrors real-world conditions. Ideally, the machines used for this purpose should not be overly powerful, but instead, mimic the computational and storage capabilities of small IoT devices.

This test represents a critical step towards further validating the proposed architecture. It's important to note, that this test may entail higher infrastructure costs compared to previous, less demanding tests.

11.4 Distribute Hyperledger Fabric Network

In the previous two sections we suggested possible improvements to further validate the architecture. However, in both cases the HFB network configuration of two organizations with one Peer Node each was kept the same, to simulate additional Identities, we suggested deploying a SC that publishes random complete Identities in the BC, the **Dummy Identity SC**.

The next step would be to do the exact same thing, but this time with real Identities that represent real Assets. To do so, the HFB network deployment files need to be modified to accommodate a custom number of organizations, each with an initial custom number of Assets. Each organization would Ideally be deployed in separate machines as well as different networks. Furthermore, each would need to run their own CA, as well as their own Orderer Peer, this is the correct way to deploy a production like HFB network. Following deployment, each Organization would use the **Register SC** described in section 9.2.4 to register their Asset's Identities, from there each Asset would run its own Process to compute the common secrete key.

12 Conclusion

Our work began with a central focus: addressing the pressing challenges of data ownership, authenticity, interoperability, and cyber-security that loom large with the rising adoption of DTs in Smart Cities in a decentralized manner.

To handle such challenges, we employed the Hyperledger Fabric Blockchain to fully register the identity of each urban asset, this Identity was a DID and its correspondent DID Document, thanks to the immutability of the BC we managed to enhance trust within the ecosystem. This was improved further by our use of Kafka topics, which introduced a mechanism for lightweight group messaging, ensuring communication was both efficient and scalable. Furthermore, our approach to encryption, enabled by the protocol binary tree Diffie-Hellman algorithm, allowed each Asset, equipped with its Private Key, and every other Asset's Public Key, to derive a common secret key. This pivotal key played a central role in symmetric encryption and decryption processes, safeguarding our communications. Ultimately, this system fills the main requirements of **Availability**, **Resilience**, **Efficiency**, **Confidentiality**, **Integrity**, **Provenance** and **Scalability**.

The results were illuminating. We successfully registered identities using SCs, demonstrating the robustness and functionality of our proposed framework. Yet, our framework isn't without its limitations. The first limitation we identified was the issue of key freshness, essential for ensuring more secure exchanges. Our strategy to solve this involved introducing random, non-real identities to add unpredictability to our common secret key. Furthermore, to ensure the confidentiality of business transactions, we showcased the use of additional channels within the HFB, indeed HFB allows us organizations to create an unlimited number of custom channels in the network. An additional limitation the impossibility of Identity Revocation, this will be a required functionality, as some Assets might be physically destroyed, or taken control of by malicious entities, some solutions were explored.

Looking forward, while our decentralized approach shows enormous promise, there may be scenarios where its strengths are magnified when paired with a second architecture that uses a centralized authority system. The choice between these two, we believe, will be shaped by the unique demands of specific use cases. There's still work to be done in this field, for the future, we suggest performing additional tests in more realistic urban environments, be it through registering more identities, more functionally capable smart contracts, or even by placing each organization in different networks.

In the closing stage of this research, we realize how critical additional research in this field is. As DTs become more prevalent components of our urban development, ensuring their authenticity, security, and seamless integration is mandatory. We've taken significant steps in setting a course towards achieving this goal, and it is our sincere hope that our work paves the way for future innovations, guiding our cities toward a future that is not only smarter but also secure and sustainable.

References

[1] Aidan Fuller, Zhong Fan, Charles Day, and Chris Barlow. Digital Twin: Enabling Technologies, Challenges and Open Research. *IEEE Access*, 8:108952–108971, 2020. Conference Name: IEEE Access.

[2] Yiwen Wu, Ke Zhang, and Yan Zhang. Digital Twin Networks: A Survey. *IEEE Internet of Things Journal*, 8(18):13789–13804, September 2021. Conference Name: IEEE Internet of Things Journal.

[3] Fei Tao, Meng Zhang, Yushan Liu, and A. Y. C. Nee. Digital twin driven prognostics and health management for complex equipment. *CIRP Annals*, 67(1):169–172, January 2018.

[4] Gernot Steindl and Wolfgang Kastner. Semantic Microservice Framework for Digital Twins. *Applied Sciences*, 11(12):5633, January 2021. Number: 12 Publisher: Multidisciplinary Digital Publishing Institute.

[5] Roberto Minerva. Digital Twin in the IoT Context: A Survey on Technical Features, Scenarios, and Architectural Models.

[6] Bruno Cruz and Murillo Dias. CRASHED BOEING 737-MAX: FATALITIES OR MALPRACTICE? 8:2615–2624, January 2020.

[7] Ehab Shahat, Chang T. Hyun, and Chunho Yeom. City Digital Twin Potentials: A Review and Research Agenda. *Sustainability*, 13(6):3386, January 2021. Number: 6 Publisher: Multidisciplinary Digital Publishing Institute.

[8] Fabian Dembski, Uwe Wössner, Mike Letzgus, Michael Ruddat, and Claudia Yamu. Urban Digital Twins for Smart Cities and Citizens: The Case Study of Herrenberg, Germany. *Sustainability*, 12(6):2307, January 2020. Number: 6 Publisher: Multidisciplinary Digital Publishing Institute.

[9] Božidar Radenković, Marijana Despotović-Zrakić, Zorica Bogdanović, Dušan Barać, Aleksandra Labus, and Tamara Naumović. A distributed IoT system for modelling dynamics in smart environments. In *2020 International Conference Engineering Technologies and Computer Science (EnT)*, pages 47–53, June 2020.

[10] Abigail Francisco, Neda Mohammadi, and John E. Taylor. Smart City Digital Twin–Enabled Energy Management: Toward Real-Time Urban Building Energy Benchmarking. *Journal of Management in Engineering*, 36(2):04019045, March 2020. Publisher: American Society of Civil Engineers.

[11] T. Nochta, L. Wan, J. M. Schooling, and A. K. Parlikad. A Socio-Technical Perspective on Urban Analytics: The Case of City-Scale Digital Twins. *Journal of Urban Technology*, 28(1-2):263–287, April 2021. Publisher: Routledge _eprint: https://doi.org/10.1080/10630732.2020.1798177.

[12] S. Q. Dou, H. H. Zhang, Y. Q. Zhao, A. M. Wang, Y. T. Xiong, and J. M. Zuo. RESEARCH ON CONSTRUCTION OF SPATIO-TEMPORAL DATA VISUALIZATION PLATFORM FOR GIS AND BIM FUSION. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLII-3/W10:555–563, February 2020.

[13] L. Wan, T. Nochta, and J. M. Schooling. Developing a City-Level Digital Twin ?Propositions and a Case Study. In *International Conference on Smart Infrastructure and Construction 2019 (ICSIC)*, Cambridge Centre for Smart Infrastructure & Construction, pages 187–194. ICE Publishing, January 2019.

[14] Siavash H. Khajavi, Naser Hossein Motlagh, Alireza Jaribion, Liss C. Werner, and Jan Holmström. Digital Twin: Vision, Benefits, Boundaries, and Creation for Buildings. *IEEE Access*, 7:147406–147419, 2019. Conference Name: IEEE Access.

[15] The Blue City Project, October 2022.

[16] Dessislava Petrova-Antonova and Sylvia Ilieva. Digital Twin Modeling of Smart Cities. In Tareq Ahram, Redha Taiar, Karine Langlois, and Arnaud Choplin, editors, *Human Interaction, Emerging Technologies and Future Applications III*, Advances in Intelligent Systems and Computing, pages 384–390, Cham, 2021. Springer International Publishing.

[17] Data Mesh Principles and Logical Architecture.

[18] The Path to Self-Sovereign Identity.

[19] Sanam Ghorbani Lyastani, Michael Schilling, Michaela Neumayr, Michael Backes, and Sven Bugiel. Is FIDO2 the Kingslayer of User Authentication? A Comparative Usability Study of FIDO2 Passwordless Authentication. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 268–285, May 2020. ISSN: 2375-1207.

[20] Annett Laube and Gerhard Hassenstein. Self-Sovereign Identities.

[21] Niclas Kannengießer, Sebastian Lins, Tobias Dehling, and Ali Sunyaev. Trade-offs between Distributed Ledger Technology Characteristics. *ACM Computing Surveys*, 53(2):42:1–42:37, May 2020.

[22] A Review of Distributed Ledger Technologies | SpringerLink.

[23] Tareq Ahram, Arman Sargolzaei, Saman Sargolzaei, Jeff Daniels, and Ben Amaba. Blockchain technology innovations. In *2017 IEEE Technology & Engineering Management Conference (TEMSCON)*, pages 137–141, June 2017.

[24] Zibin Zheng, Shaoan Xie, Hong-Ning Dai, Weili Chen, Xiangping Chen, Jian Weng, and Muhammad Imran. An overview on smart contracts: Challenges, advances and platforms. *Future Generation Computer Systems*, 105:475–491, April 2020.

[25] Akanksha Kaushik, Archana Choudhary, Chinmay Ektare, Deepti Thomas, and Syed Akram. Blockchain — Literature survey. In *2017 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*, pages 2145–2148, May 2017.

[26] Yuanyu Zhang, Shoji Kasahara, Yulong Shen, Xiaohong Jiang, and Jianxiong Wan. Smart Contract-Based Access Control for the Internet of Things. *IEEE Internet of Things Journal*, 6(2):1594–1605, April 2019. Conference Name: IEEE Internet of Things Journal.

[27] Du Mingxiao, Ma Xiaofeng, Zhang Zhe, Wang Xiangwei, and Chen Qijun. A review on consensus algorithm of blockchain. In *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 2567–2572, October 2017.

[28] Rebecca Yang, Ron Wakefield, Sainan Lyu, Sajani Jayasuriya, Fengling Han, Xun Yi, Xuechao Yang, Gayashan Amarasinghe, and Shiping Chen. Public and private blockchain in construction business process and information integration. *Automation in Construction*, 118:103276, October 2020.

[29] Nick Szabo. Formalizing and Securing Relationships on Public Networks. *First Monday*, September 1997.

[30] Konstantinos Christidis and Michael Devetsikiotis. Blockchains and Smart Contracts for the Internet of Things. *IEEE Access*, 4:2292–2303, 2016. Conference Name: IEEE Access.

[31] Bruno Rodrigues, Thomas Bocek, Andri Lareida, David Hausheer, Sina Rafati, and Burkhard Stiller. A Blockchain-Based Architecture for Collaborative DDoS Mitigation with Smart Contracts. In Daphne Tuncer, Robert Koch, Rémi Badonnel, and Burkhard Stiller, editors, *Security of Networks and Services in an All-Connected World*, volume 10356, pages 16–29. Springer International Publishing, Cham, 2017. Series Title: Lecture Notes in Computer Science.

[32] Changyu Dong, Yilei Wang, Amjad Aldweesh, Patrick McCorry, and Aad van Moorsel. Betrayal, Distrust, and Rationality: Smart Counter-Collusion Contracts for Verifiable Cloud Computing. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 211–227, New York, NY, USA, October 2017. Association for Computing Machinery.

[33] Ye Guo and Chen Liang. Blockchain application and outlook in the banking industry. *Financial Innovation*, 2(1):24, December 2016.

[34] Remy Remigius Zgraggen. Cyber Security Supervision in the Insurance Sector: Smart Contracts and Chosen Issues. In *2019 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*, pages 1–4, June 2019.

[35] Mohammad M. Bany Taha, Sivadon Chaisiri, and Ryan K. L. Ko. Trusted Tamper-Evident Data Provenance. In *2015 IEEE Trustcom/BigDataSE/ISPA*, volume 1, pages 646–653, August 2015.

[36] Xueping Liang, Sachin Shetty, Deepak Tosh, Charles Kamhoua, Kevin Kwiat, and Laurent Njilla. ProvChain: A Blockchain-Based Data Provenance Architecture in Cloud Environment with Enhanced Privacy and Availability. In *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 468–477, May 2017.

[37] Andreas Bogner, Mathieu Chanson, and Arne Meeuw. A Decentralised Sharing App running a Smart Contract on the Ethereum Blockchain. In *Proceedings of the 6th International Conference on the Internet of Things*, IoT'16, pages 177–178, New York, NY, USA, November 2016. Association for Computing Machinery.

[38] Patrick McCorry, Siamak F. Shahandashti, and Feng Hao. A Smart Contract for Boardroom Voting with Maximum Voter Privacy. In Aggelos Kiayias, editor, *Financial Cryptography and Data Security*, Lecture Notes in Computer Science, pages 357–375, Cham, 2017. Springer International Publishing.

[39] Affan Yasin and Lin Liu. An Online Identity and Smart Contract Management System. In *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, volume 2, pages 192–198, June 2016. ISSN: 0730-3157.

[40] Weiqin Zou, David Lo, Pavneet Singh Kochhar, Xuan-Bach Dinh Le, Xin Xia, Yang Feng, Zhenyu Chen, and Baowen Xu. Smart Contract Development: Challenges and Opportunities. *IEEE Transactions on Software Engineering*, 47(10):2084–2106, October 2021.

[41] Santiago Figueroa-Lorenzo, Javier Añorga Benito, and Saioa Arrizabalaga. Modbus Access Control System Based on SSI over Hyperledger Fabric Blockchain. *Sensors*, 21(16):5438, January 2021. Number: 16 Publisher: Multidisciplinary Digital Publishing Institute.

[42] Securing Emission Data of Smart Vehicles with Blockchain and Self-Sovereign Identities | IEEE Conference Publication | IEEE Xplore.

[43] Weidong Shen, Tianliang Hu, Chengrui Zhang, and Songhua Ma. Secure sharing of big digital twin data for smart manufacturing based on blockchain. *Journal of Manufacturing Systems*, 61:338–350, October 2021.

[44] HL Paper Hyperledger Overview - https://www.hyperledger.org/learn/white-papers.

[45] HL Whitepaper Introduction to Hyperledger - https://www.hyperledger.org/learn/white-papers.

[46] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, Srinivasan Muralidharan, Chet Murthy, Binh Nguyen, Manish Sethi, Gari Singh, Keith Smith, Alessandro Sorniotti, Chrysoula Stathakopoulou, Marko Vukolić, Sharon Weed Cocco, and Jason Yellick. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the Thirteenth EuroSys Conference*, EuroSys '18, pages 1–15, New York, NY, USA, April 2018. Association for Computing Machinery.

[47] Demystifying Hyperledger Fabric (1/3): Fabric Architecture, May 2019.

[48] Droople | How smart is your building's water network?

[49] IBM Documentation, April 2023.

[50] Elvric Trombert. *Microservice Architecture for Urban Digital Twin platform and its security implications*. PhD thesis, EPFL, 2023.

[51] Khin Me Me Thein. Apache Kafka: Next Generation Distributed Messaging System.

[52] Yongdae Kim, Adrian Perrig, and Gene Tsudik. Tree-based group key agreement. *ACM Transactions on Information and System Security*, 7(1):60–96, February 2004.

[53] A method for obtaining digital signatures and public-key cryptosystems | Communications of the ACM.

[54] Yong Wang, Byrav Ramamurthy, and Xukai Zou. The Performance of Elliptic Curve Based Group Diffie-Hellman Protocols for Secure Group Communication over Ad Hoc Networks. In *2006 IEEE International Conference on Communications*, volume 5, pages 2243–2248, June 2006. ISSN: 1938-1883.

[55] Katriel Cohn-Gordon, Cas Cremers, Luke Garratt, Jon Millican, and Kevin Milner. On Ends-to-Ends Encryption: Asynchronous Group Messaging with Strong Security Guarantees. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, pages 1802–1819, New York, NY, USA, October 2018. Association for Computing Machinery.

[56] Godson Michael D'silva, Azharuddin Khan, Gaurav, and Siddhesh Bari. Real-time processing of IoT events with historic data using Apache Kafka and Apache Spark with dashing framework. In *2017 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*, pages 1804–1809, May 2017.

[57] Ben Laurie. Certificate Transparency: Public, verifiable, append-only logs. *Queue*, 12(8):10–19, August 2014.

[58] Chun-An Lin and Chun-Feng Liao. User-Managed Access Delegation for Blockchain-driven IoT Services. In *2020 International Computer Symposium (ICS)*, pages 462–467, December 2020.

[59] Oracle VM VirtualBox.

[60] Ubuntu 22.04.2 LTS (Jammy Jellyfish).

[61] Getting Started - Install — hyperledger-fabricdocs main documentation.

[62] Hyperledger Explorer – Hyperledger Foundation.

[63] The Go Programming Language.

[64] bitnami/kafka - Docker Image | Docker Hub.

[65] Manas Pratim Bhattacharya, Pavol Zavarsky, and Sergey Butakov. Enhancing the Security and Privacy of Self-Sovereign Identities on Hyperledger Indy Blockchain. In *2020 International Symposium on Networks, Computers and Communications (ISNCC)*, pages 1–7, October 2020.

[66] Christian Gorenflo, Stephen Lee, Lukasz Golab, and Srinivasan Keshav. FastFabric: Scaling Hyperledger Fabric to 20,000 Transactions per Second. In *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 455–463, May 2019.

[67] Luca Greco, Pierluigi Ritrovato, and Fatos Xhafa. An edge-stream computing infrastructure for real-time analysis of wearable sensors data. *Future Generation Computer Systems*, 93:515–528, April 2019.

[68] Van-Dai Ta, Chuan-Ming Liu, and Goodwill Wandile Nkabinde. Big data stream computing in healthcare real-time analytics. In *2016 IEEE International Conference on Cloud Computing and Big Data Analysis (ICCCBDA)*, pages 37–42, July 2016.

[69] threading — Thread-based parallelism.