TÉCNICO LISBOA

InsureConnect: Blockchain and Digital Identity on the Insurance Market

João Eduardo Cardoso de Sousa Candeias Travassos

Thesis to obtain the Master of Science Degree in

Computer Science and Engineering

Supervisor: Prof. Miguel Nuno Dias Alves Pupo Correia

Examination Committee

Chairperson: Prof. Maria Luísa Torres Ribeiro Marques da Silva Coheur
Supervisor: Prof. Miguel Nuno Dias Alves Pupo Correia
Member of the Committee: Prof. Joaquim José de Castro Ferreira

November 2024

Declaration
I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Acknowledgments

First and foremost, I would like to express my deepest gratitude to my thesis advisor, Professor Miguel Pupo Correia, for his unwavering guidance, valuable insights, and continuous support throughout this research journey. His expertise and encouragement have been pivotal in helping me refine my ideas and complete this work.

I would like to thank my mother and father for their friendship, encouragement and caring over all these years, for always being there for me through thick and thin and without whom this project would not be possible. I would also like to thank my grandparents for their understanding and support throughout all these years.

Last but not least, to all my friends and colleagues that helped me grow as a person and were always there for me during the good and bad times in my life. Thank you.

Lastly, I would like to acknowledge INESC-ID for providing the financial support that made this research possible.

Thank you all for being an essential part of this journey.

This work was financially supported by Project Blockchain.PT - Decentralize Portugal with Blockchain Agenda (Project no 51, Call no 02/C05-i01.01/2022), funded by the Portuguese Recovery and Resillience Program (PPR), The Portuguese Republic and The European Union (EU) under the framework of Next Generation EU Program.

Abstract

This thesis presents InsureConnect, a solution for registering the insurance process for property damage in the aftermath of natural disasters, with a focus on enhancing transparency and trust in the insurance process. The system incorporates Self-Sovereign Identity (SSI) technologies, utilizing Decentralized Identifiers (DIDs) and Verifiable Credentials (VCs) to enable secure digital identities. These identities allow users to interact with the network and provide verifiable proof for specific operations within the insurance process.Leveraging satellite imagery and blockchain technology, InsureConnect can register both the as-is state of the land property at the moment of the insurance contract creation and the moment of a damage claim submission following a disaster. The proposed system introduces a novel approach applicable not only to natural disasters but also to diverse insurance markets.

The architectural framework of InsureConnect incorporates a blockchain, Hyperledger Fabric, along with the Interplanetary File System (IPFS) for decentralized image storage. The system defines distinct roles for Clients, Insurance Companies, Auditors, and a Higher Authority, each granted specific permissions within the blockchain network. The selection of non-related entities, such as Instituto dos Registos e do Notariado (IRN) and Associação Portuguesa de Seguradores (APS), aims to ensure the system's integrity.

Operational procedures involve a desktop application, where insurance companies create and update insurance contracts, clients and insurance companies create and update property damage claims, and auditors read assigned insurance contracts and claims. Auditors verify transactions using a read-only access approach.

The integration of decentralized identifiers, verifiable credentials, and Hyperledger Fabric smart contracts ensures the authenticity and efficiency of the claims process. This thesis contributes to the discourse on blockchain applications integrated with SSI in insurance and disaster man-

agement, offering a comprehensive solution with implications for broader insurance markets.

Keywords

Blockchain technology, Insurance contracts, Verifiable credentials

Resumo

Esta tese apresenta uma solução para o registo do processo de seguros de danos materiais após desastres naturais, com foco em melhorar a transparência e a confiança no processo de seguros. O sistema incorpora tecnologias de Identidade Auto-Soberana (SSI), utilizando Identificadores Descentralizados (DIDs) e Credenciais Verificáveis (VCs) para permitir identidades digitais seguras. Estas identidades permitem que os utilizadores interajam com a rede e forneçam provas verificáveis para operações específicas dentro do processo de seguros.

Aproveitando imagens de satélite e tecnologia blockchain, o sistema consegue registar tanto o estado atual da propriedade no momento da criação do contrato de seguro, como o momento da submissão de um pedido de indemnização após um desastre. O sistema proposto introduz uma abordagem inovadora aplicável não só a desastres naturais, mas também a diversos mercados de seguros.

A estrutura arquitetónica incorpora uma blockchain, o Hyperledger Fabric, juntamente com o Sistema de Ficheiros Interplanetário (IPFS) para armazenamento descentralizado de imagens. O sistema define papéis distintos para Clientes, Companhias de Seguros, Auditores e uma Autoridade Superior, cada um com permissões específicas dentro da rede blockchain. A escolha de entidades independentes, como o Instituto dos Registos e do Notariado (IRN) e a Associação Portuguesa de Seguradores (APS), visa garantir a integridade do sistema.

Os procedimentos operacionais envolvem uma aplicação de desktop, onde as companhias de seguros criam e atualizam contratos de seguros, clientes e companhias de seguros criam e atualizam pedidos de indemnização por danos em propriedades, e auditores lêem os contratos e pedidos de indemnização atribuídos. Os auditores verificam as transações através de um acesso apenas de leitura.

A integração de identificadores descentralizados, credenciais verificáveis e contratos inteligentes do Hyperledger Fabric garante a autenticidade e a eficiência do processo de reclamações. Esta tese contribui para o debate sobre a aplicação de blockchain integrada com SSI em seguros e gestão de desastres, oferecendo uma solução abrangente com implicações para

mercados de seguros mais amplos.

Palavras Chave

Tecnologia Blockchain; Contractos de Seguro; Credenciais verificáveis

Contents

List of Figures

List of Tables

Listings

Acronyms

CA	Certificate Authority
DApp	Decentralized Application
DID	Decentralized Identifier
DLT	Distributed Ledger Technology
DFS	Distributed File System
DHT	Distributed Hash Table
EBSI	European Blockchain Services Infrastructure
EOA	Externally Owned Account
IaaS	Infrastructure as a Service
IPFS	Interplanetary File System
IRN	Instituto dos Registos e Notariado
BUPi	Balcão Único do Prédio
MSP	Membership Service Provider
OSN	Ordering Service Node
PKI	Public Key Infrastructure
POW	Proof-of-Work
POS	Proof-of-Stake
SSI	Self Sovereign Identity
VC	Verifiable Credential
TAO	Trusted Accreditation Organisation
TI	Trusted Issuer

1

Introduction

Contents

Over the past years, blockchain technology has gained much traction and a vast interest due to the variety of its possible applications. It was first proposed in 2009 by *Satoshi Nakamoto* in the *Bitcoin's* whitepaper [Nakamoto, 2009]. It was a revolutionary peer-to-peer distributed ledger with the objective of eliminating the need for a trusted third party in online transactions.

With the evolution of the technology and the raising popularity of Bitcoin, the idea of blockchains began taking different perspectives, people starting to take notice that it could be used in other areas. In particular, it started to become applied in more restricted environments, where trust was not assured between all the parties, leading to private and permissioned blockchains [Peck, 2017]. As a consequence, the scope of the technology widened to a much more ample variety of different areas such as real estate. People started building distributed applications with this technology as its backbone. These are now called Decentralized Applications (DApps).

Amidst these advancements, the concept of Self Sovereign Identity (SSI) emerged as a groundbreaking approach to identity management. SSI enables individuals to control and manage their personal data without relying on centralized authorities, addressing issues such as privacy, data breaches, and the risk of identity theft. Utilizing Decentralized Identifiers (DIDs)

and Verifiable Credentials (VCs), SSI leverages blockchain technology to provide a decentralized, immutable, and secure means of verifying identities. This innovation allows users to selectively disclose only the necessary information, ensuring privacy while maintaining trust between parties. SSI has become an essential tool in contexts requiring secure, decentralized identity verification, laying the foundation for its integration into solutions like the housing market, where trust and efficiency lack and are paramount.

Natural disasters, such as floods, earthquakes, and typhoons, pose significant challenges to individuals and communities, often resulting in property damage and financial losses. Responding to these challenges requires innovative approaches to streamline the claims process and ensure trust between the affected individuals and insurance companies. In this thesis proposal, a comprehensive solution, named InsureConnect, is proposed for registering entities in the form of decentralized identifier documents, insurance contracts and property damage claims in the aftermath of natural disasters, leveraging satellite images, self sovereign identity (SSI) and blockchain technologies.

The proposed system, InsureConnect, is designed to enhance the reliability and transparency of the insurance process by implementing a blockchain-based framework. The use of blockchain, a decentralized and tamper-resistant ledger, aims to establish trust between the clients seeking financial assistance and the insurance companies responsible for evaluating and processing claims. The solution will be not only applicable to natural disaster scenarios but will also hold potential for adaptation to various markets, such as car crash insurances, thereby serving as a versatile model for insurance-related transactions.

The proposed solution will be developed with a permissioned blockchain developed in Hyperledger Fabric. To workout the solution, DIDs and VCs will be used to verify the clients identities and proofs of property ownership, and IPFS - Interplanetary File System [Benet, 2014], will be used as a way to store satellite images as to not overload the blockchain itself. This way, InsureConnect boasts a fully decentralised system.

There are three main services InsureConnect will have by the end: User registration as DID Documents, Insurance Contracts registration and Damage Claims Registration. Firstly, every entity in the system must properly registered to be able to utilize it. This will be achieved with the use of DID Documents as mentioned before. After that, clients and insurance companies will be able to establish an insurance contract between them both with images of the current state of the property. The insurance contract will become valid once the two parties sign the contract and add their signature. Lastly, when a client suffers property damages, they will submit they damage claims in the systems. These will be updated by the insurance company along different status to reflect the handling process of the claim.

1.1 Objectives

In the end, I expect this work to have the following contributions:

- Create a new system based on blockchain and SSI technology for insurance clients and insurance companies to come to an agreement on damage claims with a full trust system
- Implement and deploy a prototype of the blockchain system
- Evaluate the performance and effectiveness of the proposed solution

1.2 Overview

The remainder of this document is structured to provide a comprehensive understanding of the key components related to the subject of the thesis. It is structured as follows. Section 2 introduces the key concepts related to the subject of the thesis. Section 3 introduces related research and existing technology that we can use or take inspiration from. Section 4 presents the developed solution with the architecture of the system, context and procedures. Section 5 details and showcases the performed testing and subsequent results analysis. Section 6 provides the final thoughts and concludes the document.

Disclaimer: Any mentioned public organizations were not involved in the development of this thesis prototype. These were just mentioned for illustrative porpoises to introduce the presented concepts.

2

Background

Contents

In this chapter we present work from different domains that is relevant for my proposal, while also giving an introduction to each of them. I start by providing an overview of the concept of Blockchain in Section 2.1. In Section 2.2, I briefly introduce two related Digital Identity technologies and explain its major features. In Section 2.3, I provide insight on a peer-to-peer content delivery network without a central server. Finally, in Section 2.4 I briefly present and explain the European Blockchain Services Infrastructure initiative.

2.1 Blockchain

Over the last few years *Blockchain* has gained much notoriety, mostly due to the appearance of cryptocurrencies and more specifically due to Bitcoin [Nofer et al., 2017, Helliar et al., 2020, Baliga, 2017, Nakamoto, 2009], written by Satoshi Nakamoto. As blockchain was its fundamental underlying technology, its popularity grew as an innovative technology, expanding

from just the cryptocurrency sphere into other industries and use-cases. Be it for good or bad reasons, it became extraordinarily popular, but: “What is it exactly?”.

2.1.1 Blockchain Innerworkings

A blockchain is a transparent immutable distributed and decentralized append-only ledger usually constituted by multiple party’s that do not trust each other [Dinh et al., 2018]. A ledger is a collection of transactions. Its core component are transactions, that grouped together forming *blocks*, represented in Figure 2.1. Each block cryptographically points to the previous one, forming a conceptual chain, hence the name *blockchain*. Blockchains are distributed by nature with no central point of failure. They can be viewed, in a simplified manner, as tamper-resistant databases. Both its architecture and ownership are distributed. Every participant has its own copy of the ledger and its updates are transparently agreed upon with all participants of the network by a process called *consensus*. This way, problems such as destruction or loss of records, hardware, location related attacks, and trust issues no longer existent. It is very difficult to attack a blockchain and succeed to have an impact on its well-being.

As mentioned before, in a blockchain, the blocks are cryptographically connected by pointing to the previous one. Blocks contain a block header and block data. The header contains metadata for the block, such as, block number, hash of the previous block’s header, a hash representation of the block data (i.e. Merkle tree), a timestamp, size of the block. The block data contains a list of validated and authentic transactions which have been submitted to network, ledger events and some other optional data. It is in the blocks header that the chaining comes to life. With the hash digest of the previous block’s header engraved in the own header, an immutable linked system is born. By pointing to the previous block’s hash, it is near impossible to ever modify them, since altering a past block would compromise the whole chain. By altering a past block, its hash changes, therefore changing the content of all the subsequent blocks. This is infeasible in a distributed system as blockchain as it would require 51% (majority) of the network to agree to the change and a tremendous computational power. Even thought this is true, there are scenarios where the blockchain can split, branching out and eventually needing to delete one of the branches. This case will be discussed further bellow.

In order to build blocks, transactions are needed. Transactions are data transmissions across the network of computers in a blockchain system. While primarily transactions really represent a transfer of the cryptocurrency between network users, transactions can be more generally used to just transfer data as a way to permanently post it on the blockchain. While the data comprised in a transaction can vary greatly from one blockchain to another, the mechanism for transacting is largely the same. A blockchain network user sends data to the blockchain network.

In order for users to trust each other transactions and avoid frauds, blockchains employ cryptographic hash functions. This is one of the most important components in the blockchain

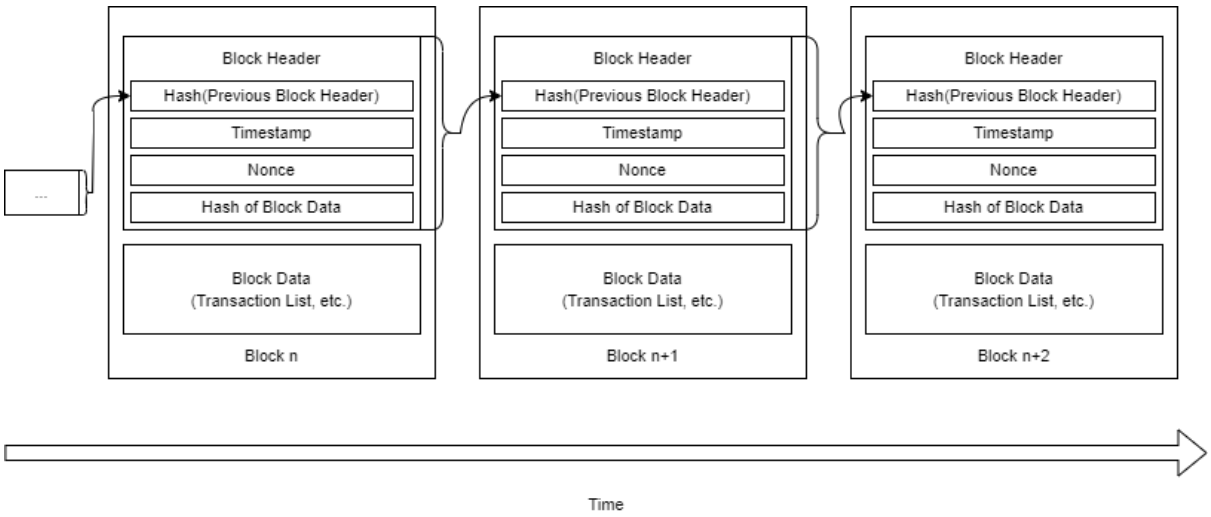


Figure 2.1: Generic Chain of Blocks

technology. Hashing is a method of applying a cryptographic hash function to data, which calculates an unique output, also known as *message digest* or, simply, *digest*. It allows participants to independently take input data, hash that data, and derive the same result, proving that there was no change in the data. This way, other participants only have to compare the data in the transaction with the hash provided in its header to verify its integrity. Not only this but, for transactions to be valid and accepted by other users, they need to be cryptographically signed by the issuer. To achieve this, blockchain technology uses asymmetric cryptography. Asymmetric cryptography uses a pair of keys, one public and the other private, that are mathematically related to each other. If one of the keys encrypt a portion of data, only its pair can decrypt it. The public key is made public for everyone to see and use, while the private one has to be kept private at all costs. If the private key is made public, that key pair no longer assures authenticity since anyone could use it to impersonate the original owner of the key-set. Not only that but any digital assets associated with that key are also lost since it is computationally infeasible to regenerate the same private key and the attacker will have full access and control over the digital assets. With some blockchain networks, user must securely store their key-pairs in software often referred to as *wallet*.

Asymmetric cryptography enables a trust relationship between users who do not know or trust one another, because it provides a mechanism to verify the authenticity of transactions. To do this, transactions are *digitally signed*. What this is means, is that they are encrypted with the users private key such that anyone with the public key can decrypt it, thus proving it really was the user who signed the transaction. A transaction is only valid if they are cryptographically signed.

There are two main types of blockchain in regard to their access scope: *public* and *private* - and more two ways of sub-categorizing them in regards to their privilege scope: *permissionless* and *permissioned*. The mains differences between public and private is how they accept new users to join it and what privileges (actions) each participant is granted [Peck, 2017]. Public

permissionless blockchains can be access by anyone at any time, while their identity's remain a secret. These are the types of blockchains behind most crypto-currencies nowadays, such as *Bitcoin* [Nakamoto, 2009] and *Ethereum* [Buterin et al., 2014]. This means that anyone can create their own personal address and begin interacting with the network as they please. In these blockchains there are usually many users trying to get their blocks to be accepted and propagated to get a reward, typically a small amount of cryptocurrency. Everyone in the blockchain is most likely financially motivated and distrusts everyone else. In this scenario, why would anyone propagate other users blocks? What is stopping every participant from only trying to propagate their own blocks? To solve these issues, blockchains employ *consensus models*. The consensus model will determine who publishes the next block.

The first consensus model within the context of blockchain technology, designated as Proof-of-Work (POW), was initially presented in the Bitcoin whitepaper by Satoshi Nakamoto [Nakamoto, 2009]. On the Bitcoin network, blocks are built by particular nodes called *miners*. All these miners are, as mentioned before, competing to get their block published before any other miner can publish theirs. In Bitcoin's consensus, a miner is allowed to publish their block after they solve a computationally intensive puzzle. The solution to this puzzle acts as a proof to the other miners that the one publishing the block has performed the necessary work to solve it. This puzzle is designed in a way such that solving it is hard but its solution enables all the participants to quickly check the blocks validity. A common puzzle is to require that the hash digest of the block has a certain number of leading zeros. The difficulty of the puzzle gets adjusted over time. Solving this puzzle requires many tries and, more importantly, a lot of resources. It is important that a lot of resources are consumed to discourage byzantine behaviour and to improve the overall security of the blockchain. The use of a computationally intensive puzzle helps to combat the *Sybil Attack* - a computer security attack where an attacker can create many nodes to gain influence and exert control. After a participant solves the puzzle, it is broadcasted to the other participants in the network, who will verify if the puzzle solution is valid. If so, the new block is added to their copy of the blockchain and the user who published the block will get the reward for doing so. All other miners will discard their current work and start to solve the next blocks puzzle to try get their reward. The reward usually is the transaction fee payed by the issuer of the transaction. This consensus model, however, has brought some controversy to the Bitcoin network, since the amount of energy consumed from every miner competing against each other is on par with a small country [Vranken, 2017].

Private and permissioned blockchains, on the other hand, are blockchains only accessible to authorized individuals. Due to this, there usually is no anonymity regarding the users identity. This does not imply that trust among the members is assured. To publish blocks and maintain the network, users must be granted authorization to do so by some higher authority (centralized or decentralized).

Private blockchains, on the other hand, are blockchains accessible only to selected participants. Because of that, the participants identities are known. That does not imply that trust

among them is assured. The most popular private blockchain is Hyperledger Fabric. Different users can be given different access authorizations, such as to read and write. “Permissioned blockchain networks are often deployed for a group of organizations and individuals, typically referred to as a *consortium* [Yaga et al., 2019]. The users partaking in these blockchains are usually striving for the same goal, so there is no need for an incentive reward for publishing new blocks. Not only that, but the consensus models between the users can also be less computationally expensive and faster. Since users get their permissions from a higher authority, if any user decides to misbehave, or try to undermine the whole process, their privileges can be revoked. These blockchains seem to go against the whole point of its technology, since decentralization starts to get reduced and and cases of censorship become possible, but sometimes that may not matter for a group who might want, for example, a shared immutable database without any help from a trusted third party.

When dealing with distributed systems on a large scale, it is just inevitable that eventually some blocks will be behind on information or have an alternative version of information, be it because of network latency or simply because of the distance between groups of nodes, since they are distributed across the globe. Blockchains are no exception to this situation and it can lead to the blockchain branching out and creating a ledger conflict, that needs to be quickly resolved to regain the blockchain’s consistency. Let’s say, for example, that two nodes solve a puzzle and broadcast their block to become the next published one and, because of the reasons mentioned before, both get accepted. This creates a situation where we can have two versions of the same blockchain. This can lead to problems such as *double-spending* - a typical problem found with digital currencies where after making a transaction an attacker quickly tries to spend the same money again before it was verified and validated into the system. The way most blockchain networks solve this problem when faced with it, consists in choosing the longer branched chain as the “official” one. The transactions on the smaller chain then become obsolete and return to the pending transaction pool to try to be included in another block later. Since the possibility of a block being overwritten exists, a transaction is not usually confirmed until several additional blocks have been created on top of the one containing the relevant transaction. Some blockchains decide to lock older blocks in the blockchain ledger as a checkpoint so that extreme scenarios, such as, lots of blocks getting undone cannot happen.

Eventually, everything needs to be updated or changed. Changes to a blockchain network’s protocol, or data structures, are called *forks*. There are two types of forks, *soft* forks and *hard* forks. Forks lie in the soft category when its changes are backwards compatible with nodes that have not been updated yet. Forks lie in the hard category when its changes are not compatible with nodes that have not been updated yet. This causes nodes with old software to reject blocks that have been produced by updated nodes. Besides slowing down the overall network process, this situation can lead to a worrisome branching in the blockhains network, creating two different versions of the same blockchain that cannot be merged without updating the out-of-date nodes. All publishing nodes need to update their protocol, since nodes on different hard

fork version cannot interact with each other. If the blockchain network we are participating in uses a cryptocurrency, then users will have independent currency on both forks. Activity must switch over to the new chain. This problem, even though it applies to blockchain networks in general, it is more prominent in public permissionless networks, since private permissioned can mitigate this issue requiring software updates to its users.

2.1.2 Ethereum

Ethereum, often described as the *world computer* [mas,], was introduced in Vitalik Buterin's paper [Buterin et al., 2014]. This new implementation of the distributed ledger technology addressed a lot Bitcoin limitations, one of which is that it is not Turing-complete [Vujičić et al., 2018]. Ethereum is an open source, globally decentralized computing infrastructure that uses a blockchain to synchronize and store the system's state changes, along with a cryptocurrency called *Ether* (divided in Wei; 1Ether = 10^{18}Wei) to meter and constrain execution resource costs [mas,]. With the Turing-complete built-in programming language, Ethereum is able to provide an abstraction layer for any participant to write smart contracts dictating their own rules for ownership, format of transactions and state transition functions [Buterin et al., 2014].

Ethereum supports two types of accounts, *Externally Owned Accounts (EOAs)* and *Contract Accounts*. EOAs are used to store an user's funds in Wei. These accounts are address by public keys and access to them is granted by showing the ownership of the corresponding private key [Chen et al., 2020]. Contract accounts, as the name suggests, are accounts linked to a smart contract (explained further bellow). EOA or contract accounts have a dynamic state defined by: nonce, balance, storage root and code hash. The nonce is the amount of transactions initiated by the owner of an EOA account or the number of contracts created by the contract account. The balance is the amount of Wei the account owns. The storage root is the hash of the root of the contract account's storage data structure (not applicable to EOAs). The code hash is the hash value of a contract account's bytecode (not applicable to EOAs) [Chen et al., 2020].

Smart contracts, even thought they are neither smart nor contracts, are executable programs written by any participant with built-in economic functions. These contracts function as agreements between mutually distrusting parties. After being compiled, these contracts are deployed on the Ethereum platform using a special contract creation transaction. Each contract has its own address, and in order to be executed, the interested user has to send funds to it [mas,]. Contracts consume what's called as *Gas*. Gas, as the name suggests, is the "fuel" for computation instructions executed in the Ethereum network. The more computationally exhausting a transaction is, the more it will consume. The idea behind it, is to make users pay for the computational costs necessary to approve, create and execute their transactions [Pierro and Rocha, 2019]. This way, users have to be very careful with the programs they want to execute, since creating, for example, an infinite loop would result in undesired consumption of

Gas. Not only that but, guaranteeing the correct execution of smart-contracts is a necessity, since avoiding to do so, could leave room for an attacker to take advantage of a vulnerability and tamper with the contract's execution, for example, redirecting money to himself. Gas is bought using Ether.

For anything to move, transactions need to happen. Transactions are cryptographically signed messages, originated by an EOA, transmitted by the Ethereum network, and recorded on the Ethereum blockchain [mas, , Vujičić et al., 2018]. Each transaction is defined by: the recipient, the amount of ether, the nonce, some data (optional), the STARTGAS and the GASPRICE.

Since the transaction is signed, there is no need for a sender field. The nonce is a sequence number to prevent message replay (see more in 'The Transaction Nonce', chap 6, [mas,]). The recipient is the Ethereum destination address. The ether is the amount of Ether, in Wei, to be sent to the destination. The STARTGAS denotes maximum amount of gas the sender is willing to buy for this transaction. The GASPRICE denotes the amount of ether, in Wei, the sender is willing to pay for each unit of gas.

The execution of a smart contract only runs as long as there is Gas to be consumed. When the Gas "runs out", the miner that was executing the code stops its execution and gets payed the transaction fees converted to Ether [Pierro and Rocha, 2019]. Miners will tend to choose processing transactions with higher GASPRICEs because they will be more rewarded in doing so [Vujičić et al., 2018]. Not only that, miners also have to acknowledge some minimal GASPRICE under which they refuse to process a transaction. Because of this factors, the sender has to, not only take care in the code he wants to execute (make it as efficient as possible) because of the STARTGAS (max amount of gas he is willing to pay), but also has to wisely evaluate and choose what is the best GASPRICE so his transaction gets processed.

A transaction flows like this: The Ethereum state transition function verifies the correctness of the transaction checking if the senders signature is valid and the transaction nonce matches the current nonce of sender. If everything checks out, then the transaction fee is computed as STARTGAS*GASPRICE. This value is then subtracted from the user senders account and his nonce is incremented. The fee will then be payed per byte in the transaction and the requested amount of ether is transferred to the recipient. If the recipient account does not exist, then it is created, and if it is a contract, then the contract's code is executed. If the sender does not have enough Ether for transaction or the STARTGAS limit is reached, all state changes, except the fees payment, are reversed [Vujičić et al., 2018].

To decide who publishes the next block, Ethereum utilizes the Proof-of-Stake (POS) consensus model. The main idea behind is model is that the more stake a user has invested into the system, the more likely they will want the system to succeed, so the less likely they will want to subvert it. *Stake* is the amount of cryptocurrency the user has invested into the system. Once staked, the assets cannot be spent. Proof-of-Stake networks use the amount of stake a user has as a determining factor for publishing new blocks, in other words, the more assets a user has invested into the network, the more likely he is to get elected to publish the

next block. With a model like this, the need for resource intensive computations disappears. There are many methods to choose who publishes the next block, such as, random selection of staked users, multi-round voting, but one question is immediately raised: “Can’t one person or an unknown party stake so much that it can take exert control over the network?”. No, because after the choice of the block publisher is through, users know as validators, vote for one node to become the publishing node for a specific amount of time. Ethereum is divided in epochs (6.4 minutes). Each epoch is divided into 32 slots. For each slot, a committee of at least 128 validators gets chosen to choose the publisher node of that slot. To become a validator candidate, a node needs to deposit 32ETH into a specific contract. Once a new slot begins, one random validator proposes a new block of transactions. The remaining validators vote on the proposal and attest to the transactions. Once the majority, has agreed, the block is added to the blockchain and the proposer of the block receives a variable amount of ETH. To penalize validators for bad behaviours such as going offline, proposing multiple blocks, or submitting contradictory votes, there is a system called *slashing*, in which validators loose a percentage of their staked ETH. The slashed amount depends on the amount of validators being slashed, otherwise known as the *correlation penalty*.

Contrary to Bitcoin, in which a transaction is never considered permanent, in Ethereum finally is assured through checkpoints. The first block in each epoch is a checkpoint. Stake owners then vote on pairs of checkpoints that are considered valid. Once a checkpoint gains a supermajority vote (two-thirds of the total , it becomes justified. When its child checkpoint gets justified, it is upgraded to finalized and all previous epochs are also finalized. The difference between justified and finalized depends simply on where it sits in the timeline. To prevent an attacker to vote against finalization with one third of the staked ETH, the *inactivity leak* exists. If the chain does not reach finality for more than four epochs, the validators voting against it will see their staked assets reduced, allowing honest voters to finalize the chain.

2.1.3 Hyperledger Fabric

The Hyperledger Project [hyp, 2024], initiated by the Linux Foundation, aims to develop a collaborative ecosystem for blockchain development through open-source technologies. Rather than being a cryptocurrency, the project serves as an open hub where software developers and organizations work together to build robust, enterprise-grade blockchain frameworks. By bringing together diverse contributors, Hyperledger enables these frameworks to evolve from initial ideas to fully-fledged solutions capable of commercial use. [Dhillon et al., 2017]. Under the Hyperledger projects there are 8 project incubating, five frameworks: Fabric, Iroha, Sawtooth, Burron and Indy, and three development tools: Composer, Explorer and Cello. We will focus on Hyperledger Fabric.

Hyperledger Fabric is a modular implementation of the blockchain architecture with plug-and-play features including plugable consensus, unique membership services for different user

roles and transaction functions [Androulaki et al., 2018,Dhillon et al., 2017]. Hyperledger Fabric works as a private and permissioned blockchain that mandates members of the network to log in via a valid membership service provider. It also utilizes a Public Key Infrastructure (PKI) to create cryptographic certificates. Fabric is composed of three particular features: private channels, chaincodes and types of nodes.

The Fabric framework offers the capability to concurrently operate multiple blockchains. Each of these distinct blockchains is referred to as a *channel*. These channels can manage the participating members, thereby facilitating the establishment of private channels in which only specific individuals or entities are granted access. Importantly, each channel maintains its own ledger and a distinct set of rules governing its operations. Nevertheless, all of these channels share a common *ordering service* responsible for the filtration and orderly processing of their respective transactions [Dhillon et al., 2017]. Each of these channels has its own set of configurations and these are persisted in special configuration blocks. Each of the logical blockchains starts out in a configuration block denominated *genesis block*, which hold the following configurations: (1) definition of the Membership Service Provider (explained further bellow) for participating nodes, (2) the network addresses of the *Ordering Service Nodes (OSNs)*, (3) shared configuration for consensus implementation and the ordering service, (4) rules regarding access to ordering service operations and (5) rules regarding how each channel may be modified. These configurations may be updated using a *channel configuration transaction*. When used, the ordering service will evaluate whether the update is valid and generate a new configuration block. The peers receiving this block will check if the update is authorized based on the current configuration. The configurations will be updated if everything checks out. [Dhillon et al., 2017]

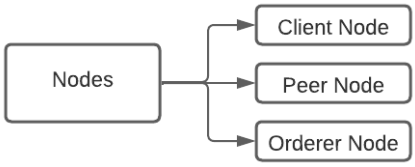
As in Ethereum [Buterin et al., 2014], Fabric runs smart-contracts which are called *Chaincode*. These chaincodes can either be normal, with executable programs written by any (un)trusted developer or, special, called *System Chaincodes*, used to configure the respective channel they are in. Normal chaincodes have a distinct feature, they are not bound to domain-specific languages. They are written in standard general-purpose programming languages, without systematic dependency on a native cryptocurrency, standing in contrast to existing blockchain technologies [Androulaki et al., 2018]. This is possible because the environment chaincodes execute in is pseudo-detached from its peer. Every user level, or application chaincode, runs in a separate process within a Docker container environment, isolating the chaincodes from each other and from the peer. This allows, not only to add plugins to support new programming languages, but also to simplify the management of the lifecycle of chaincodes (i.e, starting, stopping, or aborting). Contrary to normal chaincodes, system chaincodes execute directly in the peer process. It is important to note that, chaincodes have a set of endorsers allowed to provide *endorsements* for their execution. This set of endorsers is implicitly specified via *endorsement policy*. The *Endorsement Policy* acts as a static library for transaction and only administrators may have a permission to modify it [Dhillon et al., 2017].

Chaincodes can only be interacted with through transactions. There are two types of transactions: *Deploy Transactions* which allow new chaincodes to be created, and *Invoke Transations* which allow a chaincode to be executed on the blockchain [Dhillon et al., 2017].

For any transaction to happen in Fabric, transactions first need to be endorsed, then ordered and finally validated. Fabric departs radically from the standard order-execute paradigm where transactions are firstly ordered and propagated to all peers, and secondly, sequentially executed by all peers. Fabric follows a novel execute-order-validate paradigm [Dhillon et al., 2017]. Let's take a look at the existing three roles in order for this architecture to function properly.

Every node in Fabric can take up one of three roles(represented in Figure 2.2):

- Clients - Submit transaction proposals for execution and help orchestrate the execution phase, as well as, broadcast transactions for ordering. The execution phases will be presented further bellow.
- Peers - Maintain the distributed-ledger, execute transactions proposals and validate them. It must be noted that not all peers execute all transaction proposals, only a subset of them do based on the chaincode specification (further explanation bellow).
- Orderers - These are the ones that, collectively, form the ordering service. These nodes establish the total order of all transactions, its dependencies and cryptography signatures of the endorsing peers. Orderers must be completely unaware of the application state and do not participate in validation or execution of transactions.



.

Figure 2.2: Nodes in the Hyperledger Fabric network. From [**?**]

Since Fabric is a private and permissioned blockchain and its members must be authenticated, it uses a membership service. The *Membership Service Provider (MSP)* maintains the identities of all nodes in the system (clients, peers and OSNs) and is responsible for issuing node credentials that are used for authentication and authorization purposes. The membership service comprises a component at each node, where it may authenticate transactions, verify the integrity of transactions, sign and validate endorsements, and authenticate other blockchain operations. The default MSP handles standard PKI method for authentication of digital signatures and can accommodate Certificate Authorities (CAs). It even provides its own certification authority called *Fabric-CA*. When setting up a blockchain network, Fabric provides two modes: one offline and another online. In the offline mode credentials are generated by

a CA and distributed to all nodes. It is important to note that peers and orderers can only be registered in offline mode. The online mode serves for Fabric-CA to provide enrolling clients with cryptographic credentials. MSP allows for identity federation, for example, if multiple organizations are working in the same blockchain, each of them issues identities to its members and every peer will be able to recognize which peer belongs to which organization [Dhillon et al., 2017]

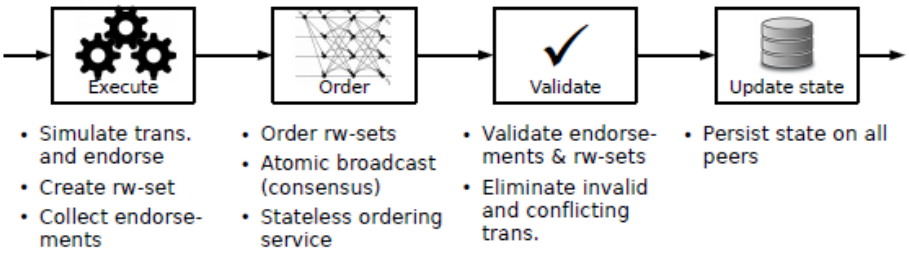


Figure 2.3: Execute-Order-Validate architecture of Fabric. From [Dhillon et al., 2017]

There are three phases that compose the execute-order-validation architecture: Execution, Ordering and Validation. Starting off in the Execution phase, clients sign and send a transaction proposal to the endorsers specified in the chaincode. Each proposal contains the identity of the submitting client, the transaction payload, parameters, identifier of the chaincode, a nonce and a transaction identifier (derived from the client identifier and nonce). Each endorser will execute the proposal in the specified chaincode which is run in a Docker container, isolated from the endorser process. After execution against the endorser's local blockchain state (simulation) the result will not be persisted to the ledger, instead, each endorser will produce a *writeset*, consisting of the updates produced by the simulation, and a *readset*, representing the version dependencies of the proposal simulation, cryptographically sign it (*endorsement*) and send it back to the client as a proposal response. After the client has finished collecting enough endorsements with the same results, until it can satisfy the endorsement policy of the chaincode, it proceeds to create the transaction and passes it onto the ordering service. The Ordering phase starts when the ordering service, upon receiving the transaction, containing transaction payload, transaction metadata and a set of endorsements, establishes a consensus on the total order of all submitted transactions per channel. Not only that, but it also batches multiple transactions into *blocks* and outputs a hash-chained sequence of blocks containing the transactions. If there ratio of peer: OSNs is disproportionate, Fabric can be configured to use a built-in *gossip service* for disseminating delivered blocks from the ordering service to all peers. From the output broadcast of the ordering service, the Validate Phase starts. Blocks will be delivered to the peers and block-by-block will enter a three sequential step validation consisting of an *endorsement policy evaluation*, *read-write conflict check* and *ledger-update-phase*. Even if a transaction is considered as invalid, it is still recorded in the ledger. In the *Endorsement Policy Evaluation* step, which occurs in parallel for all the transactions in the block, each trans-

action has its endorsements evaluated with respect to the endorsement policy configured in the chaincode. If a transaction fails this check, it is marked as invalid and its effects are disregarded. In the *Read-Write Conflict Check* step, which is done sequentially for all transactions in the block, each transaction's version keys in the *readset* will be compared to those in the current state of the ledger. If the version do not match, the transaction will be marked as invalid and its effects are disregarded. In the *Legder update phase*, the last phase, the block will be appended to the ledger and the blockchain state is updated [Dhillon et al., 2017]. The whole process is shown very succinctly in Figure 2.3 and in more depth in Figure 2.4.

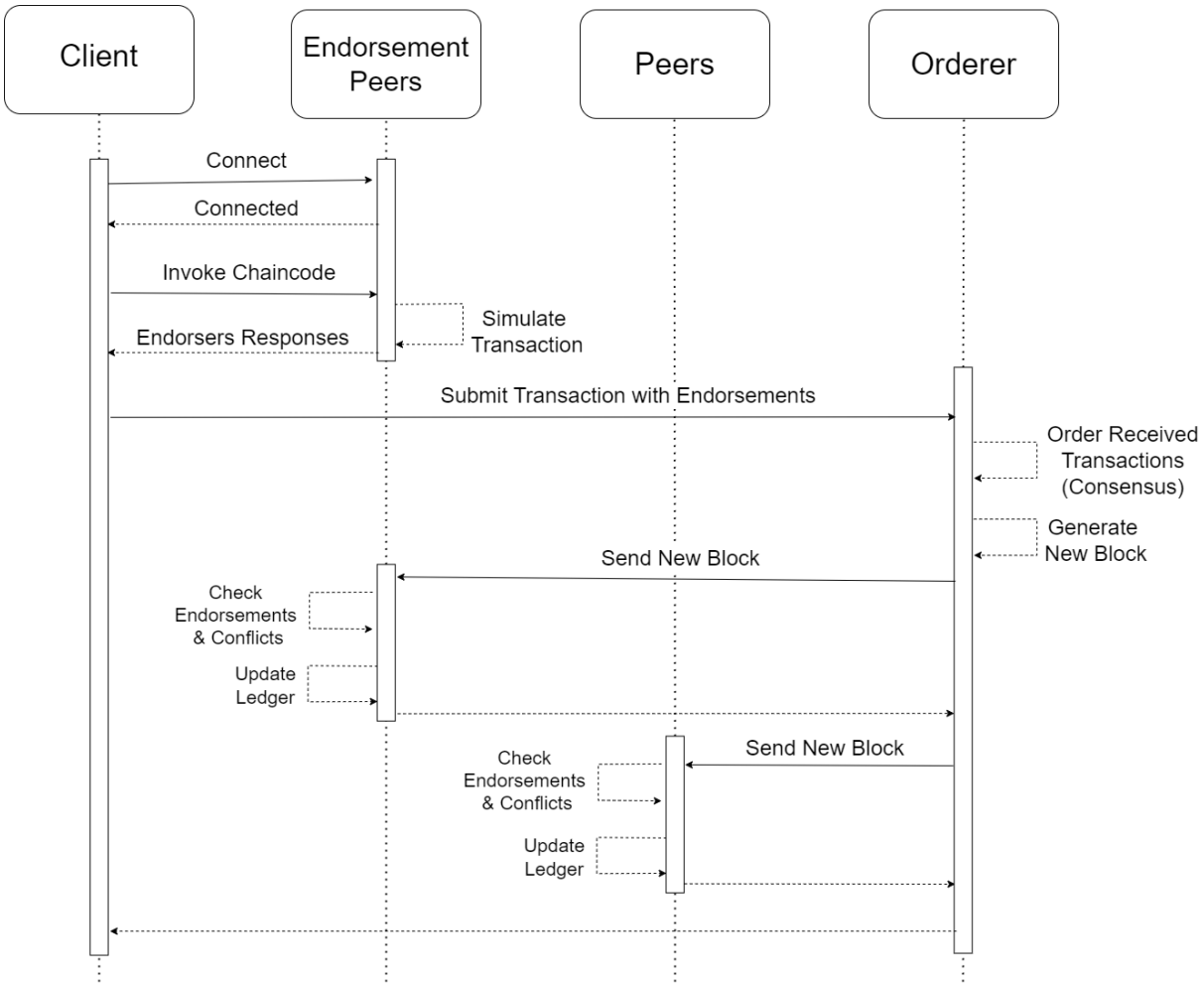


Figure 2.4: Transaction Proposal and Block Addition in Hyperledger Fabric

2.2 Self Sovereign Identity

Digital identity has become one of the significant challenges on the World Wide Web, impacting not only individuals but also institutions and devices within the Internet of Things (IoT). While authentication is an important aspect of digital identity, digital identity extends beyond simply verifying access. Digital identity encompasses proving various attributes about ourselves - such as confirming educational qualifications, residence, or even phone numbers — within

an online context. Over time, identity systems have evolved from simple, physical forms of identification like passports and driver's licenses, to siloed digital systems where each platform or organization holds its own version of your identity. This siloed approach forces users to create new accounts and credentials for every service they interact with, increasing complexity and reducing privacy. As the internet matured, intermediary-based models emerged, relying on third-party entities (i.e., social logins) to vouch for a user's identity. While this system offered convenience, it also concentrated control and risk in the hands of a few central entities, leading to concerns about privacy, security, and data misuse.

With the rise of blockchain technology and distributed ledger systems, new decentralized and permissionless models for managing identity, which support self-sovereignty and self-governance, are emerging. *Self Sovereign Identity (SSI)* [Hardman, 2016], a concept closely tied to decentralized identity models, allows individuals to fully own and control their digital identities without reliance on centralized entities. In an SSI framework, users are empowered to manage their credentials, selectively disclose information, and establish trust directly with verifiers, without the need for intermediaries. Decentralized Identifiers (DIDs) [Reed et al., 2020] and Verifiable Credentials (VCs) [World Wide Web Consortium (W3C), 2023] have been proposed as modern, decentralized alternatives for managing digital identity, supporting the core principles of self-sovereignty. These technologies go beyond authentication alone, enabling users to control and verify their identity attributes in a secure, privacy-preserving manner.

Being tied to always needing to create a new account with every corporate application we want to use can get tiring. Not only that, but the amount of usernames and passwords that we need to remember starts pilling up, becoming a hassle. SSI addresses these pain points by allowing users to interact with various platforms and services using a single, self-controlled identity, removing the need for multiple accounts. Besides this, when disclosing sensitive data to a centralized corporate entity, means there has to be some kind of trust on it which, sometimes, should not exist, as seen with the Facebook Leaks [Heiligenstein, 2023]. With SSI, individuals can share only the minimum necessary data, maintaining control over their privacy while reducing the risks of centralized data breaches.

Globally distributed ledgers, provide a way to managing the root of trust without the need for a centralized authority. Decentralized P2P networks free us from directory services, certificate authorities and domain name registries. SSI thrives in this environment, where users can create, manage, and control their identities through decentralized mechanisms. Combining *Distributed Ledger Technologies (DLTs)* and *decentralized identity management* systems enables any entity to create and manage their own identities on any number of distributed, independent roots of trust [Reed et al., 2020]. SSI also allows individuals and organizations to establish trust through cryptographic proofs, shifting away from hierarchical trust models toward a more user-centric approach to identity management.

2.2.1 Decentralized Identifiers

It is here where *Decentralized Identifiers (DIDs)* come into play. With DIDs, entities can identify themselves using proofs (ex. digital signatures). A DID is a globally unique reference linking to a *DID document* [Brunner et al., 2020]. DID documents contain a set of endpoints which allow interaction with the entity they represent, referred to as *DID subject*. A DID Document is a data structure expressed in, e.g., JSON and contains information about the identity, such as public keys.

Any entity can have as many DIDs as they want. Not allowing that would break the guidelines of *Privacy by Design*. Separation of identities, personas and contexts are no issue. Creating a DID involves generating a unique identifier and associating it with cryptographic keys and other relevant information.

DIDs are composed by tree parts; the URL scheme identifier ('did'), the identifier for the DID method and the DID method-specific identifier. A DID is presented in Figure 2.5.

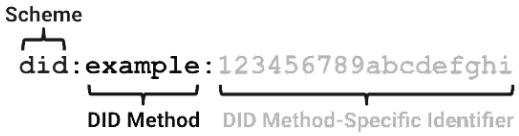


Figure 2.5: A simple example of a decentralized identifier (DID). From [Reed et al., 2020]

Every DID has the first field set to the literal string 'did' to indicate that the identifier follows the DID specification. *DID methods* are unique to each blockchain/distributed ledger. They specify how to create, update and read DID documents attached to the entity, basically they instruct how to interact with DID documents. The second part identifies the blockchain network they belong to. The last part of the DID is the actual key that uniquely identifies the entity within the chosen DID method. Let us say we want to represent a DID on the Ethereum network, that would be represented as:

$$did : ethr : 0x123456789abcdef$$

A DID by itself is not designed to provide trusted personal information about the DID subject on its own but to enable self-sovereignty of the holder and facilitate privacy. As said before, any entity can have as many DIDs as they wish, so a subject might have a DID to interact with their bank and another to shop online. Neither the bank nor the online shops will be able to use the provided DID to correlate personal information about the subject, this ensures online privacy and anonymity.

Listing 2.1: Simple DID Document Example. From [Reed et al., 2020]

```
1 {
2   "@context": "https://www.w3.org/ns/did/v1",
```

```
  "id": "did:example:123456789abcdefghi",
  "authentication": [{
    "id": "did:example:123456789abcdefghi#keys-1",
    "type": "Ed25519VerificationKey2018",
    "controller": "did:example:123456789abcdefghi",
    "publicKeyBase58": "H3C2AVvLMv6gmMNam3uVAjZpfkcJCwDwnZn6z3wXmqPV"
  }],
  "service": [{
    "id":"did:example:123456789abcdefghi#vcs",
    "type": "VerifiableCredentialService",
    "serviceEndpoint": "https://example.com/vc/"
  }]
}
```

As we can see in the example above of a DID, many fields are presented, such as, the unique identifier (id) of the DID, the authentication field, which provides information about the authentication mechanisms associated with the DID, and the service field, which presents the services that can be used to interact with the DID subject.

2.2.2 Verifiable Credentials

Verifiable Credentials (VCs) are digital versions of physical credentials, designed to securely share and prove information about a person, making them more trustworthy and convenient for remote verification [World Wide Web Consortium (W3C), 2023]. A verifiable credential is a form of machine-readable credential that is, according to the specification in [World Wide Web Consortium (W3C), 2023], cryptographically secure and privacy respecting. A VC is bound to a DID and thus linked to an identity. Verifiable credentials are created by the issuer and sent to the receiver. They contain a set of claims about attributes, e.g., name, birth date, grade, ID, or other information the issuer wants to attribute to the receiver. In order to forward a claim to a verifier, a verifiable presentation is created. A verifiable presentation allows to present only a subset of attributes, such as revealing the birth date attribute without the name attribute.

Listing 2.2: Simple Verifiable Credential Example. From [Brunner et al., 2020]

```
{
  "@context": [
    "https://www.w3.org/2018/credentials/v1"
  ],
  "id": "http://example.edu/credentials/42",
  "type": ["VerifiableCredential"],
```

```
  "issuer": "did:example:deeb1f712ebcc276e12ec42",
  "issuanceDate": "2020-06-10T04:20:00Z",
  "expirationDate": "2020-10-10T04:20:00Z",
  "credentialSubject": {
    "id": "did:example:ebfeb1f712ebcc276e12ec21",
    "name": "John Doe"
  },
  "proof": {
    // ...
  }
}
```

As we can see in the example above of a verifiable credential, many fields are presented such as, id of the credential, type, indicating it is a verifiable credential, the issuer's DID, issuance and expiration dates, the credential subject who this verifiable credential targets and, finally, the proof, which is usually a digital signature, that allows for third-parties to verify if the one who truly issued the verifiable credential is the one stated at the *issuer* field.

There are three parties involved in the creation and utilization of a verifiable credential: issuer, receiver and verifier. Let us take a simple example of someone buying alcohol. In this scenario, the receiver is the holder of the verifiable credential, the verifier would be the salesman that needs to check the buyer's age and the issuer would be a third-party who created and signed the credential.

From the previous scenario, something immediately pops out, DIDs and VCs alone are not sufficient for undoubtedly proving an identity. There is no integrated means of connecting the real-world identity to the DID's owner. While we are able to attach VCs to DIDs and therefore would be able to endorse personal information (i.e, name, age, nationality) to a DID, the standard does not describe how the trust in the issuer can be established. In the previous scenario there is no way for the verifier (the salesman) to verify if the issuer is a competent authority to attest for someones age. In traditional digital systems, such as the World Wide Web, this is obtained using a Public Key Infrastructure (PKI) [Adams and Lloyd, 1999] which is supported by a network of trusted entities, such as Certificate Authorities. To address this issue, [Brunner et al., 2020] presents two potential solutions:

- "A promising approach is the establishment of a Web of Trust of public issuers.This would enable that trustworthy institutions vouch for entities they have knowledge of and allow these entities to display their credentials to other parties, ensuring a connection to a real-world identity"
- "A hybrid approach could be implemented by the integration of CAs into the DID ecosystem. This combines the decentralized authentication infrastructure with established roots of trust."

With the help of trusted third-parties (i.e. government, corporations, non-profit organizations) assuring confidence in the verifiable credentials, it becomes way easier, and the scenario proposed above is solved. The salesman can now be sure if the buyer is, or not, of legal age to buy alcohol.

Verifiable credentials allow for multitude of purposes such as identity claim, education certificates, social security credentials, the list goes on. All of this, while maintaining a high level of certainty that the issuer is trusted, high level of certainty that the holder is the one that the verifiable credential was issued to, and high control over the disclosed data to verifiers by the owner due to the possible partial disclosure of information. To manage their verifiable credentials, holders only need a compatible digital wallet.

2.3 Distributed File Systems

A *Distributed File System (DFS)* [Levy and Silberschatz, 1990] is a method of storing and accessing files across multiple machines or devices in a way that appears to the user as if the files are stored locally. These systems allow for greater scalability, fault tolerance, and efficiency in environments where data is spread across numerous nodes. Traditional centralized file systems rely on a single server or cluster to store data, which can lead to performance bottlenecks, single points of failure, and limited scalability. In contrast, distributed file systems distribute the data across multiple nodes, ensuring that data can be accessed from anywhere in the network and making it more resilient to failures. To elevate this concept we can eliminate any central control authority over the data storage and management creating a *Decentralized File System*. Unlike only distributed systems, which are managed by a central entity, decentralized distributed systems such as the Interplanetary File System (IPFS) [Benet, 2014] use a Peer-to-Peer (P2P) architecture where data is distributed across a network of independent nodes. Since there is no central authority managing operations and storage, there truly is no central point of failure.

2.3.1 Interplanetary File System

The Interplanetary File System (IPFS) [Benet, 2014] is a peer-to-peer distributed file system that seeks to connect all computing devices with a decentralized file system. It is similar to the Web but can be seen as a single BitTorrent swarm, exchanging objects within one Git repository. IPFS provides a high throughput content-addressed block storage model by hyperlinks. IPFS combines successful ideas from previous peer-to-peer sytems, including DHTs, BitTorrent, Git, and SFS. Due to all its properties, it has no point of failure and nodes have no need to trust each other. IPFS nodes store IPFSs objects in local storage. These objects represent files and other data structures. IPFS is divided into many different sub-protocols. These are Identites, Network, Routing, Exchange, Objects, Files and Naming.

In IPFS nodes are identified by their *NodeId*, The cryptographic hash of the public-key of their public-private key pair. Nodes communicate regularly with hundreds of other nodes, potentially across the entire internet. In order to find other nodes network addresses, and peers who can serve particular objects, IPFS uses a distributed hashtable. Small values (¡1kb) stored directly on the Distributed Hash Table (DHT), but bigger values store a reference to a peer that can service them. Every node keeps a legder accounting the transfers with other nodes. This allows nodes to keep track of history and avoid tampering. When nodes establish a connection, a ledger exchange is made, and if it does not exactly match, the ledger is reinitialized from scratch.

In order to share and distribute files through its peers, IPFS uses a BitTorrent inspired protocol: BitSwap. Like BitTorrent, BitSwap peers want to acquire a set of blocks - *want_list* - and have another set they want to offer in exchange - *have_list*. Unlike BitTorrent, peers are not limited to the blocks they have. Each peer can acquire whatever block they need, regardless of what files those blocks are part of. Peers have to provide direct value to each other in the form of blocks. This works fine unless a node has nothing to offer its peers. In this case, the node has to 'work' to seek pieces its peers want. This incentivizes nodes to cache and disseminate rare pieces, even if they are not interested in them directly. Nodes provide direct value to each other. The DHT and BitSwap allow IPFS to form a massive peer-to-peer system for storing and distributing blocks quickly and robustly. On top of these, IPFS builds a Merkle DAG, a directed acyclic graph where links between objects are cryptographic hashes of the targets embedded in the sources. This is a generalization of the Git data structure. These Merkle DAGs provide IPFS with *Content Addressing*, meaning that all content is uniquely identified, *Tamper Resistance*, meaning that all content is verified for corruption and *Deduplication*, meaning that all objects that the exact same content are equal and only stored one. To traverse IPFSs objects, paths are used. Paths work as they do in the Web. The format is as follows

$$/ipfs/ < hash - of - the - object > / < name - path - to - object >$$

which ends up looking like this

$$/ipfs/XLYkgq61DYaQ8NhkcqyU7rLcnSa7dSHQ16x/foo.txt$$

IPFS clients free some of disk space for IPFS to store and manage objects. Some nodes may wish to ensure the survival of specific objects. This can be done by a process called *pinning*. This ensures that objects are kept in the node's local storage, resulting the existence of *permanent links*. When users whish to publish a node they simply need to add its key to the DHT, adding themselves as peer and giving other users the object's path.

IPFS also defines a set of objects for modeling a versioned filesystem on top of the Merkle DAG. There are four set of objects: block - represents a variable-size block of data (file), list

- represents a colletion of bocks or other lists, tree - represents a collection of blocks, lists or other trees and commit - represents a snapshot in the version history of a tree.

2.4 European Blockchain Services Infrastructure

In 2018, European Blockchain Services Infrastructure (EBSI) [European Commission,] was born when 29 countries and the EU Commission joined forces to create the European Blockchain Partnership (EBP), establishing a blockchain infrastructure for cross-border public services. The initiative focuses on leveraging blockchain to create a trusted and secure environment for the exchange of information and documents between public administrations, businesses, and EU citizens. EBSI works by having governments and other national institutions registering themselves as Trusted Issuers (TIs) of electronic documents. The organizations which give different parties the status of trusted issuer of a specific sector/ domain in a specific geography, to issue certain types of verifiable credentials, are Trusted Accreditation Organisation (TAO). For example, in the education domain, the Ministry of Education of a Country is responsible for accrediting the Universities of that country. TAOs also register the trusted schemas associated to the Verifiable Credential, i.e. Diploma. EBSI acts as a public registry of Issuers, containing the list of trusted Legal Entities that are accredited by TAOs to issue certain types of credentials. Every participant and entity in the network is represented by their own DID. Some trusted issuer issues a verifiable credential to a DID, the owner presents the variable credential to a verifier and, the verifier checks the validity of the credential.

3

Related Work

Contents

This chapter presents the existing body of research and implementations in the domains of blockchain-based land registry and satellite imagery. Some examples of land registry projects and attempts will be presented and how they tackle the issue. Furthermore, the currently available technology and market options to obtain satellite images will be presented.

3.1 Land Registry in Portugal

In Portugal, the Registo Predial (Land Registry), managed by the Instituto dos Registos e Notariado (IRN), is the official system responsible for documenting the legal status of properties. Land registry plays a crucial role in certifying property ownership and ensuring legal clarity regarding property rights, transfers, and transactions. By maintaining accurate records of ownership and boundaries, IRN helps to reduce legal disputes and promotes transparency in the real estate market. However, despite its importance, many rural properties in Portugal lack updated information, creating challenges for effective land management. It is in this context

that the Balcão Único do Prédio (BUPi) was developed, as a complementary digital platform to streamline and modernize the process of registering and managing these properties.

The Balcão Único do Prédio (BUPi) [Balcão Único do Prédio (BUPi), 2024] is a digital platform developed by the Portuguese government aimed at streamlining and centralizing the process of identifying, registering, and managing rural and mixed properties in Portugal. The primary goal of BUPi is to establish an updated and comprehensive registry of these properties, ensuring that each parcel of land is properly documented and identified. This initiative not only seeks to combat issues related to unknown or undefined land ownership but also promotes the regularization of property registration.

Through BUPi, property owners can identify the boundaries of their land using an online system that employs georeferencing technology. The platform is integrated with other public databases, such as the Land Registry and Tax Office, allowing for effective centralization of information and preventing duplicate registrations.

One of the key advantages for citizens is that registering through BUPi provides greater legal security, which is crucial in situations such as inheritance, property sales, or legal disputes.

BUPi is intended for all owners of rural and mixed-use land, including agricultural, forested, and properties with both rural and urban components. The expected impact of this initiative is substantial, as BUPi will contribute to a more complete and accurate national land registry, facilitating land management and urban and rural planning. Moreover, by ensuring that properties are correctly identified, BUPi helps prevent ownership disputes and increases transparency in the real estate market. Environmentally, the initiative supports better land stewardship by ensuring that properties are registered and properly managed, which can prevent illegal activities such as deforestation.

Beyond these primary goals, BUPi has significant legal and administrative implications. It simplifies legal processes related to property, especially in cases of inheritance or disputes, by providing clear and undisputed documentation of property boundaries and ownership. Additionally, BUPi integrates into broader land reform efforts in Portugal, aiding in the implementation of land policies that promote sustainable rural development.

However, BUPi faces challenges, including ensuring widespread adoption, particularly in remote regions with limited internet access or among older populations less familiar with digital tools. Additionally, disputes over the accuracy of georeferenced boundaries may arise, requiring careful resolution. Looking to the future, BUPi could expand to include more detailed layers of information, such as land use restrictions or real-time updates on legal changes related to property.

Technologically, BUPi is part of a broader trend toward integrating advanced tools into public administration. The platform uses GIS (Geographic Information System) technology for mapping, and as digital tools continue to evolve, BUPi could incorporate more sophisticated features, such as satellite imagery or AI-driven analysis of land use patterns.

3.2 Blockchain Land Registry

The integration of blockchain technology into land registries has gained significant attention as a means to enhance transparency, security, and efficiency in property transactions. Some land registry blockchain systems are already being developed and tested around the world in countries such as Sweden [International Council for Accreditation of IT Managers, 2016], India [Government of India, Ministry of Electronics and Information Technology, 2020], Honduras [Borgen Project, 2021], Japan [Asia, 2017], Brazil [Lemieux et al., 2018] and more ([Shuaib et al., 2020]). Some of these projects are either in a developing stage or pilot testing in small towns and others have even been completed, such as the Swedish [Cha, 2023], or , sadly, 'stalled', such as in Honduras [not specified, 2015]. A developed country like Sweden would pursue the introduction of this technology to the market in order for a more transparent process on land transfers and claims. On the other hand, an underdeveloped country like India, which has tremendous problems due to lack of land registration and false property claims, wants to start building a reliable and useful land title repository, so to resolve conflict of interests, false property claims and unfounded land disputes. Some papers written about this proposed to be state-of-the-art, such as [Khan et al., 2020, Sahai and Pandey, 2020], propose using a very similar algorithm for land transfer. Firstly, users (buyer/seller) would register themselves on the platform. Secondly, sellers need to upload the property details, such as total area, to the platform. Third, an interested buyer sends a request to the seller to access the property specification. Seller can deny or allow for the buyer's access after inspecting their profile. If the transfer is to be performed, a land inspector will be notified, who will verify the authenticity of the seller and buyers documents. A smart contract is triggered for the transfer. Both parties sign the transfer document and the land inspector registers the transfer in the blockchain. The hash value of the document uploaded by the owner is the same to the hash value to the time of buying the property (signing) then the document is 100 percent authenticated. Both these papers ([Khan et al., 2020, Sahai and Pandey, 2020]) also suggest smart contracts so that every step can be made possible.

In comparison to the system we propose in this thesis, one added factor is the integration with SSI, more specifically DIDs and VCs. None of the papers mentioned above integrate their blockchain land registry with digital identity. So, this thesis takes one step further in the digitalization process, introducing a possible new way of interconnecting technologies and easing the whole process.

3.3 Satellite Imagery

With the recent technology advancements, readily available and updated satellite images are becoming more and more a reality. These offer a unique vantage point that transcend geographical constraints allowing entities to get full coverage over large blocks of land. Satellite

images can empower entities to have a database with over time land changes. The satellites and cameras that capture these images use a wide variety of different active and passive sensors. The necessity to have different sensors is crucial due to weather conditions. On a cloudy day with less visibility clear images will not be possible to attain, so microwave sensors will be used instead of optical ones. With all the different sensors, various kinds of information can be attained such as, normal ground images, weather pattern graphs, flooding mappings and earthquake assessment mappings.

In the context of this thesis proposal, the technology we are interested in is the ground level images to record land changes over time. A barrier that must be overcome is the ease of access to these images. Right now there are some websites and institutions that provide satellite images and graphs for free such as the NASA WorldViewer [NASA,] that can provide users with different kinds of maps including flooding and normal ground images and, LandViewer [EOS,]. Both the previous examples are of public access and both face the same issue, low resolution. Low resolution leads to impossible assessment of the current state of the land being captured in the images. There are however other corporations providing supposedly high resolution ground images that could possibly be used. These were not used in research due to the contents being only accessible behind a paywall.

4

InsureConnect Design and Implementation

Contents

In this chapter, I will describe the developed system, *InsureConnect*.

InsureConnect is a system that allows insurance companies and their clients to register, manage, and resolve insurance contracts and property damage claims. The system primarily aims to assist with contracts where the client's property is large enough for satellite imagery to easily capture and detect pre- and post-event changes in the case of natural disasters such as floods, earthquakes, or typhoons. To achieve this, each participant must be registered on the platform. Insurance companies and clients will then be able to establish contracts and submit property damage claims. The authenticity of the submitted data is ensured by signing the transaction contents with the key-pair registered in the ledger's system, which should be derived from the EBSI, allowing any external third party with the proper read privileges to verify each transaction. Additionally, all media content will be stored in IPFS to ensure full decentralization of the system.

This system is an academic project, and all mentioned entities are not involved with it.

4.1 Participants and Roles

Participants and roles are necessary in any system. Each participant when registered in the system, is assigned a role, which will influence how they can interact with the system.

There are three main roles that represent a given group of participants:

- **Higher Authority** - A regulatory or oversight body responsible for creating and maintaining the DIDs for insurance companies. This role involves ensuring the integrity and authenticity of the DID documents, which are essential for verifying the identity and legitimacy of the insurance companies in various transactions and interactions.
- **Insurance Company** - An organization that provides financial protection to clients. The insurance company assesses risks, collects premiums, and pays claims to policyholders, including those affected by natural disasters.
- **Client** - An insurance company client with one or more property insurance plans. This entity is the policyholder or a beneficiary covered by an insurance policy. The goal is to receive compensation or support from the insurance company to recover from the losses incurred due to the natural disaster.
- **Auditor** - An independent individual or company responsible for examining and verifying financial records, statements, and transactions of organizations. Auditors ensure financial accuracy, compliance with regulations, and the reliability of financial information provided by insurance companies to policyholders and regulatory authorities. They have (read-only) access to the platform in order to collect all the necessary documents to perform the audition.

Each role has its own set of access permissions based on a clearance level approach, according to the task each participant is in charge of. Figure 4.1 displays all participant roles and permissions

All participants must be registered in the blockchain's ledger through the chaincode and in *Fabric's Membership Service Provider* (MSP). These two registration methods differ in that registering the user in the ledger involves invoking a chaincode to create a DID Document object representing them and permanently storing it in the ledger. Registering with Fabric's membership service provider is outside the ledger's scope. An admin entity (any entity with admin credentials) must register the new users and enroll them in a certificate authority. This topic and all the reasons behind the taken decisions will be explored in more detail shortly.

The 'Higher Authority' role is assigned to only one user and is already *hardcoded* when the system is launched. This user is responsible for registering all 'Insurance Company' users

to join the network. 'Insurance Company' users are responsible for registering their respective clients and any hired 'Auditor' that may not have been registered yet, therefore have permission to register users assigned the 'Client' and 'Auditor' roles.

Following the registration, each role will behave differently and have different permissions.

Firstly, the 'Insurance Company' role is permitted to create, update, and read insurance contracts and damage claims in which it is involved. This extensive access ensures that the insurance company can efficiently manage and process the necessary documentation and claims related to their clients.

Secondly, there is the 'Client' role. Individuals or entities assigned to this role are able to update insurance contracts, ensuring that their contract signatures can be added promptly. Additionally, clients can create, update, and read their damage claims. This allows them to report incidents, track the progress of their claims, and make any necessary amendments, ensuring they are always informed and involved in the resolution process.

Lastly, the Auditor role is characterized by a more restrictive set of permissions. Auditors are granted read-only access, specifically to records that have been designated for dispute. This limited access is intended to provide auditors with the information necessary to perform their duties of reviewing and verifying claims and contracts under dispute, ensuring transparency and accountability without compromising the integrity of other sensitive data.

These tailored roles and permissions ensure that each participant can effectively fulfill their responsibilities while maintaining the necessary levels of security and control over the sensitive information involved in insurance operations.

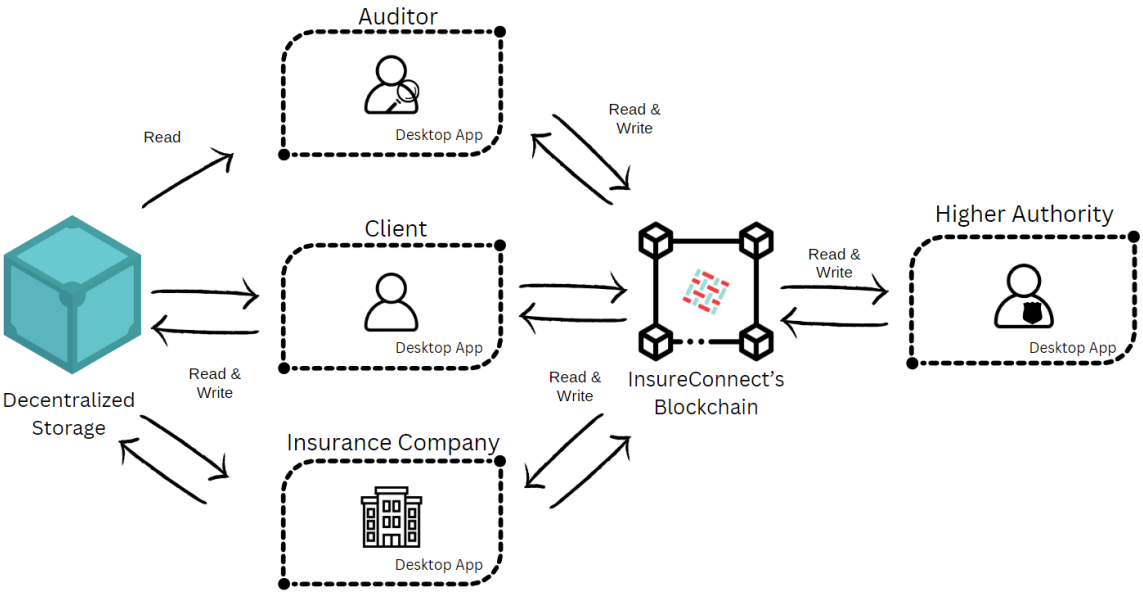


Figure 4.1: InsureConnect's Roles and respective Permissions.

4.2 Architecture

InsureConnect will make use of one private permissioned Hyperledger Fabric blockchain to store the records, a decentralized storage network to store images, and a desktop app to allow participants to connect and interact with the aforementioned systems. These are listed below.

InsureConnect is architected as a decentralized, containerized system, leveraging Docker for consistent deployment and orchestration across various environments. Every component of the system, whether it's the blockchain nodes or the IPFS storage nodes, is packaged and deployed within Docker containers. This approach ensures portability, rapid horizontal scaling, and isolation of services, allowing the entire solution to be deployed seamlessly across multiple hosts. Docker's container orchestration simplifies the management of the network, making it easy to add or remove nodes while maintaining the system's integrity. The architecture consists of three main layers: the blockchain network, the decentralized storage system, and the client-side interface. Each component and their interaction is detailed below.

- **InsureConnect's Blockchain** - Hyperledger Fabric will be used to store DID Documents, Insurance Contracts, and Insurance Claims. This permissioned blockchain ensures that only users involved in a particular transaction can access their data, providing privacy and security. The chaincode, deployed across all hosting peer nodes, enforces these access controls and ensures the proper execution of smart contracts related to claims and contracts. The blockchain also serves as the foundational layer of trust and transparency for the system, ensuring data immutability.

- **Decentralized Storage** - A peer-to-peer distributed network will be utilized to store images, specifically satellite imagery of the property land, both before and after damages have occurred. The system employs IPFS (InterPlanetary File System) for off-chain storage, addressing the high cost and inefficiency of storing large datasets like images directly on the blockchain. Each file is stored in a decentralized manner, ensuring that the content is accessible and tamper-resistant, while the blockchain stores a reference (content hash) to the file on IPFS, creating a seamless link between on-chain and off-chain data.

- **Desktop App** - This app acts as the primary interface for participants, allowing them to interact with both the blockchain and the decentralized storage. Users must upload their public certificate and private key locally to the app, enabling secure signing of transactions. The desktop app communicates directly with the blockchain to submit transactions and queries, while simultaneously interfacing with the IPFS network to retrieve and store files. Through this app, users are able to interact with both the blockchain and the decentralized storage in a unified and secure manner, without needing to understand the underlying complexity.

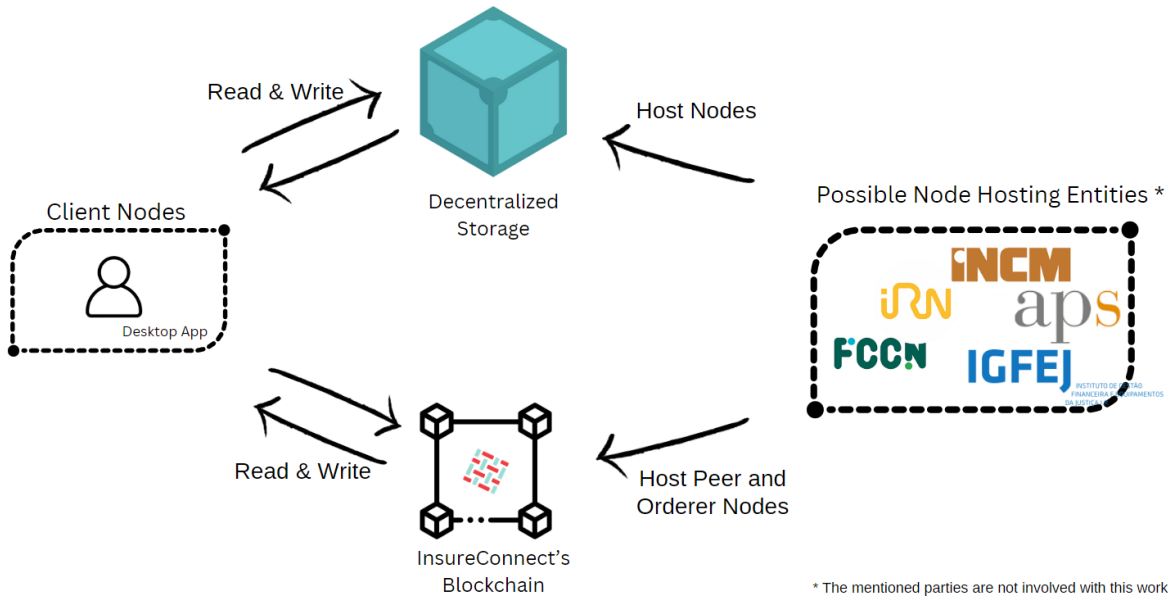


Figure 4.2: InsureConnect's Architecture

Figure 4.2 shows InsureConnect's architecture.

The objective of the solution is to implement a fully decentralized system. Keeping that in mind, all records will be stored on the blockchain, including DID documents, insurance contracts, and damage claims. Storing data on the blockchain is costly, so it is necessary to use an off-chain storage system. To that end, the system will make use of a decentralized, tamper-resistant, content-addressable, peer-to-peer storage network. In this project, we chose that system to be the InterPlanetary File System (IPFS) [Benet, 2014], which is commonly used and peer-to-peer. Users need access to an IPFS node so that they can store and access data using the IPFS network. These IPFS nodes will be hosted by the same parties that will also host the Hyperledger Fabric nodes. Participants will then be able to interact directly with the blockchain network and the IPFS distributed file system through the desktop app, which will communicate with the previously mentioned nodes.

4.2.1 Blockchain

The chosen blockchain to build this project was Hyperledger Fabric [Dhillon et al., 2017] from the Linux Foundation. This blockchain was selected because it meets the development requirements of the project, such as identity management.

The blockchain should be hosted by a minimum of 5 different entities to be able to gracefully handle fault tolerance. These entities will each host a peer node and other 3 of them will host an orderer node. So, in total, there are 8 nodes maintaining and keeping the network functioning. In case of node failures, the system is designed to remain operational as long as peer and orderer nodes are functional. If a peer node fails, other peer nodes can continue endorsing and processing transactions, while orderer nodes ensure the network can still se-

quence transactions as long as a majority are online. This fault-tolerant architecture allows the system to recover gracefully and maintain integrity, even during node outages. Every external application that connects to the peer nodes to submit transaction requests is a client node.

With at least 5 peer nodes hosted by different entities, we can ensure good resilience and fault tolerance. If one or more peer nodes go down due to technical issues, the remaining peers can continue to maintain the ledger, endorse transactions, and serve clients. This distributed setup minimizes the risk of network downtime, ensuring that the blockchain remains operational even if one or two peers face issues. Additionally, with 5 nodes, we also ensure good scalability, as the network is better equipped to handle an increasing number of transactions and client connections as the blockchain ecosystem grows. More peers can process and endorse more transactions in parallel, improving the network's capacity and performance.

With 3 orderer nodes, the same reasons presented above stand true, except that this time, it really is crucial to keep at least one of them running since, without orderer nodes, Fabric's network cannot function. Orderer nodes are mandatory to order transactions and build blocks. Without them, the whole network fails. With 3 orderer nodes, we can ensure good resilience and fault tolerance. The consensus mechanism employed by the orderer nodes is *Raft* [Ongaro and Ousterhout, 2014]. Raft was chosen over the other two options in Fabric: Kafka and Solo. Solo immediately was not an option since it is not supposed to be deployed in any production environment besides a testing one. It is composed of only one orderer node that processes and orders every transaction, making it completely centralized, vulnerable to attacks or service disruptions, and lacking fault tolerance, which could prevent recovery from node crashes or network failures. While Kafka was the default consensus mechanism in Fabric for some time, it has now been replaced by Raft as the default. Kafka, like Raft, is a crash fault-tolerant (CFT) consensus mechanism. It is based on Apache Kafka, a distributed streaming platform. Kafka provides strong consistency guarantees by replicating transactions across multiple nodes and ensuring they are committed in the same order. However, due to requiring an additional ZooKeeper [Hunt et al., 2010] cluster to manage the Kafka's pool of servers (called "Kafka brokers"), complexity is added to the system's deployment and maintenance. A high level of expertise in Kafka infrastructure and settings is required to deploy a Kafka cluster and ZooKeeper ensemble. Since there are many components, there are many places where things can go wrong. Although Kafka provides strong consistency and fault tolerance, its added complexity, particularly the requirement for a ZooKeeper cluster—can impact the overall system availability, especially during failures or maintenance. According to the *CAP Theorem* [Gilbert and Lynch, 2012], a distributed system can prioritize only two of the three properties: Consistency, Availability, and Partition tolerance. Raft is a CP (Consistency and Partition tolerance) protocol, ensuring that the system remains consistent even in the event of network partitions, at the cost of temporary availability during leader re-elections. Kafka also leans toward the CP model, but due to the need for ZooKeeper and its more intricate setup, it introduces operational complexity that can impact overall availability in practice. Due to these

reasons, Raft becomes the chosen consensus mechanism. Besides currently being the default mechanism for Fabric, Raft provides a simpler, more robust solution that is easier to configure and maintain. Orderer nodes can be added or removed on the fly, and it achieves consensus through a leader election process that is both more straightforward and resilient. In the case of a leader failure, Raft can quickly elect a new leader, ensuring minimal disruption to the network. Although, from the perspective of the service they provide to a network or a channel, Raft and Kafka are very similar, due to Kafka's complexion and Raft's simplicity, Raft becomes the *de facto* chosen consensus mechanism.

As mentioned, in Section 2.1.3, there are 3 node types. All remaining node types will a client nodes, which will be hosted by every participant who wishes to interact with the network.

To interact with the developed system, two different pairs of credentials are necessary. The first keypair is a public/private keypair, which will be derived from EBSI, and will is used to sign the transaction's contents. This signature is then appended to the transactions contents and recorded with the submitted object in the ledger. By recording the content's signature alongside the object in the ledger, anyone wishing to verify the validity of future transactions can do so by comparing the transaction details with the corresponding signature. The need for a second keypair arises from the unique requirements of our system design rather than being a standard feature of Hyperledger Fabric itself. While our approach allows us to effectively address the challenges associated with write operations, while keeping a stored proof of the issuer of the transaction permanently stored, where transaction contents can be signed for a 'write' operation on the blockchain, the same logic cannot be applied to 'read' operations, as there is no content being submitted to sign. In many systems, what is done to surpass this problem is to have the user sign a proposed challenge. This could not be done as the challenge would need to be stored inside the blockchain for later checking. Storing every single challenge for every single read operation would quickly overwhelm the network with a lot of requests, unnecessary computing and wasted storage. To tackle this problem, the second keypair comes into play. The chosen solution relies on Hyperledger's Fabric Membership Service Provider (MSP). With Fabric's MSP, each participant can be registered individually and receive their respective public/private keypair. With this pair, each user can now be recognized by the network since their transactions will now be signed, and their read operations are now authenticated, restricting read operations on specific objects to allowed users only.

Besides, node peers and ordereres, at least one entity must host a certificate authority (CA) to register and issue all the participants identities related to transaction signing.

4.2.2 Ledger

Each peer node will have one chaincode deployed (see Section 4.3), which client peers will submit transactions to. Every object in the network will be stored in the same ledger.

There are three main objects represented within the chaincode (see Figure 4.3): DID doc-

uments, insurance contracts, and damage claims. As the main objects that hold crucial data in the network, all of them need to be accompanied by a signature of their attributes in JSON format to be submitted. These three main objects are listed below:

- **DID Documents** - The DID documents are stored within the ledger for peers to access directly, avoiding the need to connect outside the Docker container in which they are running. Looking up data outside the running environment could cause diverging results due to external data updates, which might result in different outputs. In a blockchain environment, inconsistent outputs can lead to the blockchain not progressing, potentially causing it to get 'stuck' as result. Even though this might only occasionally be a minor issue, it was deemed important to prevent it to avoid unforeseen consequences. These DID documents are crucial for identifying and verifying who performs 'write' operations. They contain the identity of the entity ('did') and their public key, composed of 'exponent' and 'modulus' to form an RSA key. With this, when any transaction is submitted to write on the ledger, peers can check who is supposedly sending the transaction and verify the signature that is submitted along with it by grouping the other attributes in JSON format and comparing. With the signature verified belonging to the user that should be submitting the transaction, the transaction is accepted. There are three more fields involved in the creation of a DID Document, 'kty' (key type), 'DateTime' (the time at which the DID document was created and signed) and 'EntityType' (which assigns a role to the entity, broadening or limiting their access permissions).
- **Insurance contracts** - These, as the name suggests, represent insurance contracts between two entities. These objects contain the DID document identifiers for both involved parties, an IPFS link pointing to the IPFS directory with the images of the property, the client's property verifiable credential, the date and time at which the contract was created, and the signatures of both respective entities. The contract becomes valid only when both signatures are present, indicating endorsement by both parties. The contract is created by the insurance company and the client updates it with his signature for it to be considered valid. This object can be updated, but when it is, both parties' signatures are erased and the object must be re-signed so it becomes valid once again.
- **Damage Claims** - These are the means by which the clients can claim to receive compensation or support from the insurance company to recover from the losses incurred. These are submitted with a reference to the client, the respective insurance contract, an IPFS link, the current date and time, the claim's state (defaulting to 'PRESENT' upon creation) and the signature of the entity creating the claim (both insurance company and client can create the claim). A new IPFS link pointing to a new IPFS directory must be submitted, so both photographies, in the moment of the signature of the insurance contract and the current state, can be compared. Every damage claim has a current state which is currently at. Damage claims progress through the following states: Presented,

Processing, Handled and Canceled -, in order. These states are merely for prototype purposes and can be easily modified. They are successive and once a new stage is reached, the claim cannot revert to a previous one. A signature must also be presented to validate the issuer of the transaction. Every time the damage claim is updated a new signature must be presented with the new updated values.

Bellow, Figure 4.3 presents, is the UML class diagram of the ledger objects.

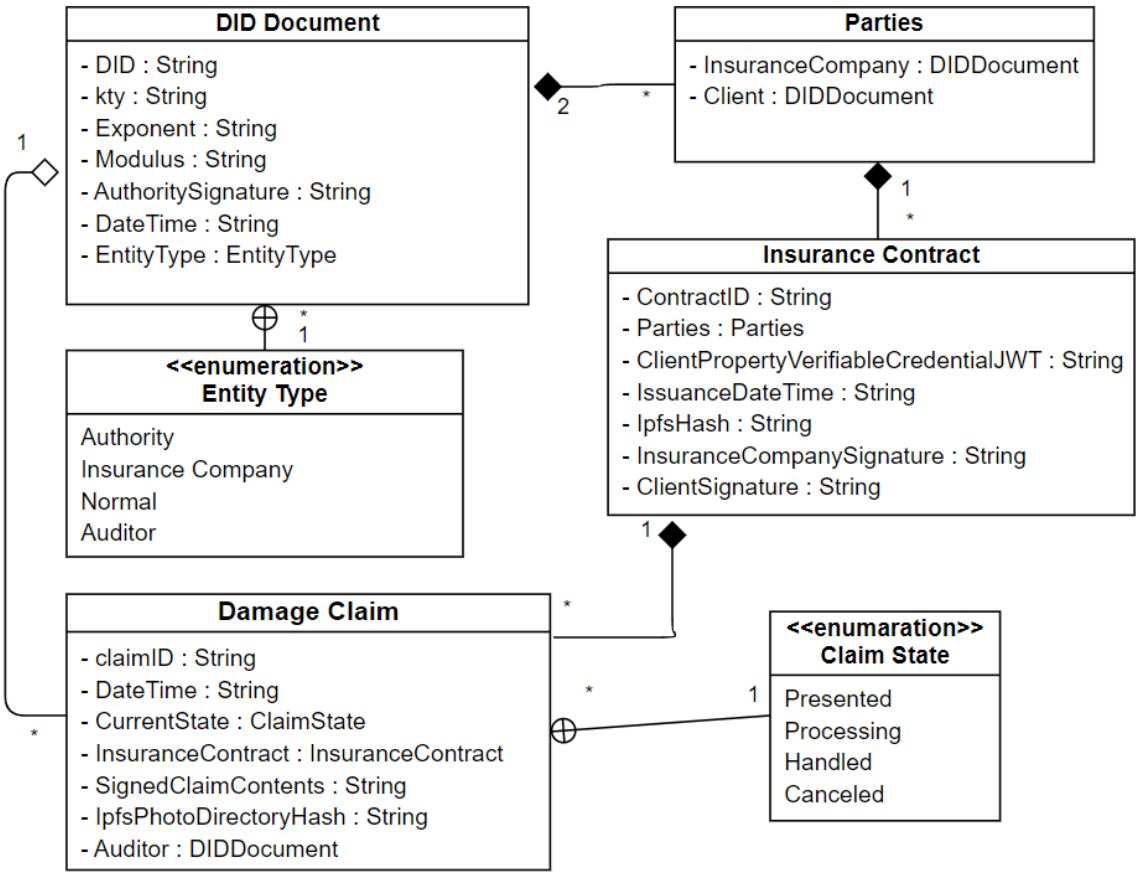


Figure 4.3: InsureConnect UML Class Diagram.

It is important to note that, regarding DID documents, although the 'kty' field is stored and exists, it serves no purpose in this prototype since the only accepted type of signatures are RSA signatures. RSA was chosen due to system constraints. Hyperledger Fabric's containers run on Java 11 [Hyperledger Fabric, 2024]. This proved to be a very challenging development factor and impacted major development decisions. With this limited Java version, many libraries which were used to compute and verify transactions of other encryption algorithms, i.e. EdDSA which is a faster and more secure algorithm than RSA while having smaller key sizes, are only compiled in Java 16 or above. This lead to uncompilable code due to versioning. RSA was the first with which the development was successful.

4.2.3 Decentralized Storage

As mentioned before, Interplanetary File System (IPFS) is used to store all contents related to property damages, more specifically, images of the property taken before and after a claim has been submitted to document the damages. When a user creates either an insurance contract object or a claim object, an IPFS link is required to submit any of the transactions. This IPFS link will directs to an IPFS directory with all the photos.

IPFS plays a critical role in the project due to the large amount of storage that images require to be stored. All this space could not be wasted on the Fabric's blockchain itself, so a new database was necessary to store them. To keep the whole solution as decentralized as possible, a decentralized database was necessary. Given this requirement, IPFS was chosen as the decentralized database for InsureConnect.

There were other options considered before choosing IPFS as the *de facto* database, such as 'BitTorrent' [Cohen, 2003] and 'Swarm' [Swarm Team, 2020]. Swarm was quickly discarded since it is tightly integrated with the Ethereum network, making it a better choice only if the solution was developed within the Ethereum ecosystem. BitTorrent rivals IPFS on some fields, such as performance, efficiency and ease of use but ultimately loses in others. BitTorrent loses in both decentralization and content immutability. BitTorrent utilizes so called 'trackers' to assist peer nodes to discover each other and content they seek, meaning the need for some centralized content exists. IPFS is totally decentralized with the utilization of a distributes hash table (DHT). BitTorrent utilizes location-based addressing which is less reliable than content-addressed data, utilized by IPFS, where files are identified by the hash of their content. This ensures that by storing the original link of the content when it was upload, it is never meddled with again, since changing the content would mean, changing its hash. This feature proves extremely useful to prove that images were not tempered with.

The solution uses the public IPFS network. Every image will be uploaded "publicly". While this stands true, as mentioned before, IPFS relies on content-address data so, without the link to the IPFS directory, the contents themselves are inaccessible. Since only the involved parties (i.e. Insurance Company, Client) can read the link from the Fabric's blockchain, only they can access the content.

While using IPFS, another very important aspect to keep in mind is IPFS's garbage collector. IPFS automatically removes data deemed unused by the garbage collector. To work around this issue, we can *'pin'* the content, as mentioned in Section 2. With this we ensure the content will never be garbage collected and persists alive.

An alternative solution could involve creating a private national IPFS network, thereby avoiding reliance on the public global network and ensuring the content is truly private. However, this approach would require a greater number of computers and entities to host storage units. Relying solely on the existing five entities responsible for hosting the Fabrics blockchain would essentially result in a centralized distributed database. This is because the current num-

ber of storage units is insufficient to maintain true decentralization, where the system would remain unaffected even if one or two nodes were to fail.

4.2.4 Desktop App

As mentioned before, a desktop app will be used by every user to interact with the network. Within the app every user will have access to try and perform every operation. The app does not discern between 'Insurance Company' type user, 'Client' user or any other type of user. The discerning is done by the chaincode itself in the blockchain, so the app presents every possible option of interaction with the chaincode to every user. As mentioned before, the blockchains uses the users identities and submitted signature to verify the users transactions.

InsureConnect's desktop app can divided in four major components in accordance to their use:

- **Identity Management menu** - menu responsible for creating a DID document in the blockchain, retrieving an identities data and update a verifiable credentials.
- **Insurance Contract Management menu** - menu responsible for creating insurance contracts, updating the clients signature on the contract, updating the contract and retrieving an insurance contracts data.
- **Claim Management menu** - menu responsible for creating claims, updating them and retrieving their data.
- **Add Public and Private Keypairs menu** - menu responsible for adding the public and private keypairs, one issued by the Fabric's MSP and the other from EBSI, necessary to submit transactions to InsureConnect's blockchain.

To add further context to the last menu, the "Add Public and Private Keypairs menu", this menu serves for entities to upload their public/private keypairs to the application, so the application stores it and is able to sign the transactions for the Fabric's blockchain to recognize.

4.2.4.A Web App

Although the desktop application was ultimately chosen as the final solution, the original approach involved a web application hosted by each node. However, this solution was later discarded due to the 'Read' operations issue described earlier. In the web app solution, users would submit the transaction content and its corresponding signature through the web interface. The web app's host would then build the transaction request and submit it using admin privileges. While 'Write' operations would remain unchanged, 'Read' operations would become indistinguishable, making it impossible to identify who submitted them. This issue arose because all transaction requests originated from the same user—the node that received the

transaction contents via the web app and built the transaction request to send to Fabric. Due to the privacy concern, the web app solution was abandoned in favor of a more privacy-respecting desktop application.

4.3 InsureConnect Operations

In this section, I describe the supported operations on the chaincode. It is divided into two subsections where each one addresses to different objects. The first subsection talks about identity objects methods, specifically focusing on DID documents. The second subsection details methods for interacting with insurance contracts and managing claims. Figure 4.7 shows every possible operation by every role.

This section also aims to guide the reader through the entire process, from the creation of a 'DID Document' to the resolution of a 'Claim.'

4.3.1 Identity Management

Within this menu, there are two further options: 'Create a DID Document' and 'Check a DID Document'.

To create a 'DID Document', the following fields must be provided:

- **Decentralized Identifier (DID)** - This field holds the identity of the entity in the platform
- **Exponent** - This field holds the exponent component that composes an RSA key
- **Modulus** - This fields holds the Modulus component that composes an RSA key
- **Date Time** - This fields holds the date and time at which the transaction was signed and submitted to the network
- **Entity Type** - This field holds the type of user this entity will be
- **Authority Signature** - This field holds the signature of the user who issued the transaction and submitted the 'DID Document' creation request

Once the transaction is submitted to the Fabric's network, the signature is verified. A 'Higher Authority' user signature is required to create 'Insurance Company' users and an 'Insurance Company' user signature is need to create 'Client' and 'Auditor' users. If the signature is valid, a new 'DID Document' is created and stored in the ledger.

At this stage, an additional procedure is needed: the registration and enrollment of the user in Fabric's MSP. As mentioned before, this is done by users with admin permissions. This admin permissions are granted only to the Higher Authority and Insurance Companies. Once the registration and enrollment are complete, the generated public/private key pair is sent back

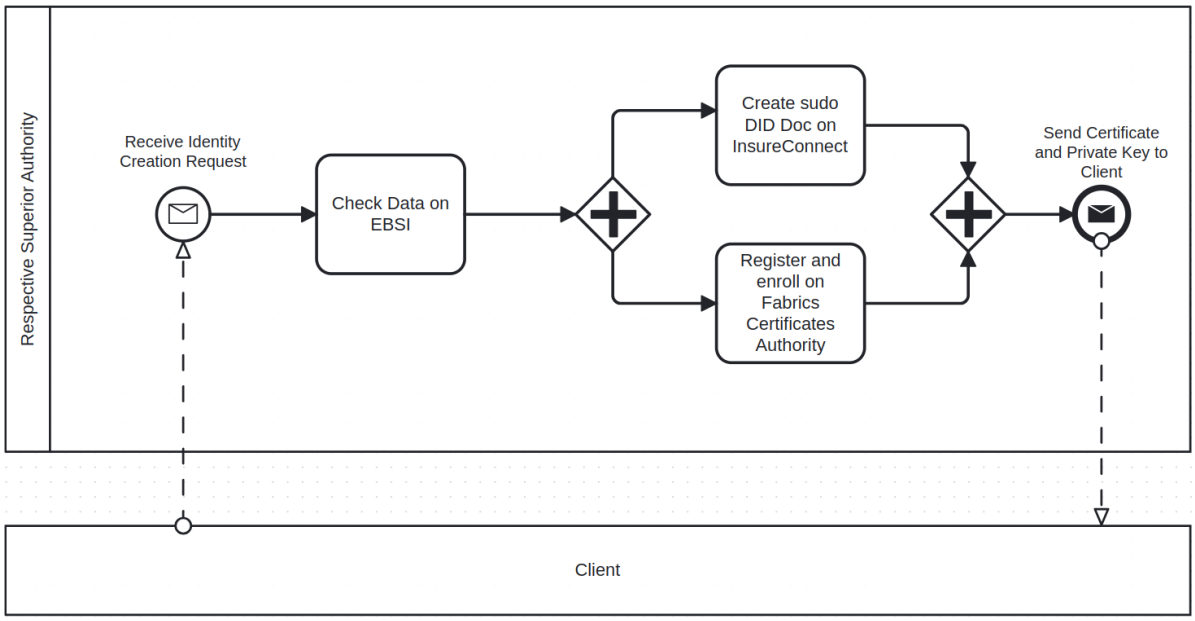


Figure 4.4: BPMN graphic displaying identity creation process.

to the newly created user, allowing them to load the credentials into their desktop app. Figure 4.4 illustrates the identity creation process.

To check a 'DID Document,' the only required input is the DID that the user wishes to verify. The corresponding data is then retrieved and displayed.

4.3.2 Insurance Contract Management

Within this menu, there are four further options, 'Create an Insurance Contract', 'Update an Insurance Contract', 'Update the Client Signature' and 'Check my Insurance Contracts'.

To create an 'Insurance Contract', the following fields must be provided:

- **Insurance Company DID** - This field holds the identity of the insurance company
- **Client DID** - This field holds the identity of the client
- **Client Property IRN VC** - This fields holds the client's VC issued by IRN stating that the client is the owner of the property the contract will cover.
- **IPFS Hash** - This fields holds the link to a IPFS directory with the images of the property at the time of contract issuance
- **Date Time** - This field holds the date and time at which the transaction was signed and submitted to the network
- **Insurance Company Signature** - This field holds the signature of the insurance company issuing the transaction and submitting the 'Insurance Contract' creation request

Once the contract is created, the 'Client' user must use the 'Update the Client Signature' option to add their signature to the insurance contract, making it valid.

An insurance contract can be updated, but only by the insurance company. The only field which is allowed to be updated is the 'IPFS Hash'. Once it is updated, the 'Date Time' field also is updated to reflect the time of the modification. The insurance company must submit a new signature along with the updated fields, and the client' signature is be deleted. While the client doesn't update their signature, the contract is considered invalid. Once the client signature is added back to the contract, the contract becomes valid again. Figure 4.5 demonstrates the Insurance Contract creation/update process which happens to be the same.

To check an 'Insurance Contract', all that's needed to input is the target identity DID. The data of all the insurance contracts associated to that entity are then retrieved and displayed. It is here where the Fabric's MSP certificate/private key pair comes into play and becomes important.As previously mentioned, since there is no content being submitted in a 'Read' operation, no signature can be added to verify the transaction issuer. It is here that the MSP identity becomes crucial. The app will use the previously added private key (associated to the public key in the certificate issued by the MSP) to sign the transaction itself and send it to the network. Inside the chaincode, the system checks who issued the transaction. If the issuer of the transaction matches the target identity DID, the insurance contracts associated with that entity are retrieved and displayed. If not, the request fails.

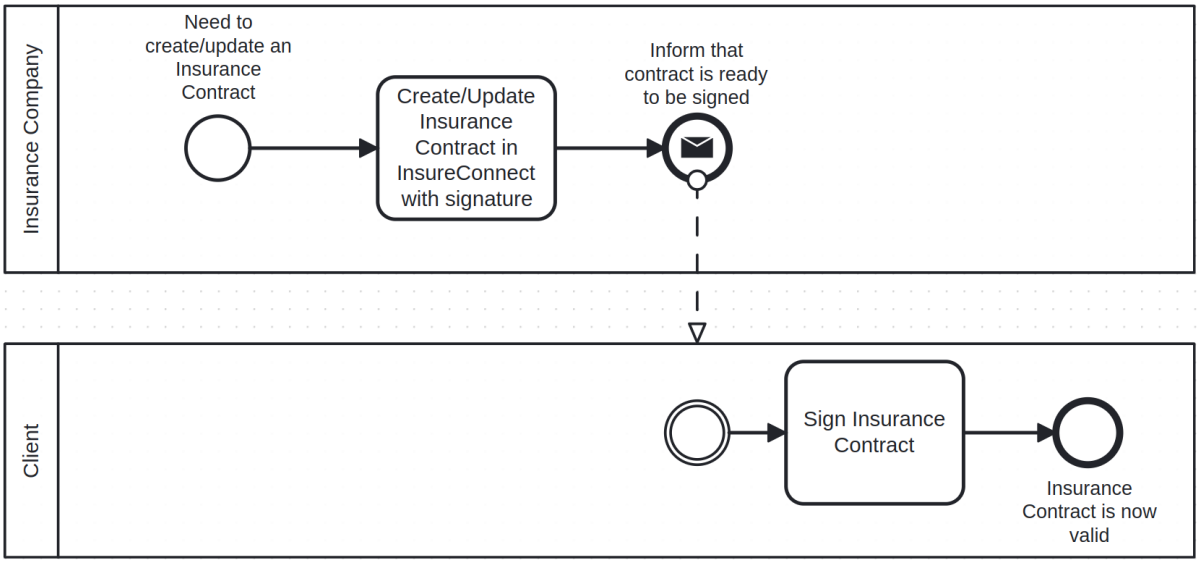


Figure 4.5: BPMN diagram displaying insurance contract creation/update.

4.3.3 Claims Management

Within this menu there are three further options: 'Create a Claim', 'Update a Claim', 'Assign an Auditor', and 'Check Claims'.

To create an 'Claim', the following fields must be submitted:

- **Claim ID** - This field holds ID of the claim
- **Insurance Contract ID** - This field holds the ID of the insurance contract the claim will be covered by
- **IPFS Hash** - This fields holds the link to a IPFS directory with the images of the property at the time of claim issuance
- **Date Time** - This field holds the date and time at which the transaction was signed and submitted to the network
- **Contents Signature** - This field holds the signature of either the insurance company or the client depending who issued the transaction and submitted the 'Claim' creation request

The claim can be created by either the client or the insurance company. The fields 'Claim State' and 'Auditor' are not filled during the creation, since they will be automatically filled in the object by the chaincode. By default, the claim's initial state will be 'PRESENTED,' and the 'Auditor' will be set to 'null'. The starting stage must also be included in the signature.

After the claim has been created, both the client and the insurance company can use the 'Update a Claim' option to update the claim. Additionally, the insurance company can assign an auditor to the claim. The only editable fields are 'Claim State' and 'Auditor', respectively for each function. The new updated claim state must progress for the claim to be handled and finish. Again, when either claim state or the auditor are updated, a new signature must be submitted by the one who updated it, along with the new updated contents. Figure 4.6 demonstrates a claim's lifecycle.

To check a 'Claim', all that's needed to input is the target identity DID. The logic is the same as with the 'Check Insurance Contracts' option in the 'Identity Management' menu. The user inputs the target DID they want to check, the app signs the transaction, and if both the target DID and the author of the transaction's signature match, the request is accepted, and the claim is retrieved and displayed.

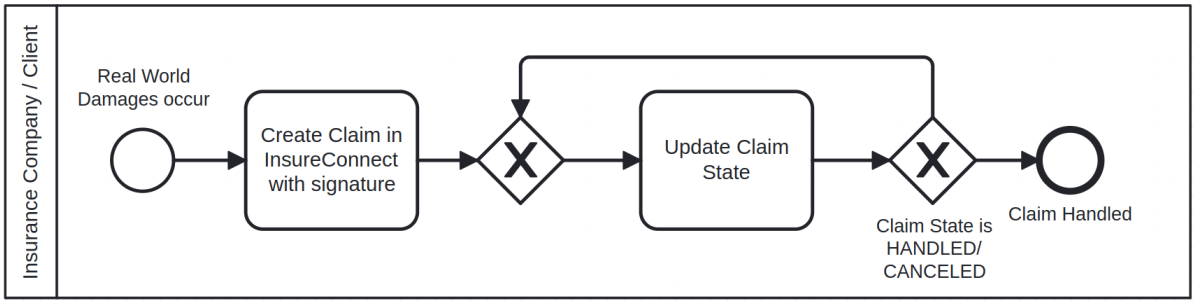


Figure 4.6: BPMN graphic displaying the lifecycle of a claim.

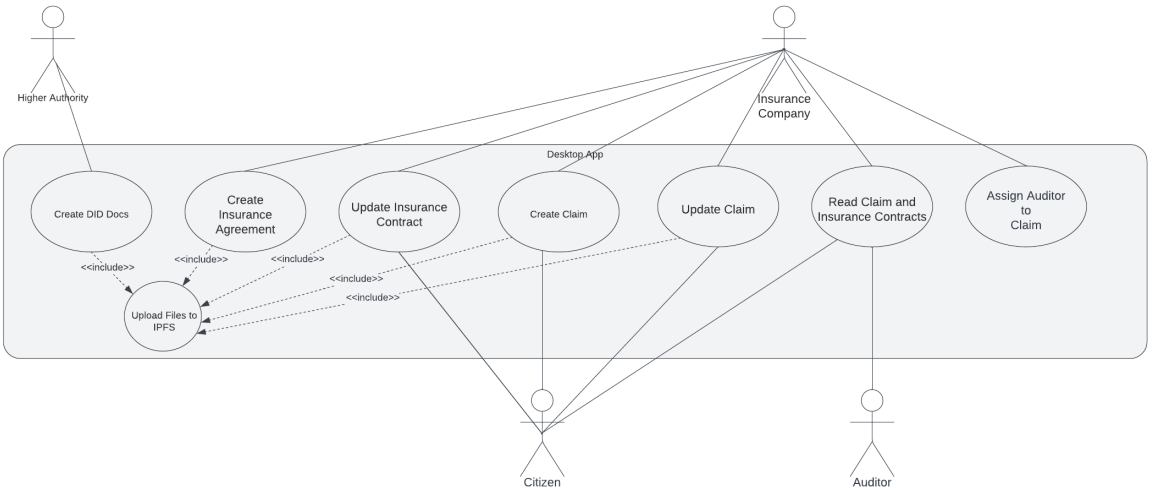


Figure 4.7: UML InsureConnect's Operations Diagram

4.4 Solution Recap

This chapter presented the InsureConnect system, whose objective is to establish a well-documented insurance process from beginning to end. From the moment the entities are created in the platform to the moment a claim has finished its lifecycle.

The chapter begins by establishing the roles and participants - Client, Insurance Company, Auditor, and Higher Authority - while explaining their access permissions. Following that, the architecture is laid out, detailing the active components involved: Hyperledger Fabric, IPFS , the desktop app and the possible nodes for hosting the network. Each component is then explained in detail to justify every decision that impacted the final product. Such decision were, for example, the chosen consensus mechanism, Raft, the need for two different public/private keypairs, why are RSA signatures utilized instead of others, and why was a desktop app developed rather than an online web interface.

To conclude this chapter, all of InsureConnect's operations are described to provide insights into the features offered by the desktop app. The different objects lifecycles are also presented.

5

Evaluation

Contents

This chapter delves on the evaluation performed on the system prototype and the test environment is presented to evaluate the performance of InsureConnect. Blockchain architecture, software version and configurations will be covered. The testing methodology is explained and afterwards the performance data is analysed to explore aspects such as the latency and throughput of the system based on different request loads.

5.1 Experimental Setup

Figure 5.1 presents the Hyperledger Fabric network prototype infrastructure deployed in a single computer. The deployment is made using a custom script to launch the nodes and deploy the smart contract.

The load testing experiments were performed on a Intel Omen with an Intel® Core™ i5-8300H, 8GB of RAM and 74GB disk capacity (dual boot computer) running Ubuntu 22.04.4 LTS. The main objective was to assess the performance and scalability, regarding loads, of the system.

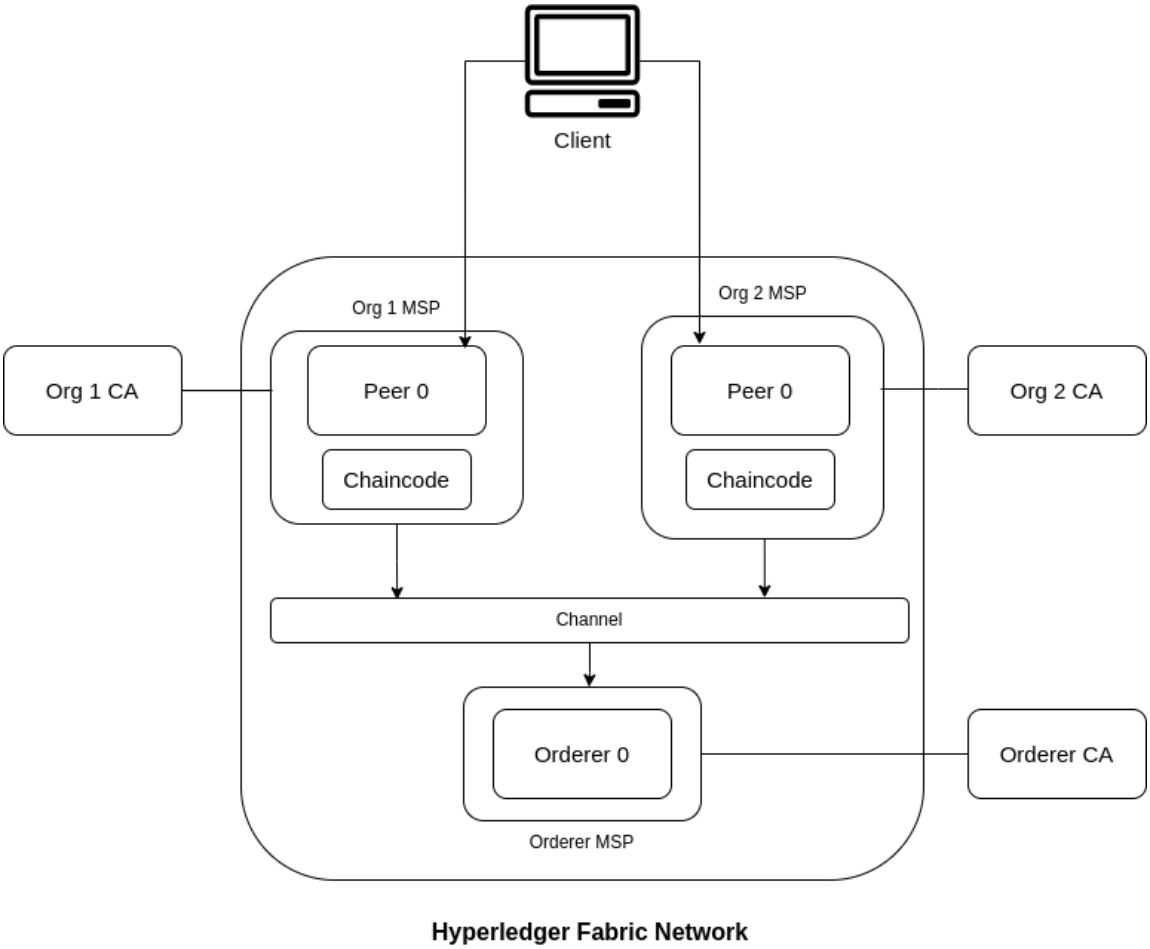


Figure 5.1: Test Hyperledger Fabric network infrastructure

The details component involved in the system are listed bellow. The testing environment configurations are based on a local docker Fabric execution.

- Each peer and orderer node in the Hyperledger Fabric network was setup with Hyperledger Fabric v2.5.6
- The Certificate Authority (CA) was setup with v1.5.9
- Each required component (peers, orderers and CAs) ran separately in their own docker container each
- Each component has no RAM limitation. The RAM is managed and assigned by the operating system
- Hyperledger Fabric CouchDB was used as the state database for each component and was setup with v3.3.2. When launched it was always completely empty
- There was one smart contract deployed which handled all transactions. When deployed the first action it performed was to write the DID Document of the 'HigherAuthority' in the ledger

5.2 Methodology

The tests were conducted by running a custom written Java program. The program ensures the creation of N threads, depending how many requests are required to be sent. From the peer nodes point of view, each thread is a different client with a different connection. Each thread will compute the amount of time it is required to sign the transaction contents, establishing the transaction and receiving a response. All the recorded times are stored in a '.csv' file for later usage and analysis..

5.3 Experimental Results

The evaluation focused on measuring how the system performs under varying loads of concurrent requests. The performance of five functions was tested, each subject to request loads ranging from 50 to 3000 concurrent requests.

The following data presents the findings related to the performance of each individual tested function, highlighting their strengths and weaknesses when faced with an increasing number of concurrent requests. All the chosen functions to be tested were write operation functions, specifically: 'Create DID Documents', 'Create Insurance Contract', 'Update Insurance Contract Signature', 'Create Claim', and 'Update Claim'. These were chosen because write functions are the ones which really pressure the system and force it to perform the most heavy operations. Inside each write function the read functions are called for relevant data lookup, so every aspect is effectively involved in the testing. This data will serve as a foundation to optimise the system for read-world scenarios with a dynamic rate of requests.

Across all tests, a consistent trend emerged: as the number of concurrent requests increased, both latency and throughput increased as well. However, throughput eventually showed diminishing increments and, along with this, dropped connections became more frequent due to the high volume of concurrent requests and limited resources. The percentage of dropped connections is displayed in Table 5.1.

The conducted tests do not surpass the 3000 concurrent requests mark due to the computer crashing, which resulted in either results that did not make any sense or by dropping all connections altogether.

Figure 5.2 displays both latency and throughput increasing together for function 'Create DID Document', but a subtle difference exists. While latency keeps growing mostly linearly in relation to the number of concurrent requests, throughput is starting to halt. This behaviour is completely normal and is expected. As more concurrent requests arrive and the system resources start to get more and more utilized, the time it takes to process each one increases, leading to higher latency and reduced throughput.

The results of Figure 5.3 for the function 'Create Insurance Contract' are the first to show dropped connections for the 2500 and 3000 concurrent requests. The percentage of dropped

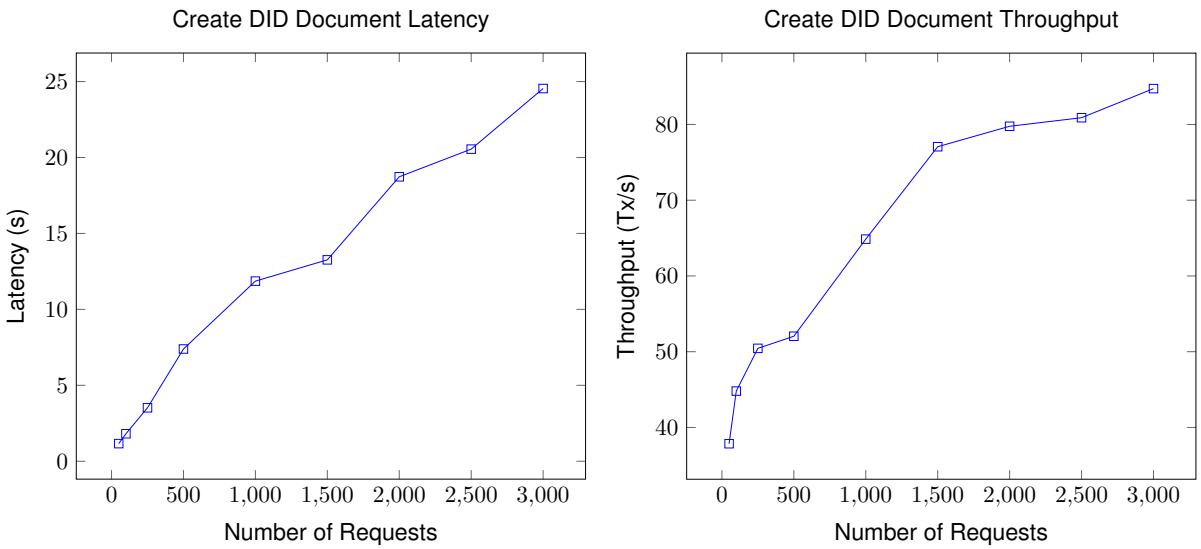


Figure 5.2: Latency & Throughput of Create DID Documents Function

connections is displayed in Table 5.1. By analysing the data, we can observe that while latency increases somewhat linearly, throughput keeps increasing until 1500 requests, after which it drops and basically stagnates. This showcases good performance, as even with a really high concurrent load, the system keeps up and does not decrease the throughput.

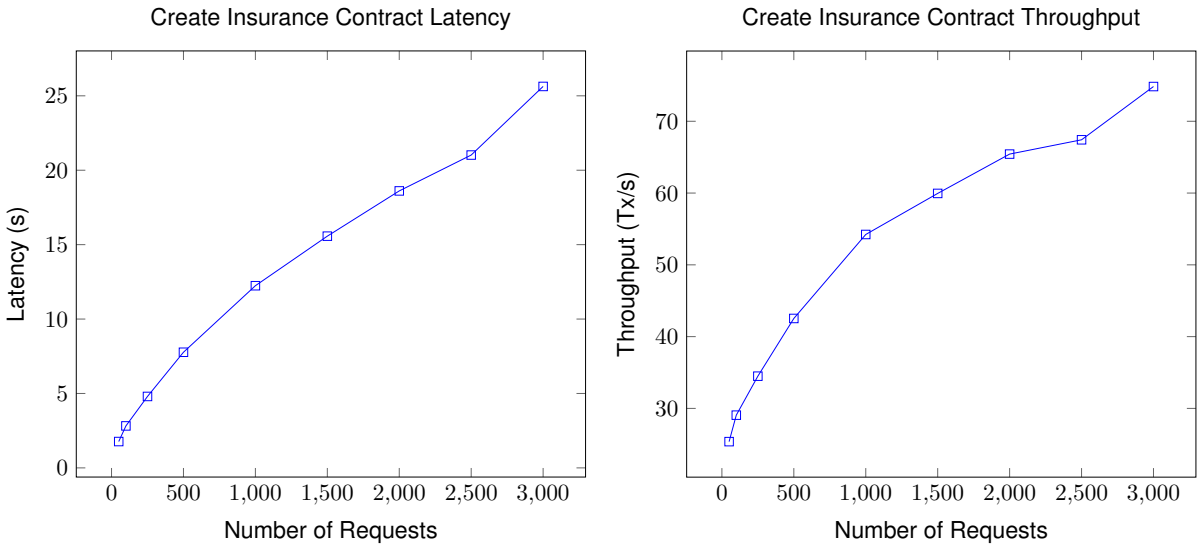


Figure 5.3: Latency & Throughput of Create Insurance Contract Function

Figure 5.4 showcases the 'Update Insurance Contract Signature' function results having a similar behaviour to those of Figure 5.3. Latency varies from 1.22 seconds for 50 requests to 15.44 seconds for 3000 requests. Throughput also keeps increasing until 1000 requests displaying good performance, and stagnating until 3000 requests, where it experiences a slight increase.

Continuing the trend, the 'Create Claim' function results present somewhat similar results presented in Figure 5.5, with the latency somewhat linearly increasing, starting with 1.83 sec-

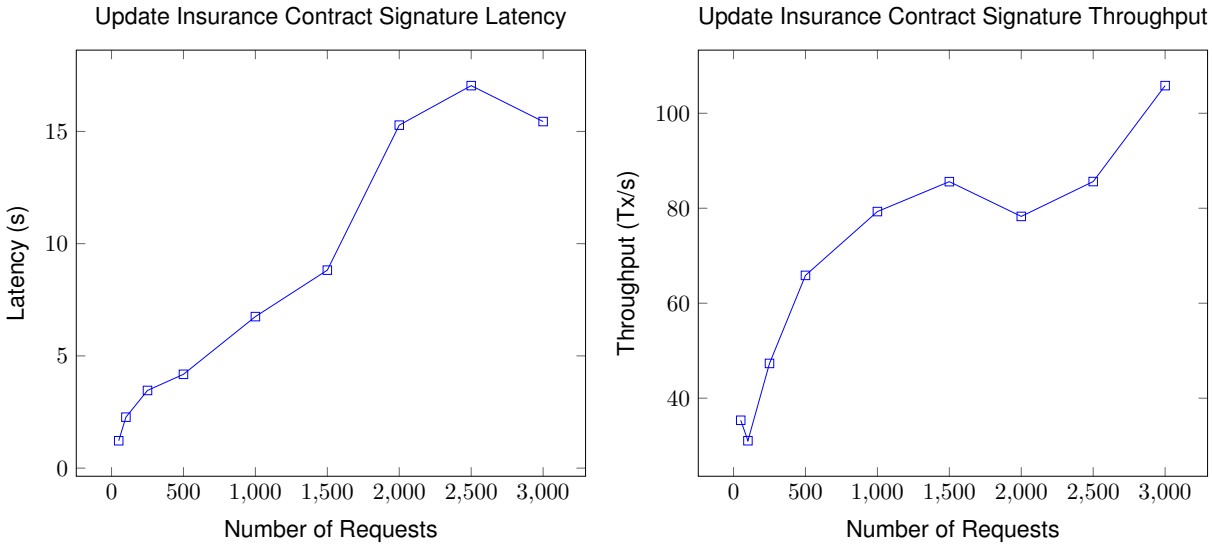


Figure 5.4: Latency & Throughput of Update Insurance Contract Signature Function

onds for 50 requests and finishing with 24.50 seconds for 3000 requests. The throughput, meanwhile, reinforces the good performance of the system by not decreasing and only stagnating even while the concurrent requests keep increasing.

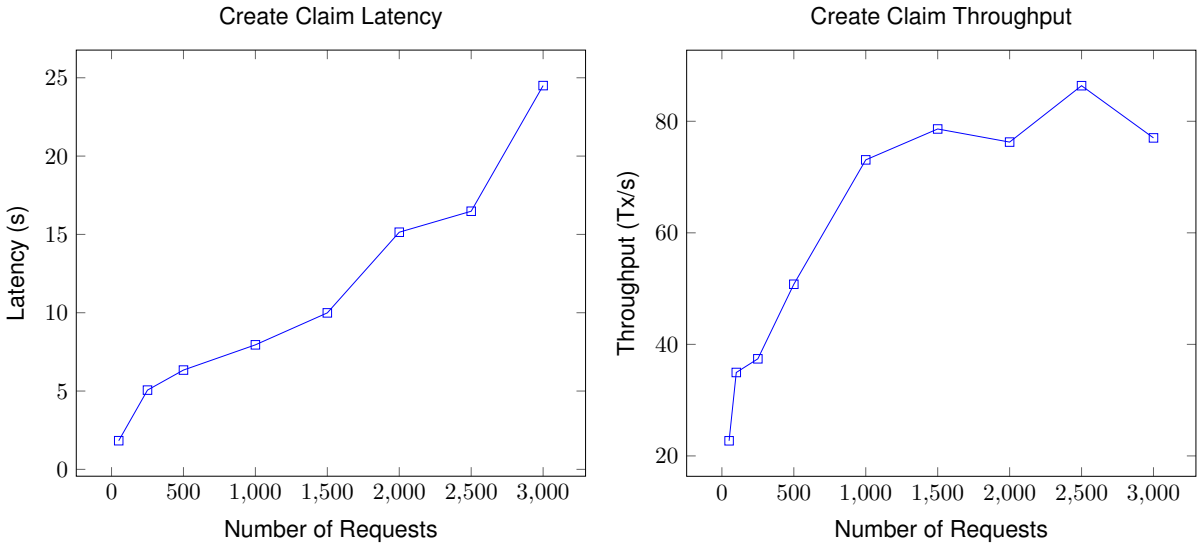


Figure 5.5: Latency & Throughput of Create Claim Function

Figure 5.6 provides information on the 'Update Claim' function. The latency varies from 1.40 for 50 requests and 14.21 seconds for 3000 requests. Throughput keeps increasing indicating good performance almost stagnating but with a last spike for 3000 requests.

As expected, the results demonstrate that as concurrent requests increase, the system's latency also increases. The throughput also usually behaves in a mostly expected way by rapidly increasing until it starts to stagnate due to system resources exhaustion, leading to a slower processing and response time for the requests. It is also worth noting that for every function except 'Create DID Documents', some connections get dropped around the 2500 -

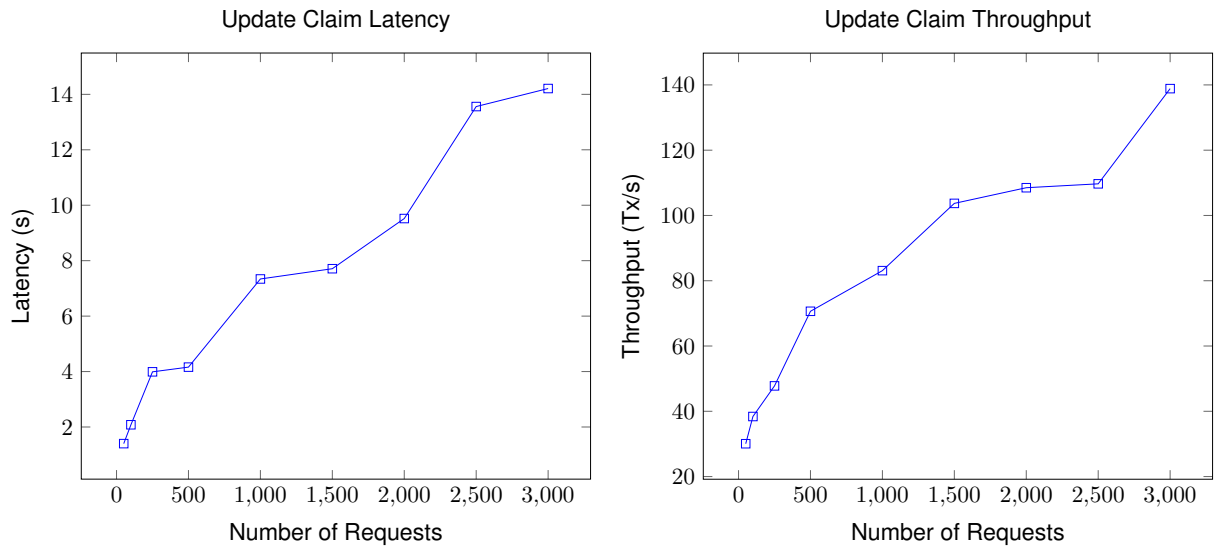


Figure 5.6: Latency & Throughput of Update Claim Function

3000 requests mark. The increase in latency with a higher concurrent number of requests can be attributed to the system's finite resources being spread across multiple requests, resulting in processing delays and longer response times. While this stands true, it is also important to point out that even as response times increase, the increment is not enough to completely halt the throughput of the system as we can see that in some functions the throughput also increases with the maximum test of 3000 concurrent requests. The system as a whole whole ceases to function when the concurrent requests exceed 3000, due to limitation of the physical machine it is running on. A comprehensive overview of this system's malfunctions across multiple functions is displayed in Table 5.1 with the error percentage of dropped connections. As can be seen, while dropped connections do occur, their rate is so low that it does not significantly impact the dataset, and almost no real-world users would be affected by this issue.

Number of Requests	Create DID Documents (%)	Create Insurance Contract (%)	Update Insurance Contract Signature (%)	Create Claim (%)	Update Claim (%)
50	0	0	0	0	0
100	0	0	0	0	0
250	0	0	0	0	0
500	0	0	0	0	0
1000	0	0	0	0	0
1500	0	0	0	0	0
2000	0	0	0	0	0
2500	0	0.12	0.12	0.16	0.2
3000	0	0.16	0.16	0.2	0.3

Table 5.1: Performance metrics based on the number of requests

6

Conclusion

Contents

In this chapter, a conclusion to all the performed work is presented. By summarizing the outcomes and reflecting on the whole process, I aim to present a clear and concise overview of the solution. This is the final chapter of my thesis.

6.1 Conclusion

This thesis presents a solution for the insurance market that eases the process of registering insurance contracts and filing new claims with the security and confidence of blockchain technology and digital signatures. The solution employs a fully decentralized system by incorporating both a blockchain for record-keeping and a decentralized storage system for file storage. Satellite imagery is utilized to record the different states of property damages at different time points.

The system was developed using the private permissioned Hyperledger Fabric blockchain, which allows for identity registration for each different entity using the system and IPFS decentralized storage because of its content-addressed lookup based on the files hashes so, even if the files are lost in IPFS, they can be re-hashed to prove their authenticity.

The blockchain stores three types of assets: Decentralized Identifier Documents, Insurance Contracts, and Damage Claims. Based on the insurance market landscape, three related parties were defined: the Client, the Insurance Company and the Auditor - along with an additional one for the solution itself: the Higher Authority. Each of these has its own function permissions they are allowed to perform. The Higher Authority is responsible for creating insurance companies DID documents. Insurance companies create DID Documents for both their clients and required auditors who are not already registered, and create insurance contracts. Clients sign contracts. Damage Claims can be created and updated by both insurance companies and clients. Auditors have read-only access to the claims and respective insurance contracts they are assigned to.

In order to ensure transaction authenticity in both, write and read operations, Decentralized Identifier Documents and Hyperledger Fabric's Membership Service Provider (MSP) are utilized. Two key/pairs are required to fully utilize the solution as intended. DID documents store the key to verify the transactions contents signature and, consequently, the identity of the submitter. Fabric's MSP stores the key to validate the signed transaction for read operations. The transaction itself is signed, and the peer nodes consult the respective certificate authority where the user's certificate is stored.

All of the functionalities are available to every user in the desktop app. This reinforces the fact that there is no login portal, and every authentication step is done in every transaction that is performed on the blockchain itself. Users must upload their both their credentials to the desktop app.

To evaluate the developed system, latency, throughput and error rates were measured. The solution was deployed on a local machine with limited resources. Each node was run inside a docker container without resources limitations. The results were very positive, with tests showing consistent trends: both latency and throughput increased as the number of concurrent requests grew. Latency showed a very linear increase while throughput kept a steady increase rate until . Sadly the system could not be saturated to test its limits do due resources limitation.

To evaluate the developed system, latency, throughput, and error rates were measured. The solution was deployed on a local machine with limited resources. Each node running inside a Docker container without resource limitations. The results were very positive, with tests showing consistent trends: both latency and throughput increased as the number of concurrent requests grew. Latency showed only a minimal increase, while throughput maintained a steady growth rate with some little stagnation for the higher number of requests. Unfortunately, the system could not be completely saturated to test its limits due to resource constraints.

In summary, the described solution was developed to easy and create a more transparent environment for all involved parties. It ensures that all parties have access to the necessary information to properly performed their actions. The blockchain technology provides a clear-cut history of every transaction without being unknowingly modified. Every transaction is authenticated with digital signatures.

No one can deny signed immutable and permanent data entries. All are held accountable.

6.2 Future Work

Currently, the system is deployed by running a script that launches Docker containers locally. The first step would be to evolve the system into a Kubernetes network while utilizing Infrastructure as a Service (IaaS) to accommodate the kubernetes pods. This would not only ease the launch process, but also provide a load balancer to the network, along with more dedicated CPU and resources.

Focusing on the Decentralized Identifier Documents, a potential step-up for this project would be to integrate it with Hyperledger Indy. Hyperledger Indy is a blockchain platform designed to provide tools, libraries, and reusable components for creating digital identities rooted in blockchain technology. It is interoperable with other blockchains. This integration would eliminate the need to store the DID documents themselves on the InsureConnect's blockchain, resulting in a more cohesive solution.

Furthermore and to finish, an integration with an European Blockchain Services Infrastructure (EBSI) compatible wallet, such as Walt.id [Walt ID, 2024], would really benefit the project for DID creation and VC's issuance, giving it more trustworthiness and reliability both in the European space and all over the world. Integrating with EBSI would heavily reduce the storage requirements on our blockchain, as DID Documents would no longer need to be directly stored in our ledger. All VCs would also be easily validated with a high variable possible types of VCs. The possibility of off loading VC verification would really benefit this project, as changing the possible VC types currently requires updating the chaincode on every node to ensure compatible responses (a hard fork).

Bibliography

[mas,] *Mastering Ethereum: Building Smart Contracts and DApps*.

[Cha, 2023] (2023). Blockchain 2023: Trends and developments in Sweden.

[hyp, 2024] (2024). Hyperledger fabric documentation. Accessed: 2024-10-08.

[Adams and Lloyd, 1999] Adams, C. and Lloyd, S. (1999). *Understanding public-key infrastructure: concepts, standards, and deployment considerations*. Sams Publishing.

[Androulaki et al., 2018] Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., De Caro, A., Enyeart, D., Ferris, C., Laventman, G., Manevich, Y., Muralidharan, S., Murthy, C., Nguyen, B., Sethi, M., Singh, G., Smith, K., Sorniotti, A., Stathakopoulou, C., Vukolić, M., Cocco, S. W., and Yellick, J. (2018). Hyperledger fabric: A distributed operating system for permissioned blockchains. In *Proceedings of the Thirteenth EuroSys Conference*, EuroSys '18, New York, NY, USA. Association for Computing Machinery.

[Asia, 2017] Asia, N. (2017). Japan to tidy up scattered property records.

[Balcão Único do Prédio (BUPi), 2024] Balcão Único do Prédio (BUPi) (2024). Balcão Único do prédio (bupi). Accessed: 2024-08-25.

[Baliga, 2017] Baliga, A. (2017). Understanding blockchain consensus models. *Persistent*, 4(1):14.

[Benet, 2014] Benet, J. (2014). Ipfs-content addressed, versioned, p2p file system. *arXiv preprint arXiv:1407.3561*.

[Borgen Project, 2021] Borgen Project (2021). Blockchain-based land registry.

[Brunner et al., 2020] Brunner, C., Gallersdörfer, U., Knirsch, F., Engel, D., and Matthes, F. (2020). Did and vc: Untangling decentralized identifiers and verifiable credentials for the web of trust. In *Proceedings of the 2020 3rd International Conference on Blockchain Technology and Applications*, pages 61–66.

[Buterin et al., 2014] Buterin, V. et al. (2014). Ethereum white paper: a next generation smart contract & decentralized application platform. *First version*, 53.

[Chen et al., 2020] Chen, H., Pendleton, M., Njilla, L., and Xu, S. (2020). A survey on ethereum systems security: Vulnerabilities, attacks, and defenses. *ACM Comput. Surv.*, 53(3).

[Cohen, 2003] Cohen, B. (2003). Incentives build robustness in bittorrent. In *Workshop on Economics of Peer-to-Peer systems*, volume 6, pages 68–72. Berkeley, CA, USA.

[Dhillon et al., 2017] Dhillon, V., Metcalf, D., and Hooper, M. (2017). *The Hyperledger Project*, pages 139–149. Apress, Berkeley, CA.

[Dinh et al., 2018] Dinh, T. T. A., Liu, R., Zhang, M., Chen, G., Ooi, B. C., and Wang, J. (2018). Untangling blockchain: A data processing view of blockchain systems. *IEEE Transactions on Knowledge and Data Engineering*, 30(7):1366–1385.

[EOS,] EOS. Eos - earth observing system.

[European Commission,] European Commission. European blockchain services infrastructure (ebsi).

[Gilbert and Lynch, 2012] Gilbert, S. and Lynch, N. (2012). Perspectives on the cap theorem. *Computer*, 45(2):30–36.

[Government of India, Ministry of Electronics and Information Technology, 2020] Government of India, Ministry of Electronics and Information Technology (2020). Land registration using blockchain.

[Hardman, 2016] Hardman, C. A. (2016). The path to self-sovereign identity. *Life With Alacrity*.

[Heiligenstein, 2023] Heiligenstein, M. X. (2023). Facebook data breach timeline. Accessed on 2023-10-02.

[Helliar et al., 2020] Helliar, C. V., Crawford, L., Rocca, L., Teodori, C., and Veneziani, M. (2020). Permissionless and permissioned blockchain diffusion. *International Journal of Information Management*, 54:102136.

[Hunt et al., 2010] Hunt, P., Konar, M., Junqueira, F. P., and Reed, B. (2010). {ZooKeeper}: Wait-free coordination for internet-scale systems. In *2010 USENIX Annual Technical Conference (USENIX ATC 10)*.

[Hyperledger Fabric, 2024] Hyperledger Fabric (2024). Hyperledger fabric chaincode java compatibility guide. Accessed: 2024-08-19.

[International Council for Accreditation of IT Managers, 2016] International Council for Accreditation of IT Managers (2016). Blockchain and land registry.

[Khan et al., 2020] Khan, R., Ansari, S., Jain, S., and Sachdeva, S. (2020). Blockchain based land registry system using Ethereum blockchain. *Researchgate. Net*, 12:3640–3648.

[Lemieux et al., 2018] Lemieux, V., Lacombe, C., and Flores, D. (2018). Title and code: Real estate transaction recording in the blockchain in brazil (rcplac-01). *Case Study*, 1.

[Levy and Silberschatz, 1990] Levy, E. and Silberschatz, A. (1990). Distributed file systems: Concepts and examples. *ACM Computing Surveys (CSUR)*, 22(4):321–374.

[Nakamoto, 2009] Nakamoto, S. (2009). Bitcoin: A peer-to-peer electronic cash system.

[NASA,] NASA. Nasa worldview.

[Nofer et al., 2017] Nofer, M., Gomber, P., Hinz, O., and Schiereck, D. (2017). Blockchain. *Business & Information Systems Engineering*, 59:183–187.

[not specified, 2015] not specified, A. (2015). Blockchain land title project stalls in Honduras. https://www.coindesk.com/markets/2015/12/26/blockchain-land-title-project-stalls-in-honduras/.

[Ongaro and Ousterhout, 2014] Ongaro, D. and Ousterhout, J. (2014). In search of an understandable consensus algorithm. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, pages 305–319.

[Peck, 2017] Peck, M. E. (2017). Blockchains: How they work and why they'll change the world. *IEEE Spectrum*, 54(10):26–35.

[Pierro and Rocha, 2019] Pierro, G. A. and Rocha, H. (2019). The influence factors on ethereum transaction fees. In *2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*, pages 24–31. IEEE.

[Reed et al., 2020] Reed, D., Sporny, M., Longley, D., Allen, C., Grant, R., Sabadello, M., and Holt, J. (2020). Decentralized identifiers (dids) v1. 0. *Draft Community Group Report*.

[Sahai and Pandey, 2020] Sahai, A. and Pandey, R. (2020). Smart contract definition for land registry in blockchain. In *2020 IEEE 9th International conference on communication systems and network technologies (CSNT)*, pages 230–235. IEEE.

[Shuaib et al., 2020] Shuaib, M., Daud, S. M., Alam, S., and Khan, W. Z. (2020). Blockchain-based framework for secure and reliable land registry system. *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, 18(5):2560–2571.

[Swarm Team, 2020] Swarm Team (2020). Swarm: The decentralized storage and communication system for a sovereign digital society. https://www.ethswarm.org/swarm-whitepaper.pdf. Accessed: 2024-08-16.

[Vranken, 2017] Vranken, H. (2017). Sustainability of bitcoin and blockchains. *Current opinion in environmental sustainability*, 28:1–9.

[Vujičić et al., 2018] Vujičić, D., Jagodić, D., and Ranđić, S. (2018). Blockchain technology, bitcoin, and ethereum: A brief overview. In *2018 17th International Symposium Infoteh-Jahorina (INFOTEH)*, pages 1–6. IEEE.

[Walt ID, 2024] Walt ID (2024). Walt id. Accessed: 2024-09-17.

[World Wide Web Consortium (W3C), 2023] World Wide Web Consortium (W3C) (2023). Verifiable Credentials Data Model v2.0. https://www.w3.org/TR/vc-data-model-2.0/#what-is-a-verifiable-credential. Accessed: 2024-09-20.

[Yaga et al., 2019] Yaga, D., Mell, P., Roby, N., and Scarfone, K. (2019). Blockchain technology overview. *arXiv preprint arXiv:1906.11078*.