UNIVERSIDADE DE ÉVORA


**ESCOLA DE CIÊNCIAS E TECNOLOGIAS**

DEPARTAMENTO DE INFORMÁTICA


# TÍTULO | Secure Framework for Cloud Data Sharing based on Blockchain


Nome do Mestrando | André Figueira

Orientação | Pedro Salgueiro


**Mestrado em Engenharia Informática**

Dissertação


Évora, 2018

UNIVERSIDADE DE ÉVORA


**ESCOLA DE CIÊNCIAS E TECNOLOGIAS**

DEPARTAMENTO DE INFORMÁTICA


# TÍTULO | Secure Framework for Cloud Data Sharing based on Blockchain


Nome do Mestrando | André Figueira

Orientação | Pedro Salgueiro


**Mestrado em Engenharia Informática**

Dissertação


Évora, 2018

*I dedicated this to my Family.*

# Preface

Little over a year ago I started this dissertation and from the very beginning I had in mind how complex it could be, however, I was determined to learn more about Blockchain and everything about it and as such I accepted the challenge. Every chapter in this dissertation is the result of many days, weeks, months of sacrifice and dedication in research and development on something I was not certain I could accomplish.

From this hard work and determination I hope that you, the reader, will find it as interesting as I did from the very beginning to the very end.

# Acknowledgements

Firstly, I would like to thank Professor Pedro Salgueiro, my dissertation advisor, for providing constant motivation, guidance, knowledge and support from the very beginning until the end, as well as the patience. Without his contribution this dissertation would certainly not have been possible. I would also like to thank him for providing me with this topic which during the development of this dissertation determined my professional career goals, which is to be centered on Distributed Ledger Technology, notably Blockchain.

I would like to thank my parents for providing me with their support during the development process of this dissertation and well as during my academic progress until this point.

Finally, i would like to thank my friends and colleagues who listened, advised and challenged me during the development of this dissertation.

# Contents

# List of Figures

# List of Tables

# Acronyms

**HF** Hyperledger Fabric

**CA** Certificate Authority

**DAPP** Decentralized Application

**DAO** Decentralized Autonomous Organization

**BTC** Bitcoin

**ETH** Ethereum/Ether

**UID** Unique Identifier

**ID** Identifier

**PoW** Proof of Work

**PoS** Proof of Stake

**DPoS** Delegated Proof of Stake

**BFT** Byzantine Fault Tolerance

**GPU** Graphics Processing Unit

**ASIC** Application-Specific Integrated Circuit

**CPU** Central Processing Unit

**API** Application Programming Interface

**SDK** Software Developer Kit

**SHA** Secure Hash Algorithms

**URL** *Uniform Resource Locator*

**BBDS** Blockchain Based Data Sharing

MB Megabyte

GB Gigabyte

TB Terabyte

# Abstract

Blockchain is a relatively new and disruptive technology that is considered a distributed database working as a ledger, with the ability to facilitate the recording of transactions and tracking of assets. It is a growing list of records called blocks linked together by the hash of the previous block, where each block contains the most recent transactions in the network.

Smart contracts are agreements between entities that are written in code, and, when associated with Blockchain they will operate without interference, censorship or malicious intentions.

Data Sharing on Clouds is common but requires trust on third parties to ensure various aspects, such as security and privacy, but these are unknown aspects the owner has no controls over. The Cloud Storage providers control the data access and sharing over the data. Sharing data through third party services, using unknown methods is a delicate process regarding the privacy and security aspects. These two aspects are crucial points when it comes to personal and private data.

In this work, the concept of using Blockchain to create a Data Sharing mechanism is explored. This proof of concept explores how data access and permissions can be controlled using blockchain and smart contracts, by giving control to the owner and focusing on smart contracts and blockchain.

**Keywords:** Blockchain, Smart Contracts, Data Sharing, Privacy, Security

# Sumário

## Framework Segura para Partilha de Dados em Clouds sobre Blockchain

Blockchain é uma tecnologia relativamente nova e disruptiva, considerada uma base de dados distribuída e funcionando como um livro de registo. Tem a capacidade de facilitar o registo de transacções e de rastreamento de bens. É uma lista crescente de conjuntos de registos chamados blocos, ligados um aos outros através da *hash* do bloco anterior.

Os Contractos Inteligentes são acordos entre entidades escritas em código. Quando associadados à tecnologia Blockchain operam sem qualquer inteferência de terceiros, censura ou intenções maliciosas.

A partilha de dados na cloud é bastante comum mas requere confiança em terceiros para garantir que vários aspectos como segurança e privacidade são assegurados, mas estes são aspectos sobre quais o dono dos dados não tem controlo. A *cloud* tem controlo sobre o acesso e a partilha de dados, sendo que a partilha de dados por serviços de terceiros é um processo delicado quando se refere a privacidade e a segurança desses dados. Estes são aspectos que são cruciais quando se refere a informação pessoal e confidencial

Neste trabalho, o conceito de blockchain para criar um mecanismo de partilha de dados é explorado. Esta prova de conceito explora como a partilha de dados e o controlo de acesso pode ser executados usando blockchain e contractos inteligentes. Isto dando controlo ao dono dos dados e focos em blockchain e contractos inteligentes. Permitindo que o dono dos dados seja responsável pelos seus dados.

**Palavras chave:** Blockchain, Contractos Inteligentes, Segurança, Privacidade, Partilha de Dados

# 1

# Introduction

*In this Chapter we provide an introduction to the topic of this dissertation, where an introduction to blockchain data sharing, motivation for the project, opportunities and objectives and the structure of this dissertation are described.*

Blockchain is a disruptive and relatively new technology that is increasing in popularity. Using this technology will be a central component in the work described in this dissertation. The addition of Blockchain and smart contracts, it can be a viable solution to the problem of data privacy, access control and sharing of data. And by taking advantage of already existing blockchain framework implementations, more notably, Hyperledger Fabric which is a dedicated business blockchain, can be a great addition to the proposed topic, allowing the implementation of a secure data sharing mechanism and methods to control the rules and policies of data access in the possible way.

Data in the blockchain is tamper proof and records are kept as long as the network persists and using smart contracts which thrive in the blockchain technology being uncensored agreements and cannot be interfered by third parties. This makes an ideal environment for a data sharing mechanism, access control and privacy of data which gives the owner of data control the decisions.

To explore, research and implement a proof of concept various topics were approached and clearly defined before tackling the data sharing mechanism, these are essentially Blockchain and Smart Contracts.

This dissertation is centered on the privacy and control aspects of data shared through cloud based service, aspects that are crucial when it comes to personal, private or confidential data which is of great concern, when dealing with small, individual users up to corporations and academic institutions who have to trust others to make sure these aspects are verified while not truly owning their data. The topic of this dissertation revolves around the study and implementation of a cloud based framework for data sharing based on Blockchain. The capabilities and properties of Blockchain may prove to be a strong and a viable solution to handle the security and data sharing of data and most importantly allow for a true sense of ownership and full control over data access and sharing.

## 1.1 Motivation

Data sharing on cloud platforms is very common and secure in modern times, but sharing on cloud platforms and relying on third parties to manage and maintain this data secure and private with unknown means of doing it, is not easy to accept. It requires a certain degree of trust to have on the third party, especially when it means to trust others entities with private, sensitive or confidential information while having third parties handle all the process in between. This leads to several issues including who can access data, who has accessed data, for how long can data be accessed. These are problems associated with trust, authorization and ownership of data that concern users among others.

### 1.1.1 Opportunities

The work presented in this dissertation presents various opportunities specially on corporate and academic levels. Sharing of information is complex due to the ease of replication, little control from owner of the data, where the storage provider controls everything although this is complex problem to tackle. The work presented in this dissertation can with the usage of blockchain and smart contracts create a data sharing mechanism that will give control over to the users who use the network, whatever application it may be used in. This work additionally, provides a tamper proof record of all interactions at any given point in time of every data that has been registered.

## 1.2 Objective

The objective of this work is to provide a tamper proof mechanism that can control the access and data sharing where the owner of the data has full control over his data. This mechanism is based on the blockchain using smart contracts to handle all the business logic in order to control and keep record of every access and data sharing on the data. With the main goals of this work are:

- Research Blockchain technology and Smart Contracts
- Research Hyperledger Fabric as the chosen Blockchain implementation for the project
- Research on a system to ensure the privacy and confidentiality of the data hosted on the storage provider.
- Implement prototypes for a data sharing mechanism based on Blockchain.
- Test and experimental evaluation of the developed prototypes.

Additionally there are secondary goals related to the applications that interact with the blockchain network in order to make a more stable proof of concept

- Implementation of a simple prototype for file hosting.
- Implementation of a simple prototype of an application to interact with the file host.
- Test and experimental evaluation of developed prototypes.

## 1.3 Structure

Chapter 2 corresponds to the state of the art of the topic, it presents the topics approach by the project itself providing a brief explanation of each component and these fit together.

Chapter 3 provides a clear theoretical explanation of the Blockchain, not just the a wide definition and general topics of it but a detailed approach of various components associated with this disruptive technology, from transactions structure to consensus protocols definition and structure to use cases. Chapter 4 describes smart contracts in a detailed manner since this component is essential to the development of the project.

Chapter 5 describe a network referenced multiple times in Chapter 3 and Chapter 4. This the blockchain implementation used by the project. Since it is complex, as all blockchain implementations are, and used for the development of this project in this chapter it is presented an not in depth overview of various components that make up this blockchain implementation.

Chapter 6 corresponds to all the decision process that was required to create a data sharing mechanism based on blockchain. It explains possible approaches, design choices, solutions, alternative solutions faced during the development of this project. Besides the architecture design process, describes the project itself, describing the architecture, smart contracts, explanation of data stored in the blockchain, how to interact with it, and how every component fits together to create a data sharing mechanism based on the blockchain.

Chapter 7 is the experimental evaluation, provides more technical explanations on the behaviour of the network. It is a series of test cases from various interactions with the network. This in a development environment.

Finally Chapter 8 presents conclusions and future work.

# 2

# State of the Art

*This chapter contains a revision of various technologies related to the work described in this dissertation. As such it is divided in a series of topics, representing the ones approached in this dissertation. The topics are blockchain technology, smart contracts and data sharing.*

## 2.1 Blockchain Technology

Blockchain is considered a distributed database working as a ledger, "*having the ability to facilitate the recording of transactions and tracking of assets.*" [Gup17]. It consists of individual blocks containing a set of transactions of some sort, where each block references the previous block, resulting in a chain of blocks, thus named a blockchain. The Blockchain holds information of tangible or intangible assets having the capability to reduce risks, increase visibility, faster automated processes in the process as opposed to other commonly used methods [Gup17]. Often described as decentralized, it is replicated across multiple peers in a network, thus the majority of the peers maintain the same copy of the blockchain, although, in some networks, peers may be prevented from holding a copy of the blockchain (private networks). The blockchain, since replicated to every peer makes it persistent to change, tamper proof, immutable and unable to be corrupted. As mentioned before the Blockchain is made by blocks, each block is a set of transactions of some sort that were checked and accepted by the network rules of consensus. Once a block is considered valid, peers add the block to their copy of

blockchain, which the majority of the peers have an equal copy of. This together with the chain that links blocks prevents the blockchain from being modified and any attempt to modify it is easily detected. In other words, the block and the transactions that are part of are no longer reversible and the change in the blockchain is final, the effect is permanent. The blockchain as it is replicated across multiple peers, the blockchain is always available, having no single point of failure, because every peer can answer as of being part of a peer to peer network. In terms of blockchain network, it can be public or private, depending on the situation [Lew15].

- Public Blockchain Network, any user can write and read data, interact with the blockchain and anyone can join the network, an example is the Bitcoin network.
- Private Blockchain Network: the users are assumed to be known but there are enforced restrictions to participants who can write and read data, maintain the blockchain, etc…

### 2.1.1 Background of Blockchain

The Blockchain was first implemented and conceptualized by *Satoshi Nakamoto*, a name used by an unknown person or group, to be used in Bitcoin, being its core component, also created by Satoshi Nakamoto. The blockchain was created essentially to be an immutable ledger to record transactions, "*becoming a chain of digital signatures that defines an electronic coin*" [Nak] , making it possible to create a decentralized digital currency (Bitcoin) allowing "*online payments to be sent directly from one party to another without going through a financial institution*" [Nak]. Becoming the key aspect in solving the problem of "double spending" [Nak], which is the idea of using the same amount more than once, a problem that is often associated with digital currencies.


Figure 2.1: A bitcoin blockchain block [Lew15].

Figure 2.1, by Antony Lewis *A Gentle Introduction to Blockchain Technology* [Lew15], shows an abstract view of the contents of a bitcoin blockchain block, it contains information from various transactions which have been confirmed by the network, the reward for the peer who successfully contributed in creating the block and a link which connects blocks together. In technical terms, a block header contains the transaction list is a Merkle Tree[1], the previous block hash, a timestamp[2] and a nonce[3].

[1] A tree whose peers are the hash of the child peers and the leafs are the hash of data
[2] Date and time value
[3] Random number to be found by the a peer of the network

## 2.2 Smart Contracts for Business Logic

Described by Nick Szabo, smart contracts can *"secure many algorithmically specifiable relationships from breach by principals, and from eavesdropping or malicious interference by third parties"*, this because many *kinds of contractual clauses (such as collateral, bonding, delineation of property rights, etc.) can be embedded in the hardware and software we deal with* [Sza97]

Smart contracts, often compared to vending machines, are applications, written in a programming language (Golang, Solidity, …) by users of the network, "*that run exactly as programmed without any possibility of downtime, censorship, fraud or third party interference*" [eth]. A smart contract has rules, conditions and penalties written in code, and when invoked, automatically determines what is the appropriate action according to the set of rules and conditions written in the contract for that given situation, thus enforcing, validating and verifying. With the support of the blockchain removes the need to rely on middleman to confirm the conditions and others processes of a contract, or having the fear of third party malicious actions interfering, as these contracts are automatically invoked when a specified situation is triggered. These contracts are stored in the blockchain or any other appropriate method and as such all data is stored within the blockchain and is immutable, increasing safety, visibility and allowing to minimize trust by reducing human judgments that could influence the transactions.

## 2.3 Private Blockchain Framework

Private blockchains are permissioned networks. These allows only registered and known users to interact with it and peers are considered trusted at some level. Due to the nature of being private, network size when comparing to a large scale blockchain network like Bitcoin it is to be considered smaller, therefore, the consensus mechanism may be different. Due to the restrictions and use cases of a private blockchain network these are ideal for business networks as every user should be known.

### 2.3.1 Hyperledger Fabric

Hyperledger Fabric is a private blockchain. This framework implementation is a platform for a distributed system of records, or a distributed ledger, allowing application with high confidentiality, scalability, consensus and security using its own blockchain implementation and own version of smart contracts (named chaincode). The Hyperledger Fabric allows private networks and "sub private" networks (named channels) of decentralized peers highly focused on consensus, but there is no proof of work. Consensus in this platform is only achieved in its entirety when certain policy criteria are checked [Hyp18] and certain peers approve it. A transaction in order to take place must be endorsed by certain peers, according to certain endorsement policies. After that, there should exist consensus among the peers who are allowed to create blocks. Thus, a very complex system of consensus and understanding among the peers must take place when either invoking or deploying a transaction. In the network, in general terms, there are two types of peers, validating peers which handles the verification of the transactions interacting with the blockchain and so its maintenance, and non-validating peers which act as a bridge ensuring communication between clients and validating peers and can, if allowed to, verify transactions able to endorse. [Cac16]

The Blockchain being the most important aspect of the Hyperledger Fabric, its purpose is holding state and ledger data, run chaincode and execute transactions [Hyp18]. Chaincode is Hyperledger Fabric version of smart contracts, handling the application and business logic (agreed by the network) and waiting to be invoked by a user. "*Chaincode initializes and manages ledger state through transactions submitted by applications*" [Hyp18], also known as blockchain applications. Transactions that undergoes in the blockchain are of two types, de-

ploy and invoke. Deploy transactions create chaincode and when successful await to be invoked. Invoked transactions, invoke chaincode, permits the client to execute functions from a previously deployed chaincode. Must be noted that not all invoke transactions will end up in a block, as transactions that simply read from the blockchain are different. Regarding the distributed ledger, State and Blockchain are data structure components of the ledger. State is the latest state of transactions, working in key/value pairs, it reflects only successful state transitions of transactions, this means, it contains the latest values for any given key, and can always be reconstructed via the Blockchain. Blockchain contains all the history of the state transitions of all transactions that took place, both invalid and succeeded, thus keeping a record of everything that undergoes in the network. The blockchain is a definite source of data as the State component can be constructed via the Blockchain.

## 2.4 Interaction with the Blockchain Network

In the context of Hyperledger Fabric, interaction with the blockchain network comes by the use of external programs. These programs allow users to connect to peers of the network enabling them to interact with the blockchain network. This, however, is only true if registered and accepted as a user of the network, which is achieved by communicating with the Certificate Authority within the network. Interaction with the blockchain is only allowed through the use of smart contracts, thus being only able to query and update the blockchain and getting responses from it. Besides interaction with the blockchain, in the context of this dissertation, it connects users to the storage server allowing coordination between the two components.

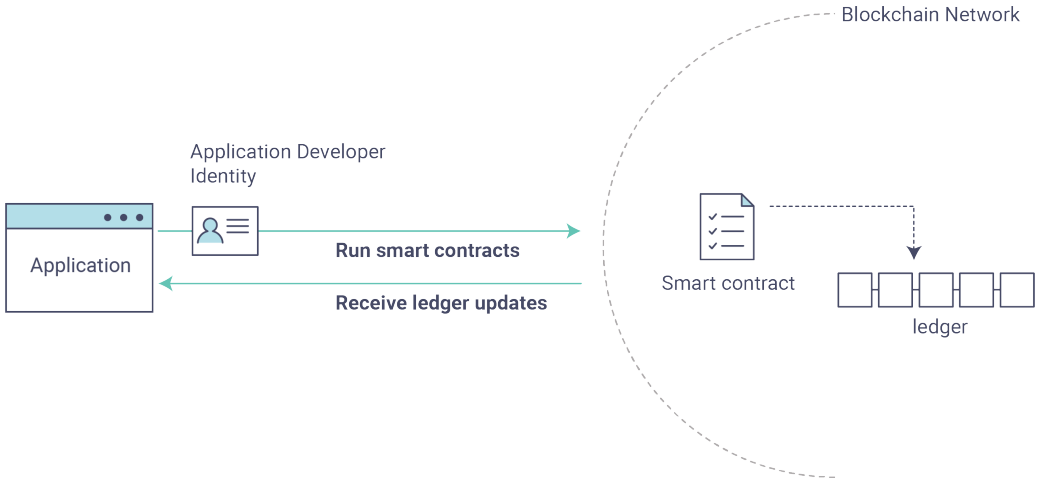



Figure 2.2: Application interacting with the Blockchain network. [Hyp18]

Figure 2.2, by *Hyperledger Fabric Official Documentation* [Hyp18], provides an abstract view of how these programs interact with the blockchain network. A step that must e noted is that in order to perform a request to the blockchain network to run smart contracts, it must come from a registered user within the network. This enables only known users to interact with the network.

**Example in the context of the work to be described:** The application displays the owned and/or shared files by requesting this information from the blockchain via smart contract and if the user wants to download a shared file, then this application would firstly use a smart contract function to determine if the user still has access to that specific file and if true would return an unique identifier so it can be used to request that file to the repository.

## 2.5 Data Sharing on Clouds

Data sharing on clouds is the ability of sharing data to another user, in which data is hosted in a remote storage provider with the properties of a cloud, most importantly high availability. The storage providers will then handle, in theory, the security, privacy and confidentiality of the data and handle access control and revocation of privileges. This raises issues of confidentiality and privacy due to the uncertainty of how data is treated and stored, complications increase if the cloud provider gets compromised, revealing the data and possibly methods to decrypt the data. With no owner control over these aspects, it is expected by users to take control over security of their own data [ADK+17] and provide enough safety that access to data can only come by the owner or any user with access. In other terms, the owner needs to have complete control.

The issue is keeping data secured and at the same providing a way to able to be shared by other users. There are multiple methods to solve this problem [ZRL+10, ADK+17, CCT+14].

Shared secret encryption allows a shared secret between owner and user. But limited to one time uses, as one leak of the secret and the data is no longer secured.

Asymmetric key encryption uses a set of private and public keys, in which the public key is shared by other users, and the private key remains a secret to the owner of the private key. Asymmetric encryption uses the other user public key to encrypt data and can only be decrypted by the corresponding user private key.

Several approaches on data sharing mechanism exists but require to have a certain amount of trust on the cloud repository in order to have, besides the owner's private key, a shared private key between repository and owner to enable data sharing [ZRL+10] or uses a trusted cryptography server [ADK+17]. These approaches do not require the use of re-encryption, but rely on additional factors.

Other approaches provide no sharing of private keys and no additional servers but essentially creates a new form of encryption that uses a master-key to extract other set of keys that must be securely sent to other users for sharing of data that does not require the use of a trusted server. [CCT+14]

GNU Privacy Guard is another method for file encryption with multiple recipients [gnu]. Uses of gnu privacy guard extends data privacy and secure communication between users, and additionally multiple e-mail programs support this method. [Gua] Additionally, it allows multi-key file encryption, which works by encrypting a symmetric key with the asymmetric keys of the recipients.

In Chapter 6 this approach is used as a possible setup for data sharing combined with blockchain, but was not implemented.

## 2.6 Data Sharing on Blockchain

Exploring blockchain data sharing presents a new set of difficulties, but presents a new set of opportunities. Data Sharing on Blockchain is the concept of using blockchain properties to share data with other entities by the means of smart contracts and other appropriate tools when the data is hosted on a remote storage repository. A mechanism as this allows the owner to keep track and himself control the permissions of users who can access data he owns, which uses identity and smart contracts to control and enforce the rules, which are unable to be interfered by third parties. This method creates an additional layer of security and privacy which is manageable by the owner of the data and the use of blockchain and smart contracts, instead solely by a cloud storage provider handling these aspects. Besides the smart contracts, blocks contain every transaction every made, this means any event regarding data can be easily inspected by authorized users: who accessed what data and when, and these logs can never be modified or deleted. It also allows to easily revoke any user access to data. [XSS+17, AEVL16, XSA+17]

Smart contracts provide a way to enforce these interactions and ensure rules are not broken. These are essential to the development of a data sharing mechanism in order to automate the process of enabling data sharing and data access control, and ensure no third party interferes with the behaviour. These tools execute exactly as coded, without any possibility of downtime, fraud, censorship and third party interference. As long as the owner does not loose or leak the credentials, it makes it so only the owner has control over the access and sharing of the data and the smart contracts enforce the rules dictated by the owner. [AEVL16] Additionally, peers execute the same code, and there must be consensus over the results. So, the outcome is not dependent on a central authority, but on decentralized network of peers.

Private blockchains facilitate this capability, being able to be deployed anywhere. Hyperledger Fabric capabilities provides identity over anonymity to ensure every participant is known and registered in the network. Additionally, sharing of data only comes from registered, authorized users. [Hyp18, XSS+17]

### 2.6.1 Data Access Permissions

Permissions is a mechanism that keeps control over the access permissions of every user the owner of data has given, essentially being able to identify who of those users are and keeping a clear record who previously had access to what and who currently has permissions on what, for how long and what permissions. Using blockchain properties means the owner has increased control over these aspects and can have complete control of who is he sharing with. The most important part is providing a clear record of every access given to any data, by any user. Providing a clear tamper proof record of the interaction on data.

### 2.6.2 Data Access Control

Data access control ensures data can only be accessed by authorized users in which the owner of data has given access to. Blockchain properties ensure access control firstly to only users registered in the network, and then to users with access without the need of a third party to determine if a user has access to any data or not, therefore it does not need the request the storage provider if a user has given access or not, the information is already stored in the blockchain at some point in time. Additionally, it also differentiates if a user that has permissions from a user that owns the data. This is done by attaching the identity of the user to the data.

Regarding current access control permissions and previously granted access control, the blockchain keeps track of every interaction made and every user that had and has access to any piece of data. This system does not rely on the storage provider to keep control of the permissions or access by external users, but uses the smart contracts to enforce this [XSA+17, AEVL16, XSS+17]. In theory and practical terms, a user only requires to setup a share agreement between him and another party, and the smart contracts will handle all the business logic, from allowing access to the data to removing privileges solely based on the blockchain. This does not mean a cloud storage provider will not contribute with additional measures.

Smart contracts need also to grant access to data or not, based on the current state of user permissions stored in the blockchain. Since the smart contract acts without interference, and the blockchain is tamper proof, it can easily detect if an access request is valid or invalid.

### 2.6.3 Revocation of Privileges

Revocation of privileges is the action of stopping the access to any data by any user, either manually or by automatically due to conditions. For example data violations: Blockchain enables this procedure without the

assistance of storage providers, with the aid of smart contracts and the data stored/added, and shifting the decision process to the blockchain network and stopping access and logging it.

The owner is in complete control of external users of who access the data, as such he can execute smart contract functionalities to manually revoke privileges, or alternatively state in the creation of a share agreement the time limit of the access. Revocation also happens immediately at moment of requesting if a user with permissions does not have the correct permissions, **i.e:** a user with read access attempts a write access. This action should automatically revokes the data access for possible violation. These unstoppable contracts written in code, will automatically determine when a user loses their access to what data and stop any more access, unless provided by the owner.

### 2.6.4 Ledger

Blockchain, or Ledger, has stated in Section 2.1, is a tamper proof system of records, therefore it provides great opportunities for the owner to keep control on data access and users with permissions. Being able on any point in time to determine who, when, what was modified or accessed without a single point of failure. [AEVL16, XSS+17]

## 2.7 Data Storage Approaches

There are multiple proposed approaches in order to achieve the stated goals, each with their own challenges and differences. From fully decentralized to hybrid approaches but the most important point is that all of these maintain the blockchain as its most crucial and core component. Essentially the blockchain provides a record of information of files and share access.

Another approach is to use the blockchain blocks as data storage. By the properties of the blockchain this may be a viable approach for data storage. Alternatively, if blocks are not used, the purpose of the blockchain would be to locate the file, present the users their files and shared files and help retrieve the original file from its corresponding repository (either centralized or distributed) control data sharing and data access control while ensuring ownership. Smart contracts would be used, as the only way, to allow users to interact with the blockchain in other to perform various operations, like register a file in the blockchain, create share agreement, among others. The following data storage approaches are possible:

- **Block Data Storage**. With this approach, the file data is to reside in the blocks of the blockchain. This approach makes data share the properties of the blockchain, which is immutability, tamper proof, ownership where each peer would contain a replica of the data. While this may seem a viable approach has many complications regarding the consistency, security of the blockchain as well as confidentiality and privacy of the data. [LSZ15, SZ15]

- **Distributed System**. With this approach, a file is to be fragmented into smaller equal encrypted pieces and distributed to the network to be kept in other peer's physical machines, according to rules stated in the smart contract. The blockchain would then handle all the relevant information in order to log, locate, decrypt and rebuild the original file, while still providing complete ownership of the file. [sto]

- **Hybrid System**. This approach would require a centralized approach, meaning a central authority where files would be kept in a central repository while using the blockchain, as the decentralized component of the network. for support in terms of ownership and other relevant information to log, retrieve and decrypt user's files when requested. Files would be shared according to rules stated on smart contracts. This

approach would mean have a centralized component, the repository, and a decentralized component, the Blockchain.

## 2.8 Related Work

The blockchain concept led to the rise of blockchain based services. Cloud backend services are obviously among them. Storj and Sia are two of these services. Any of the two presented solutions imply costs lower than more known and traditional methods like Google Drive, Dropbox, Amazon S3, for example. [sia, sto] Although they use a relatively new innovative and disruptive technology, using blockchains provides a far more affordable decentralized solution to cloud [WLB14]. Additionally healthcare based blockchains tend to provide a data sharing mechanism aswell.

### Storj

Storj is an open source but not entirely fully decentralized backend cloud storage service implemented on top of the Ethereum Blockchain that allows its participants to rent unused hard drive using the public blockchain as a ledger to keep track of shared files. It does require trust on third parties (named "*bridges*") to connect users to storage participants in order to pay for storage. Each block in the blockchain contains hash functions, keys, file locations, etc…. Shared files are fragmented and spread across the decentralized network and encrypted, guaranteeing the owner, he is the only one who has access to the complete file, it uses its own digital currency to reward the participants for keeping the network up and running, although allows payments to be done in other currency besides the network [sto].

But consequently it has some issues. It is not fully decentralized meaning, it has single point of failure. Making it vulnerable to Distributed Denial of Service due to the "bridges", incapacitating the entire network.

### Sia

Sia is an open source project, providing a fully decentralized, with no single point of failure cloud backend service where participants of the network can rent his unused hard drive space and host files. *"Instead of renting storage from a centralized provider, peers on Sia rent storage from each other. A blockchain, similar to Bitcoin, is used for this purpose"* [VC14]. It works by creating a contract between the storage provider and client's data, and periodically submit proof, verifiable through the Blockchain, of their continued storage, until the contract expires, while redundantly storing data across multiple hosts in order to achieve high availability. The platform uses its own digital currency in order to reward the participants of the network [sia].

But like Storj, it has some issues. In order to help the network it requires synchronizing with the entire blockchain which can take a considerable amount of time, due to the size of the blockchain. It also have some scalability issues in handling large volumes of data due to limitations of the blockchain. In addition, payments are only done in the network currency Siacoin. That could be viewed as a downside.

### Healthcare and Blockchain Data Sharing

As referenced before multiple approaches to data sharing on blockchain have been made. All of these approaches are healthcare related, since data privacy and confidentiality in these areas is very importance. [XSA+17, AEVL16, XSS+17] Some of the works are

- MedRec is a decentralized application running on top of Ethereum, using blockchain, it permits to manage authentication, confidentiality, accountability and data sharing. While also allowing to integrate with the existing local storage for easier adaptability [AEVL16].
- BBDS, Blockchain Based Data Sharing, is similar to MedRec but relies on a private and permissioned blockchain which guarantees only invited and accepted users can interact with the medical records, heavily going by Proof of Verification. This premises allows users to be known and all the interaction to be logged in the blockchain [XSS+17].
- MedShare is similar to the previous approach, This approach aims at trust-less medical data sharing among cloud storage providers. The system suggests a permissioned and private blockchain. *The design employs smart contracts and an access control mechanism to effectively track the behavior of the data and revoke access to offending entities on detection of violation of permissions on data* [XSA+17].

# 3

# Blockchain

*In this Chapter, Blockchain is the centered topic, in which a clear description of this disruptive is provided. Besides an overall description, this Chapter also describes various concepts that define blockchain in a more detailed level. Blocks, transactions, blockchain networks, consensus protocols and forking are approached concepts. Additionally some uses cases of Blockchain in various areas, like Cloud Service, Health and Media among others.*

## 3.1 What is Blockchain?

Blockchain is a Distributed Ledger Technology . Considered a distributed database working as a ledger, in other words a distributed ledger, it provides the ability to facilitate the recording of transactions and tracking of assets. Blockchain is a continuously growing chain of digital blocks, where each block references the previous block connecting the entire chain, tracing all the way back to the Genesis Block[1]. Each block is a set of records allowing permanent and verifiable records of transactions in a secure manner. Each Block, if valid, is then added to the blockchain. So each block contains the most recent occurrences in the network up to that point.

[1] The first block of a Blockchain

The blockchain is a disruptive technology since it revolutionizes the perception of data, permitting for a network to have records of all participants in a single ledger shared by all. Peer to peer, thus decentralized, transactions that take place and ownership are proven via cryptography instead of trust on a central authority. Transactions are permanent and irreversible (at least not by conventional terms) which means the blockchain is cryptographically secured, it provides a single record of truth which contents are replicated by peers in the network and a distributed consensus ensures that all parties follow an immutable agreement and share the same information and information is added via consensus.

The objective of the first ever implemented blockchain (Bitcoin) was to solve the problem associated with digital currency which is double spending. This notion is using the same currency more than once, which should never be allowed. Because digital information is easily replicated, so creating digital currency posed the same challenge. Blockchain solved this problem.

This disruptive technology, having many applications, sought out to change finance is now being referenced by some as the next big innovation since the Internet.

Quoting Ethereum Co-Founder Vitalik Buterin, *"whereas most technologies tend to automate workers on the periphery doing menial work, blockchains automate away the centre. Instead of putting the taxi driver out of a job, blockchain puts Uber out of a job and lets the taxi drivers work with the consumer directly."*

## 3.2 Bitcoin

Bitcoin, is the first blockchain to be created, posing a massive impact and described in just 9 pages. The following text, is the abstract of the Bitcoin whitepaper[2].

*" A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network. The network timestamps transactions by hashing them into an ongoing chain of hash-based proof-of-work, forming a record that cannot be changed without redoing the proof-of-work. The longest chain not only serves as proof of the sequence of events witnessed, but proof that it came from the largest pool of CPU power. As long as a majority of CPU power is controlled by peers that are not cooperating to attack the network, they'll generate the longest chain and outpace attackers. The network itself requires minimal structure. Messages are broadcast on a best effort basis, and peers can leave and rejoin the network at will, accepting the longest proof-of-work chain as proof of what happened while they were gone."* [Nak]

The Bitcoin whitepaper of just nine pages described how it all worked and and introducing the notion of digital currency in the possible manner. In short, Bitcoin is, a peer to peer version of electronic cash that allow payments to be sent directly from one party to another without going through a financial institution first. [Nak]

Digital currencies all suffer from at least one common issue which is the double spending problem, the concept of using the same currency more than once. Since digital information is easily duplicated, a digital currency presents the same issues, for this reason the blockchain was conceptualized and implemented by *Satoshi Nakamoto*[3], a name used by an unknown entity.

The blockchain as described in the Bitcoin Whitepaper is an immutable ledger to record transactions using digital signatures. This makes it possible to create a decentralized payment as the blockchain is a *"chain of digital signatures that defines an electronic coin"* [Nak], allowing *"online payments to be sent directly from one party to another without going through a financial institution"* [Nak]. This method allows every transaction to be

[2] A less technical and less extensive article

[3] Author of Bitcoin Whitepaper and Creator of the first Blockchain

logged in the blockchain in a form of block and its contents agreed upon by a consensus algorithm called Proof of Work to make extremely difficult for the blocks to be tampered with but easily inspected, ensuring it's stability and security. In order to confirm that a transaction did exist and to prove that users did not attempt malicious actions regarding transactions, all transactions are publicly available, to remove the need for a central authority to check every single transaction. This means every user can check the legitimacy of every transaction from the Genesis Block.

Bitcoin was the basis for many other blockchain implementations. Its source code is publicly available[4], and multiple projects used this as inspiration to build their own projects.

## 3.3 Transactions

A transaction represents an exchange of ownership of value from one user to another or any information that is adds content to the blockchain cryptographically signed by the submitter. In other words, transactions are operations that will perform write operation on the blockchain, increasing the size of the blockchain. These transactions are then broadcasted to the network and picked up by peers. When they reach a certain number of transactions, or after enough time, they are bundled into a block.

Naturally, depending on the type of blockchain (see Section 3.5 for details) transactions are usually not encrypted. In public networks, other peers maintaining the network as well any other user are aware of every transaction and can easily inspect them. This method is to enforce security and substitute trust over cryptographic proof, because a peer of the network wouldn't be able to tell malicious transactions from valid ones, it may be added that transactions are visible to outside users[5].

In private networks or any network that supports private transactions or impose restrictions on read operations, users may not be able to inspect them, besides authorized users.

In Bitcoin, transactions are between two parties, each one are represented by an address, which is an unique alphanumeric sequence which is the hash of the public key. These addresses are public, enabling the transfer of value. Although in other networks, transactions do not need to represent an exchange of ownership of value between two parties, but just logging of information without the need of the second party.

Transactions are the essence of the blockchain as they provide the means to keep record of users activity in the network by registering it in a permanent way. Depending on the network type of the network transactions may or not be visible to all other users. In private chains this is the contrary, since the peers are considered trusted, transactions are not visible outside the network but only within it, but may be not visible to certain users of the networks. (additional rules)

When a transaction is formulated, it is broadcasted to the network for peers to pick it up in order to generate a **block**, transactions are placed in a queue waiting their turn to be picked up. Although transactions will eventually be bundled with other transactions into a block there is no exact guarantee when that will occur.

In public blockchains, monetary incentive solves this. Transactions in order to ensure they are picked up and placed into a block may require a fee. The amount of fee sent depends on who sends the transaction, but, in a general, the higher the fee the higher the chances it will get picked up as soon as possible. In private blockchains, or blockchains that do not necessarily resolve around monetary values to "get things done", it may just be by arrival time or urgency because incentives are not the focus.

Transactions by definition in the blockchain are irreversible and final, once submitted and if considered valid

[4] `https://github.com/bitcoin/bitcoin`

[5] `https://www.blockchain.com/pt/explorer` can for example show the history of all transactions

they will no longer be changed, but will only be considered irreversible when an agreed amount of confirmations[6] is reached, this makes it difficult for any attacker to perform malicious actions that could reverse the transactions (double spend).

In order to submit a transaction the user or peer may be required to synchronize his blockchain with the network before sending but will depend on the client software.

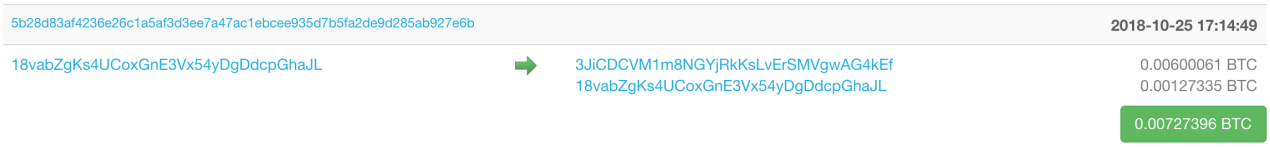



Figure 3.1: Bitcoin Transaction.

Figure 3.1[7] explores a transaction between two parties, one address to another and the amount of Bitcoin transferred from one address to the other. This is a case of a very simple transaction with one input and two outputs. By inspecting Figure 3.1, one of the output address is the same as the input, this is the ”change”, because the value of input was greater than the output value, so some of the currency must be refunded.

## 3.4 Block

Blocks by definition are batches of transactions that occurred at some point in time specific to a blockchain network. The term blockchain comes from how blocks are generated. Each block contains the hash of the previous block which acts as a secure link, allowing every block to be traced back to the Genesis block. Each new block added to the chain represents the most recent transactions that occurred in the network.

The system of connecting blocks with the hash of the previous block along with the consensus protocol make the blockchain very resilient and tamper proof because one change to an old block's contents imply a different hash which causes the next block to be incompatible and so forth, breaking the link and making the whole chain incompatible. This makes the detection of an malicious action easily detectable.

All blocks added to the chain are considered valid by rules of consensus representing all valid and irreversible transactions, when enough confirmations are achieved. The blockchain is replicated across participating peers, where each peer contains a full copy of the blockchain considered valid by the consensus protocol.

In the case the peer wants to participate in the creation of blocks, the peer must have his blockchain synchronized at all times, which implies downloading the full blockchain to the peer's machine so this matches the blockchain that is considered valid by the consensus protocol.

How new blocks are generated and accepted to the blockchain depends entirely on the consensus protocol associated with that network. (see Section 3.8 for details) Additionally it also depends on the implementation of the blockchain, the contents of a block can also differ, in the following sections describes three different blocks of three different blockchains.

[6] Number of blocks that follow the one the transaction resides in

[7] `https://www.blockchain.com/en/btc/tx/5b28d83af4236e26c1a5af3d3ee7a47ac1ebcee935d7b5fa2de9d285ab927e6b`

### 3.4.1 Bitcoin, Ethereum and Hyperledger Fabric Blocks

As previously mentioned, the contents of a block depends on the implementation of the blockchain. In order to exemplify the difference, two different blockchain's blocks are presented: Bitcoin and Hyperledger Fabric.

#### Bitcoin block

The Bitcoin block is the first concept of blocks in blockchains. The structure is relatively simple yet complex, the block contains information regarding number of transactions, size of block but most importantly is the block header. The block header contains the hash of the previous block, a Merkle Tree of transactions at the leaves [Nak], a timestamp and a nonce[8]. Figure 3.2 represents a block header in the network.

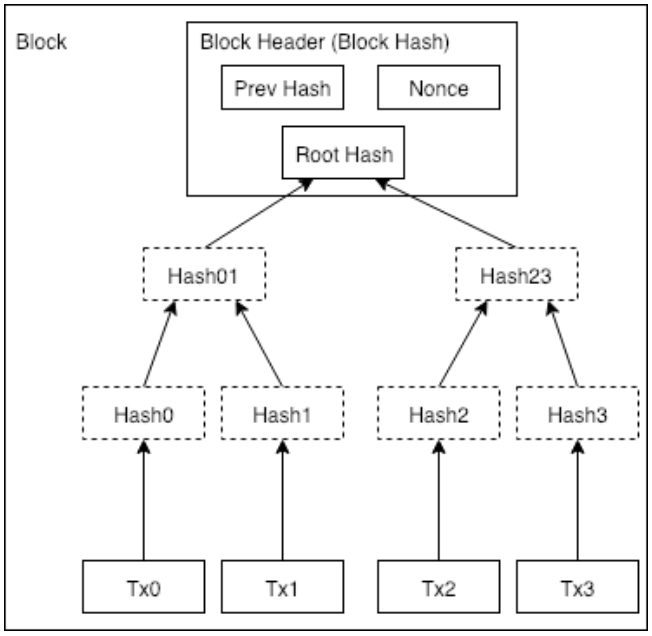


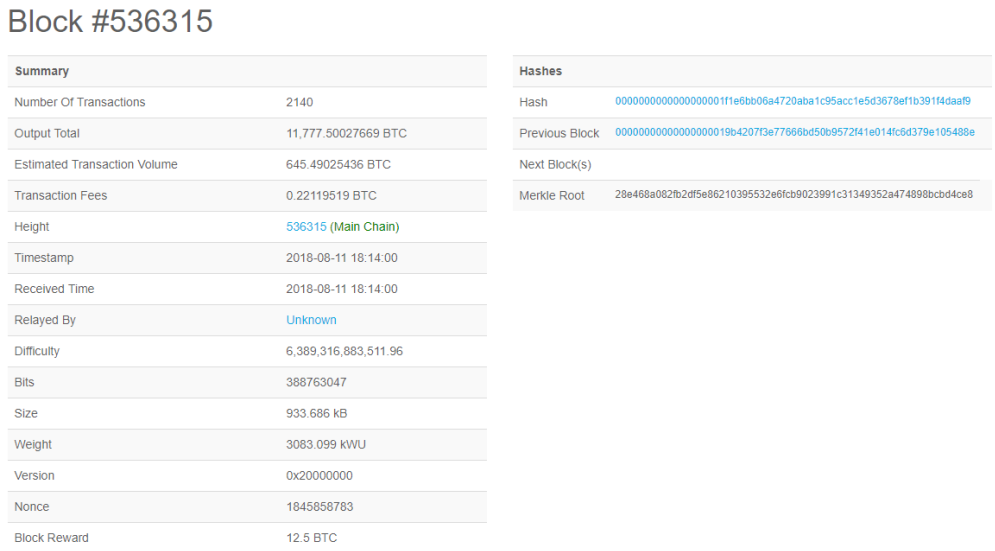

Figure 3.2: Bitcoin Block Header Structure [Nak]

Additionally, *blockchain explorers* are a term for platforms that scans the blockchain and provides a clean interface for users to inspect the contents more easily. Figure 3.3 contains additional block information and statistics extracted from a block at some point in time[9]

The important factors to note on the block from Figure 3.3[10] are the hash, the difficulty, the nonce, the version, the previous block hash an timestamp and the merklee tree root.

- **Hash** represents the cryptographic proof that the block has been put some effort into building it. And the **Previous Hash** representating the connection to a previous block.
- **Difficulty** represents the number of tries required in order to generate a nonce that will output that **Hash**.
- **Nonce**, a random number, that enabled the block to achieve the target **Hash** structure.

[8] A random number that will give a block the target hash, participating peers will try to find this value via "mining"
[9] `https://www.blockchain.com/btc/block-height/536315`
[10] https://www.blockchain.com/btc/block-height/536315



Block #536315

| Summary | |
|---|---|
| Number Of Transactions | 2140 |
| Output Total | 11,777.50027669 BTC |
| Estimated Transaction Volume | 645.49025436 BTC |
| Transaction Fees | 0.22119519 BTC |
| Height | 536315 (Main Chain) |
| Timestamp | 2018-08-11 18:14:00 |
| Received Time | 2018-08-11 18:14:00 |
| Relayed By | Unknown |
| Difficulty | 6,389,316,883,511.96 |
| Bits | 388763047 |
| Size | 933.686 kB |
| Weight | 3083.099 kWU |
| Version | 0x20000000 |
| Nonce | 1845858783 |
| Block Reward | 12.5 BTC |

| Hashes | |
|---|---|
| Hash | 00000000000000000001f1e6bb06a4720aba1c95acc1e5d3678ef1b391f4daaf9 |
| Previous Block | 00000000000000000019b4207f3e77666bd50b9572f41e014fc6d379e105488e |
| Next Block(s) | |
| Merkle Root | 28e468a082fb2df5e86210395532e6fcb9023991c31349352a474898bcbd4ce8 |

Figure 3.3: Bitcoin Block Statistics and Information.

- **Version** represents the software version of the block.
- **Merkle Tree Root**, the root hash of tree containing the transactions at the leaves.
- **Timestamp**, represents the moment of creation. Representing the most recent occurrences in the network.

#### Hyperledger Fabric block

Blocks in Hyperledger Fabric are discussed in detail in Chapter 5. Nonetheless Figure 3.4 presents the structure of the block in version 1.0 of Hyperledger Fabric. Shows an almost completely different structure from the Bitcoin block. A notable component is the Endorser signature, that will be in detail in Chapter 5.

## 3.5 Blockchain Network Types

Blockchain networks come in two types, public and private, each have their differences, functionalities and reason to exist and as the names suggest, a public blockchain is meant to be accessible by anyone, while a private blockchain is meant to be only accessible depending on certain conditions, invitation for example.

### 3.5.1 Public Blockchain

Public blockchains, are mostly open source and permissionless[11], meant to be accessible by anyone with no restrictions being applied, which means any user can freely join the network, interact with it and contribute, usually by ”discovering” new blocks to be added to the blockchain. In other words joining a public blockchain is where any user perform read and write operations, if valid by the network rules of consensus. By definition a public blockchain means transactions are publicly available meaning every transaction is visible. This method

[11] Anyone can read from the blockchain anyone can make changes and contribute as a peer as long as they follow the rules

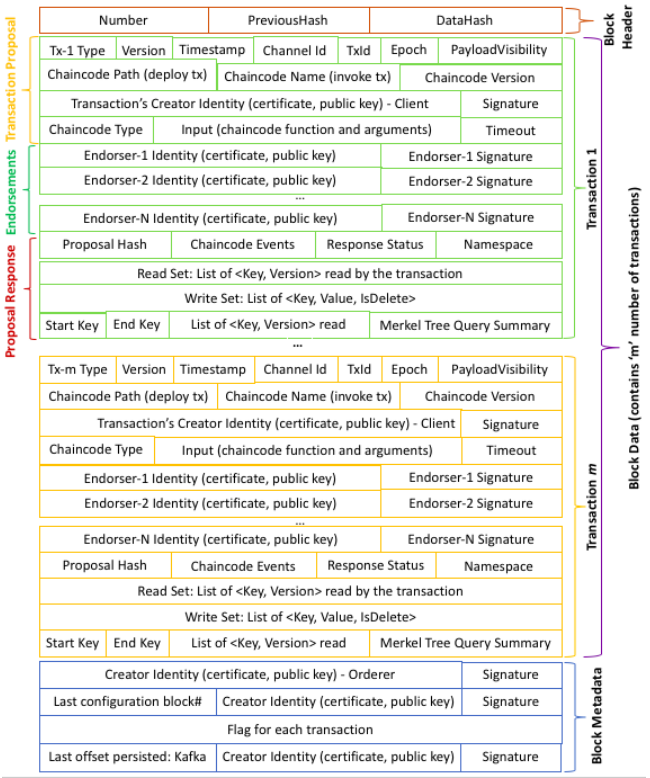



Figure 3.4: Hyperledger Fabric Blockchain Block Structure

increases the overall security and integrity of the blockchain as every user is aware of all transactions and anyone can confirm the existence of every transaction that as ever occurred since start of the network (genesis block) which is particularly useful for the detection and prevention of double spending. [Nak] Since anyone can join a public network it means trust cannot be considered as a reliable factor for security, stability and growth of the network. Instead, they rely on publicly presenting transactions and cryptographic proof to ensure said factors.

Bitcoin is the first blockchain and public blockchain described in Section 3.2, as it is the foundation of blockchain development.

Ethereum is another widely extremely popular public blockchain as it introduced a blockchain for the development of smart contracts, along with a custom programming language, permits its developers to use the properties of the blockchain not only limited to finance, although it also permits the creation of cryptocurrency using the Ethereum Blockchain as the eco-system, these are called Tokens instead of Cryptocurrency. With the objective of becoming a global computer, it permits the creation of Decentralized Applications. Similar to normal applications but these use the blockchain and smart contracts as part of their backend.

There are many and open source blockchain public networks and more on the rise. Bitcoin and Ethereum are two of the most well known and largest blockchain networks.

### 3.5.2 Private Blockchain

Private blockchains as the name suggests are private networks, and can be open source but permissioned[12]. These networks are usually associated with a level of trust among its peers. In order to become a peer certain conditions beforehand may apply, such as invitation only, and to when be a participant of the network there can be restrictions on available operations, such as restricting certain users on read only. And tend to go by identity instead of anonymity or pseudo anonymity.

These networks, may be more complex in terms of peers, while in public blockchains there is usually one type of peer (those who create blocks), in private blockchains there may be more than one type of peer. For example, in Hyperledger Fabric, a private blockchain implementation, there are two important peers, an Endorser who endorses a transaction, and the Orderer who collects endorsed transactions and creates blocks.

Participants of private networks are usually restricted, while in public networks a participant can read/write from the blockchain, maintain the blockchain , and many other operations, the private networks can restrict their users on read only, write only, only a selective peers to maintain the blockchain, among other restrictions. Although there are restrictions they allow for increased privacy and confidentiality, such as private transactions taking place which are only visible within the network, and even within the network they may only be visible to set of users or peers or both. This overall greatly increases its applicability.

These networks are usually associated with closed situations like corporations under an umbrella, various departments in a organization, financial institutions who want the benefits from the blockchain but want their transactions not to be visible to the world, among others.

These networks rely on trusted peers, to an extent, because it implies a much smaller network when compared to public networks like Bitcoin. In the work described in this dissertation Hyperledger Fabric is the chosen blockchain framework.

## 3.6 Cryptocurrency

Cryptocurrency is a digital asset that is used as an alternative method of exchange of value that is purely digital. By definition cryptocurrencies can either be centralized or decentralized.

Centralized currencies are no different from paper currency, it requires the need of a central authority to control the flow of currency, and validate transactions to ensure no double spending. Decentralized currencies, however, are a different type where there is no central authority to dictate the flow or the future of the currency, an example of this is Bitcoin, the first decentralized cryptocurrency controlled by the community as there is no central entity or government regulation applied to it.

What defines a cryptocurrency is ”as a chain of digital signatures” [Nak], where each transaction is a change of ownership of value from A to B.

Cryptocurrency wallets are also standard as they provide an interface to interact with the blockchain. These wallets maintain the private and public keys of the users, only to be accessed by the owner.

Cryptocurrency is not limited to public networks. Any network can implement a form of cryptocurrency as it is just needed to write the business logic for it.

**The most known cryptocurrencies:** Bitcoin (**BTC**), Ethereum (**ETH**), Litecoin (**LTC**), Ripple ( **XRP**).

[12] Restrictions on who can contribute as a peer and can join

### 3.6.1 Double Spending Problem

The most important issue when facing a digital currency is double spending. The concept of using the same currency more than once. Digital assets are easily replicated. This leads to a major concern when referring to digital currency as the idea of an user duplicating his digital currency and sending it to another party poses a major flaw in the digital currency scheme, leading to lack of trust and other consequences. Blockchain was the first successful solution to solve this in a decentralized way.

### 3.6.2 Blockchain and Cryptocurrency

Blockchain is not exclusive for cryptocurrency. Not all blockchains are cryptocurrency focused, an example is Hyperledger Fabric, a private blockchain implementation, which will be described in Chapter 5. The blockchain merely provides a strong, valid and proved solution for digital currencies to shine, but essentially blockchain is an immutable ledger allowing the recording of information to be done in a permanent way. The cryptocurrency also works, as a method of incentive to peers to keep stability and security in the network in order to keep the network healthy by contributing to it by being a peer. But if a blockchain supports the build of complex smart contracts, essentially any network, even not cryptocurrency focused, can have cryptocurrency as it just business logic that can be programmed into a smart contract.

## 3.7 Wallets

Wallets are a sort of programs or part of it, which are a enable store and secure the credentials of a user. This is used to execute transaction proposals and possibly other features. Which by definition enables the interaction with the blockchain network, taking advantage over its features. If a network supports cryptocurrency then it provides a way to keep check his balance on the stake of the blockchain network, transactions proposals (request to send value to another address) and possibly other features.

Depending on the type and software of the wallet and the network type, it may be required that the wallet is synchronized with the corresponding blockchain in order to interact with the blockchain network. This means having a copy of the blockchain in the physical machine of the user. In private or permissioned networks, this may not be the case, and instead these communicate with peers.

When related to cryptocurrency, wallets do not store the currency, as that goes against the properties of the blockchain. Instead, wallets scans the full blockchain or a world state of it, hence the possibility of having the full copy of the blockchain, and determines what is the account balance of the user.

There are multiple types of wallets, not limited to a program running on a laptop for example, each with their advantages and disadvantages and some rather controversial due to its nature.

The different types of wallets are the following:

- Offline Wallet or Desktop Wallet. Also know as the desktop wallet, it provides an interface for the blockchain or blockchains (if the wallet supports multiple networks) and the keys are maintained in the physical machine of the user. It provides security and the owner controls the keys (maintains ownership). This is the most common type of wallet. The downsides are obvious, a physical machine is always vulnerable to attacks so the user must be careful and must protect it. The user can forget or lose the keys, hence will loose the cryptocurrency, but these wallets tend to have some safety mechanism, such as backup, so the user wont easily lose them.

- Paper Wallet. Cheap alternative to the hardware wallet, emits a readable code and is used to recover the private and public keys of the user, which can confirm the entity of the user hence access his stake in the network. The paper wallet cannot be damaged and must be kept safely. If the code in the paper is unreadable then the keys are unable to be recovered. Usually the codes are represented in QR code form.
- Hardware Wallet. The most secure type (but not free) type of wallet is the hardware wallet. Requires the use of an actual physical device to allow interaction with the blockchain such as perform transactions, securing and recovering private/public keys, accessing the wallet, among other features. A widely used hardware wallet is the Nano Ledger S, as seen in Figure 3.5, which resembles a USB flash drive, it supports numerous cryptocurrencies, and even if this hardware wallet is lost or damaged, the user can still recover the keys, by a safety mechanism on setup of the Nano Ledger S.
- Online Wallet. The Online wallet is rather controversial, this wallet is hosted online, which means the host controls the keys and not the actual user, this means the user is not in complete control of over his identity on the network which goes rather off track of what a blockchain truly his. Nonetheless, these hosts are usually mostly safe and secure and abide by the law of the country the host resides in. Personal information like passport or national id card or driver's license will most likely need to be submitted in order to use the services of said host. Online wallets are very common when associated with cryptocurrency exchanges. Much like a stock exchange but cryptocurrency exclusive. (hence the online wallet)


Figure 3.5: Ledger Nano S

## 3.8 Consensus Protocols

Consensus protocols maintain the stability and consistency of the blockchain and ensures consensus over the contents of the blockchain. Consensus determines how new blocks added to the blockchain and which blockchain does every peer in the network consider to be the valid blockchain. Essentially every decision that affects the blockchain must be determined by consensus. Additionally any change to the consensus protocol if accepted by consensus will result in a soft or hard fork, as explained in Section 3.9. Blockchain networks must implement a consensus protocol and the ones described in the following sections are the most popular ones. Proof of Work the most popular and well known and the one used in Bitcoin; the emerging and viable alternative to Proof of Work being Proof of Stake and Delegated Proof of Stake; and last the Hyperledger Fabric choices of consensus.

### Block Confirmations on the Blockchain

In order to increase security in a blockchain network, a transaction is only considered irreversible and final if the block where the transaction resides is followed by a consensus defined amount of blocks first. This is to prevent, by making it difficult for malicious users to attempt an attack on the state of blockchain network. Notably the double spend attack or majority attack. (see Section 3.8.1 for more details)

### 3.8.1 Proof of Work

Proof of Work, is the most known consensus protocol related to blockchain and also the one Bitcoin relies on, but it did not originate in Bitcoin White Paper. This system is based on an existing one called *hashcash*, *was originally proposed as a mechanism to throttle systematic abuse of un-metered internet resources such as email, and anonymous remailers*. [B+02]

In the context of blockchain, when using POW, the generation of new blocks a exponentially difficult task but permits users and peers to easily verify its legitimacy. In a short definition, proof of work is a protocol that requires peers to solve complex mathematical puzzles in order to generate valid blocks. This protocol is associated with the common used word in the cryptocurrency world:"miner".

#### How it works

Miners are peers of the network whose task is generating new valid blocks with valid transactions, achieved by solving complex mathematical puzzles, which produces a hash that represents that block and meets the requirements agreed upon by the network, and then rewarded by their efforts only if their block meets the requirements of the protocol and is accepted to the active blockchain. Proof of work is nothing but a piece of data, a hash, that must be difficult to generate, difficult to replicate but legitimacy of that block must be easily verified.

This protocol requires a exponentially growing computational power that grows based on various factors like number of generated blocks, how difficult it was to generate a block and the computational power used to generated a new block. In technical terms each generated block proof of work (the resulting hash) must be lower than a certain difficulty target, and the hash must have a specific structure. In Bitcoin must have an X amount of 0 bits at the start of the hash. This becomes increasingly more difficult as the blockchain grows in size which leads to a very low probability of success for each new block leading to a ludicrous amount of trial and error. The difficulty to generate a new block is also adjusted every certain amount of generated valid and accepted blocks so the rate of block generation remains constant. Each new block contains the hash of the previous block providing a way to link the genesis block to the most current valid block and every block in between.

In technical terms, miners collect transactions and additional data, in Bitcoin is timestamp, Merkle Tree, or any and the previous block, and this is used to formulate the structure of a block, after that is done the "mining" begins, miners need to find the hash that has that specific structure. The way this is done is by means of a nonce as presented in Figure 3.3 and Figure 3.2. Miners need to find the nonce which is a random number that will generate a hash with that structure, this is solely based on trial and error so there is no real time limit when a miner will find that nonce, mainly luck so there is no other alternative then pure brute force. Modern computers have no trouble cycling through large numbers especially if its an interval of Long or Integer size. This process is so complex because it is not guaranteed that cycling through all possible values of a nonce will find the correct hash, so a change in the data that is calculated along with a nonce must be cycled as well, notably the transactions and its order in the Merkle Tree and timestamp, the previous block hash remains the same.

This protocol makes the blockchain very resilient and very difficult to be tampered as a block gets older and older. This is because in order to modify the contents of a block in the blockchain will result in a totally different hash of that block. This makes it incompatible with the hash of the previous block stored in the block that follows it and this incompatibility will propagate all the way to the most current block therefore requiring a massive amount of computational power as it needs to re-generate all blocks that follow the modified old block and all of the work that generated them. In more technical terms, modifying an old block is not possible, instead

a modified block will generate a fork. See section 3.9 for more details.

In order to incentive miners to solve these complex mathematical puzzles, each time a block is generated the miner creates a transaction from no one to himself, named **coinbase**, with an agreed amount of cryptocurrency. If the block is accepted by the network the "miner" creates new cryptocurrency rewarded to himself as well along with the fees associated with the transactions. So miners compete for the acceptance of blocks in order to get rewards from it.

Fees in Proof of Work are must as users who wish to use the blockchain via transactions will want their transaction to be processed as fast as possible so the users include a fee (payed to the miner) to try and include their transaction in the next accepted valid block.

### The Longest Chain Rule

The longest chain rule is the most important rule of Proof of Work, this implies the network to agree upon. Valid blocks must reach other peers in order to have their blockchain synchronized with all other peers, but this means blocks take some time to reach other peers which naturally can lead to conflicts because blocks can be "discovered" concurrently. These conflicts are related to multiple valid blocks being broadcasted to the network, roughly at the same time and depending on number of factors like geographical location of the peer will influence which block the peers get first so this leads to the issue of what the valid block that a peer must accept in their copy and will eventually lead to **competing chains**. In networks that use Proof of Work consensus protocol, they follow the **longest chain rule**. This states that a peer must accept the first block he receives and must start building on top of that block. If later that same peer receives a block that corresponds to a different chain which is obviously longer chain than his, then he should discard the work he has done on his previous chain and accept that new longer chain as the valid one, since these peers will not have the complete information they must reach to other peers to complete it. Figure 3.6 presents a visual representation of the **longest chain rule**.


Figure 3.6: Longest Chain Rule [Lew15]

Figure 3.6, by Antony Lewis *A Gentle Introduction to Blockchain Technology* [Lew15], presents a case of forking, where the blockchain is forked and peers mine on different sides of the blockchain, block 81a, block81b and block 81c. When block 82b is found, peers working on block 81a and 81c, discard the previous work and immediately start working on block 83b.

**Peers discard the previous chain they were working on, whats happens to the transactions?** Every time transactions are broadcasted to the network they are placed in queue, when peers discover they are working on the wrong chain, all the transactions of the now inactive chain are placed back in a queue, peers ask for the information of the missing block(s), assuming they didn't received the block from the other chain and kept a copy, which may be possible. This most likely contain the same transactions of the block in the inactive chain, so peers who receive the information start comparing with the queue with the active chain. Since those blocks were not confirmed, missing transactions will eventually get fulfilled.

Additionally the longest chain is considered the **active chain**, which in Proof of Work it refers to the chain that has the most of work in, and by definition, is the chain where consensus exists, as in, equal to all other peers and all peers state this chain as the valid one and the one to follow. But there are multiple inactive chains, which reflect the case of the conflict previously described, peers may keep the old block but must rely on the active chain if peers want their contribution to take place.

Nothing stops a peer from continuing to build on that older chain, but this action has to be taken into consideration that no consensus will happen because that chain is considered inactive so no rewards unless it results in a 51% Attack.

### 51% Attack, Majority Attack or Double Spend Attack

This vulnerability deserves an explanation by itself as it is a major issue in a blockchain that uses Proof of Work. It refers to an entity controlling more than 51% of the network total computational power, leading to effectively double spend which is the concept of using the same currency more than once, which results in reverting previously confirmed and valid transactions. However it is not possible to change the rewards of generating blocks or revert users transactions (as it requires access to private keys) **unless** they were in the block(s) that were affected by the double spend. The malicious user however can approve transactions which are malicious since the user as more than 51% of the computational power those transactions can be considered valid.

**How it works:** If the entity holds more than 51% of the computational power of the network, he has a much higher chance of generating blocks faster than the rest of the network so the malicious user can submit a transaction, start a competing chain (without broadcasting it) and start working on the chain and after the transaction is confirmed the malicious user can broadcast the chain made if the chain is larger than the current active chain. This in the proof of work algorithm states that the new chain has more work in and now is considered the active chain, effectively conducting a double spend because the transaction in the previous longer chain is now reverted.

This vulnerability has a flaw from the start, if an user/group manages to get 51% of the total computational power and attempts a double spend attack, the network is then compromised which may lead to users leaving the network causing lack of credibility to the blockchain network. In terms of value, the digital currency value would drop, losing the investment.

In short 51% attack can do the following:

- Reverse transactions affected by double spend and double spend transactions previously seen in the blockchain.
- Prevent transactions from being confirmed.
- Prevent miners from mining valid blocks.

But in contrast, the malicious user cannot:

- Reverse other people's transactions without their cooperation. (unless it was affected by the double spend)
- Prevent transactions from being sent at all.
- Change the reward of generating blocks
- Generate cryptocurrency "magically".

In order to increase security on this type of attack, the number of confirmations makes the attack exponentially more difficult because in order to effectively double spend, more blocks would have to be generated. But in contrast the more confirmations the longer it takes to confirm a transaction.

In Bitcoin or Ethereum, this type of attack has never been made (with malicious intention not by means testing to find a bug on the live network with consensus of the network) or at least successfully.

### Uselessness of Computational Power

Uselessness is a situation that often occurs in the longevity of a proof of work blockchain, it refers to the amount of computational power that is wasted in trying to solve the mathematical puzzle. Since multiple tries are often required to achieve the solution, and is not guaranteed that some other miner will not solve the puzzle first this will mean other miners contribution will be useless and this computational power has no other use than serving the network, producing an enormous quantity of computational power that has no other purpose and cannot be redirected to some place, leading to waste.

### Centralization

Since to generate a block is a race where there is no prize for second place and with the growing computational needs to generate blocks peers start teaming up with their computational resources and work together to find blocks, this is called Pool Mining. There are numerous mining pools, and the number of peers joining mining pools is continuously growing. This can lead to a problem of centralization. Figure 3.7[13] shows a pie chart of the top miners between 22-08-2018 and 23-08-2018 (as in who mined the most blocks)

Pools that win the race, get the rewards of the block generation and distribute rewards to the participants accordingly to the policy of the mining pool. As difficulty increases, the more computational power is required, the more users join pools in time may lead to pools owning more than 51% of the network total computing power, leading to centralization.

### Energy Consumption

As mentioned before, the increase in computational powers means increase in energy consumption, Figure 3.8 shows a graph of the amount of estimated energy consumption in one year difference, as the graph shows there is a jump from from 20 TWh to 80 TWh according to statistics of the same source of that provided this energy consumption index chart the Bitcoin network consumes almost as much energy as Austria as of 22 of August 2018 and is expected to continuously grow.

[13] `https://www.blockchain.com/pt/pools?timespan=24hours`

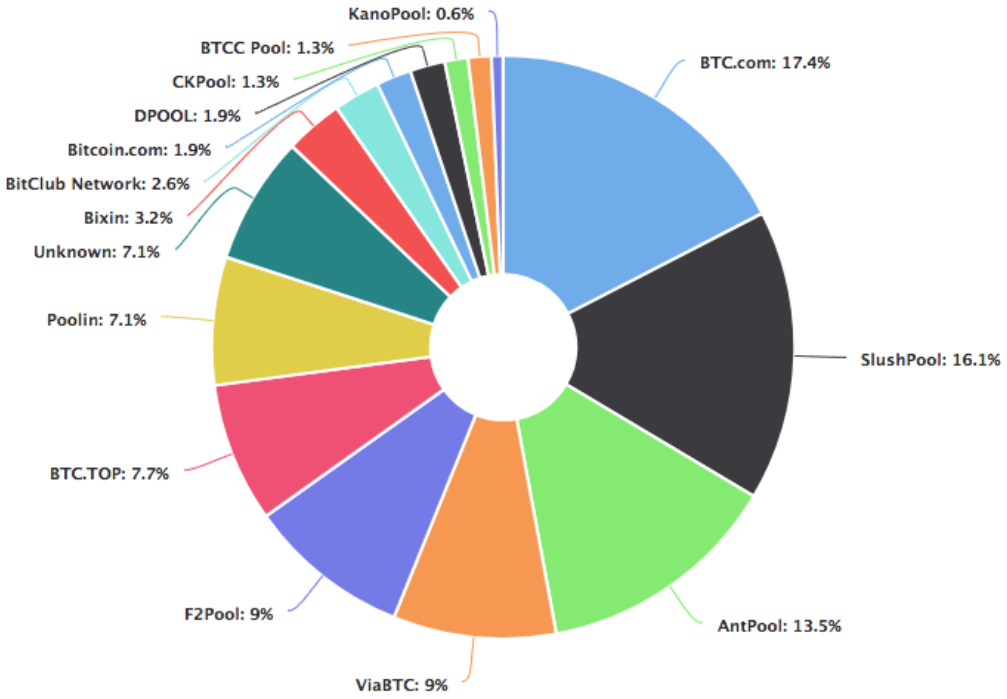



Figure 3.7: Bitcoin Miners Pie Chart.

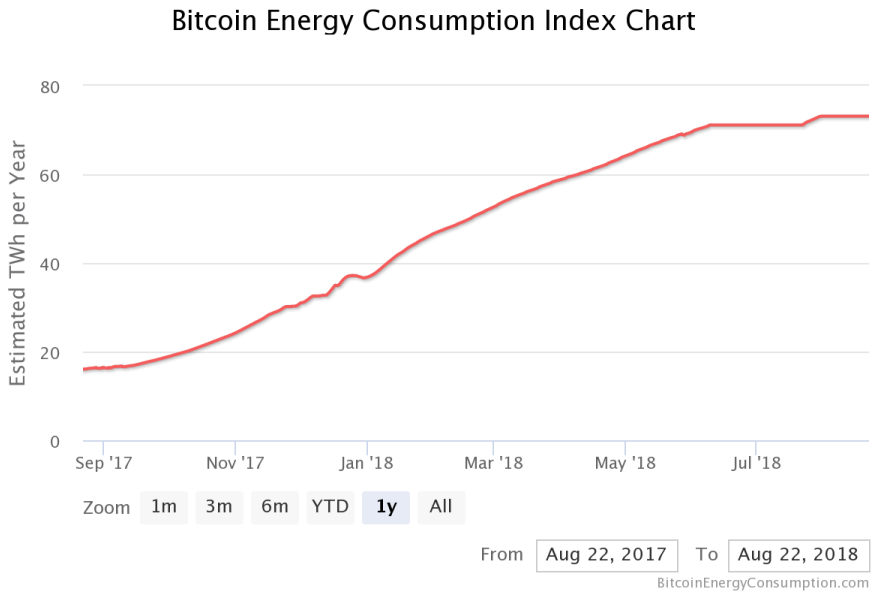



Figure 3.8: Bitcoin energy consumption comparison in one year of difference

### Contributing to Proof of Work

Mining is done by running code on the client side that generates hashes with the hardware that is configured by the miner. Since the difficulty increases, it is expected that a wide number of tries will occur until a hash that

meets all requirements is found, so the more hashrate[14] the hardware has the more likely the user will generate a valid block faster than others.

Since mining is a competition for who finds the block first and with the increased requirements as presented in Figure 3.8, miners got together and form what is called a "mining pool", where miners join their efforts to find a block and are evenly rewarded by their contribution, Figure 3.7 presents various mining pools in existence as of this date.

There are multiple ways to contribute and it all depends in total hash rate of the hardware or combined hardware. In order to achieve requires computational power of which the following methods are explored: CPU power, GPU power, ASIC power.

- **CPU mining** consist in using the CPU of the miners machine. This method is considered inefficient for large networks such as Bitcoin or Ethereum and is discourage as of today because of the low hashrate it produces when compared to other methods that are much more efficient.
- **GPU mining mining** is a standard, it provides a lot more computational power than the CPU and as such is widely adopted by miners, especially for large blockchain networks like Ethereum or Bitcoin. High end graphic cards are usually the chosen ones because it provides the most cost/benefit. GPUs are flexible to use, in other words, the miner is not bound to contribute to one blockchain, but can change the focus to other ones, because they are not built to run in a specific blockchain network, like ASICs. Figure 3.9 shows typical mining setup using GPUs.
- **ASIC (Application-Specific Integrated Circuit) mining** are popular but no easily acquired. It's hardware built for the single purpose of mining, hence the name "application-specific" and as such it is usually more powerful than GPUs. The most popular of the ASIC miners are the Antminers which are Bitcoin mining machines. ASIC machines are application specific, they have the single purpose of solving hash algorithms to one cryptocurrency in specific. If the miner wants to contribute to other blockchains it will require different ASICs. Figure 3.10 shows an Antminer S9.

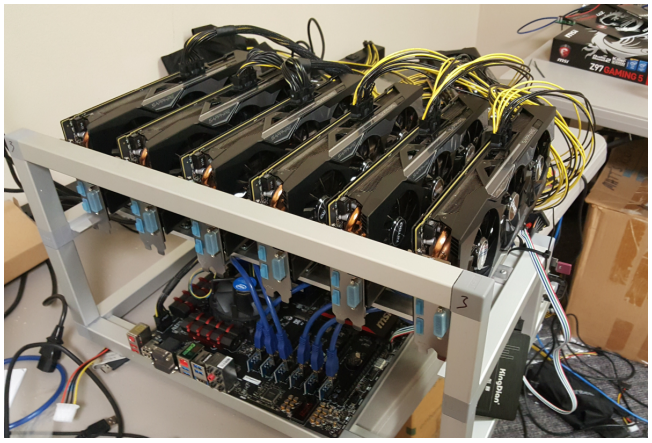


Figure 3.9: GPU mining setup

Figure 3.11 shows an Antminer ( or ASIC ) mining farm, an individual ASIC is very expensive, more expensive then a high end GPU, thus requiring serious investment. These farms are usually placed in a large warehouse filled with shelves of these ASIC machines. These warehouses tend to be located in places where energy costs are very low. Depending on the status of the country towards bitcoin mining, these farms can be considered somewhat illegal.

[14] Number of hashes generated per second

Figure 3.10: Antminer, a Bitcoin ASIC

Figure 3.11: An ASIC mining farm

### Advantages and Disadvantages of Proof of Work

Regardless of what has been stated, proof of work is the longest running consensus protocol in Blockchain implementations. It is a strong and viable consensus protocols and as obvious each protocol has its advantages and disadvantages.

Advantages:

- Arguably the most secure consensus protocol.
- Difficult to generate blocks but very easy to verify legitimacy.
- Very secure and resistant against any network attacks.
- Replaces the trust factor in favour of cryptographic proof.
- Incentive for the hard work.
- The bigger the network the safer it gets.

Disadvantages:

- Energy Consumption because of continuous increase of amounts of computational power
- 51% Attack
- Uselessness of computational power
- Centralization
- Not reliable in private networks or many other use cases.

### 3.8.2 Proof of Stake

Proof of stake was first introduced by Sunny King and Scott Nadal the creators of **Peercoin** stated in Peercoin whitepaper [KN12]. This blockchain network is a hybrid proof of work and proof of stake where *proof-of-stake is used to build the security model of a peer-to-peer crypto currency and part of its minting process, whereas proof-of-work mainly facilitates the initial part of the minting process and gradually reduces its significance"* [KN12]

By itself, Proof of Stake is a category of consensus algorithms that focuses on a peer's stake in the network, it is intended as a viable alternative to Proof of Work in order to counter eventual issues in the long term which are related to the increase in energy consumption (visible in Figure 3.8) and centralization. (visible in Figure 3.7) The goal of Proof of Stake is the same as Proof of Work which is to achieve undeniable consensus in the network but these protocols behave very differently from one another. While Proof of Work heavily depends on computational power, Proof of Stake does not, however, it relies on stake and the availability of peers to achieve consensus. As Proof of Work will bring a higher cost in hardware to keep up with demand, in Proof of Stake anyone with any hardware can participate: can be a smartphone or laptop instead of multi GPU rigs or a warehouse filled with ASICs, thus being significantly more energy efficient then its counterpart.

In Proof of Stake there are no "miners" instead peers which goes by the name of **"forger"** or **"validator"**.

#### How it works

When the need to generate a block comes, any peer holding stake is eligible to participate, so, peers who want to take part of the process place a percentage of their stake in a "vault", being locked for a determined amount of time until a peer is chosen as the current block creator. In more accurate terms, this translate into a "special" transaction from validator to vault. How peers are chosen as the current block creator will depend on the implementation of the protocol, not having a fixed guideline. Peercoin is the first blockchain to introduced a Proof of Stake mechanism, it makes a random selection from the pool of available validators at the time based on

coin age which means how long the validator as been holding the cryptocurrency. In other words, higher the coin age the higher the probability to get chosen. [KN12]

If a peer wins the right to generate the block the next step is to follow all the rules for a block to be considered valid, which means ensure valid transactions. Since there might be no incentive to add blocks to the longest chain, this again depending on the implementation of the protocol, it is by the decision of the validator which competing chain to add a block to, but naturally validators add blocks to the longest chain for consistency and value of investment.

Like Proof of Work, Proof of Stake is not safe from malicious actions. Malicious validators can attempt to commit fraudulent transactions or try to censor others, but if the validator attempts to do so he will loose a big chunk of his vaulted stake (the currency is never recovered) since confirmations are required this means fraudulent transactions or censorship will be easily detected and since the cryptocurrency is locked, this will make loosing some if not all of the malicious validator investment. So, this system of locking heavily disincentives fraudulent validators. Another worrying issue is the Nothing at Stake, which will be described in the following section.

As mentioned before Proof of Stake depends on the implementation of the protocol thus making it a category of algorithms. **Ethereum Project** defines two major types of Proof of Stake algorithms, **Chain Proof of Stake** and **Byzantine Fault Tolerance Proof of Stake**. [Vuk17, pro]

**Chain Proof of Stake**, the right to generate a block is pseudo-randomly assign to an user that holds some amount of stake. This block must then point to some previous block, which should be the latest block of the longest chain. [Vuk17, pro]

**Byzantine Fault Tolerance Proof of Stake**, works with a system of votes where voters are randomly assigned the right to vote for proposal of blocks. There are multiple blocks to be validated and users send votes to which block to be added to the blockchain. Its a multi round process for each voting session. In the end of the process, voters must agree on which block will be added to the chain, but if at the end of the process no decision is made then the whole process is restarted until a block is added. [Vuk17, pro]

### Nothing at Stake

Nothing at stake is a concerning issue in Proof of Stake networks. Although at of the date of this work is just a theory since there have been no occurrences. ”Nothing at Stake” does not mean a malicious attack because this can be considered by definition a normal behaviour of the protocol. At the time of forging a block a forger/validator must be chosen, but there could be a case where one or more users managed to arrive at the same threshold, which determined who gets to be the validator. This issue can relate to **longest chain rule** as Proof of Work incentives users to work on one chain (the longest chain) in order to be rewarded because no rewards are given by working on different competing chains. Since this protocol requires reduced computational resources but only a mix of factors related to the stake of the user it means there is minimal work involved when there are competing chains and the case of multiple forgers at the same time it results on a fork of the blockchain and now majority of the network has to agree which chain to follow. This is were the ”nothing at stake” kicks in, a peer can vote on every fork because it is in his best interest to do so (economic wise), then remove his investments from the losing chain votes and place it on where the majority of voters lies, since the peer loses nothing, this incentives users to vote on multiple chains because they have ”nothing at stake”. This also leads to the double spend attack but much more difficult where a user can make a transaction on a chain wait for it to get confirmed (again multiple people forging on multiple chains until majority agrees) then switch to the another and the other chain gets the blocks transactions reverted. This is somewhat unrealistic because, for example, there needs to be 99% of users on one chain and 99% of users on the other chains (same users forge on competing chains) and that remaining 1% user can effectively double spend. By default peers tend to focus

on just one chain, which reasons may include, value of investment.

The Casper protocol, proposed by Ethereum Foundation, which is a Proof of Stake mechanism but at early phases a Proof of Work/Proof of Stake hybrid, solves this problem by introducing punishments for voting on the multiple chains. [BG17]

Nothing at stake tends to be just on the theory side because it may affect the credibility of the network, and since these networks require monetary investment, it may reduce a malicious users from doing so, but it is still possible to do so.

#### Advantages and Disadvantages of Proof of Stake

Just because its an energy efficient solution to Proof of Stake it still has its disadvantages. Which should be taken into consideration.

Advantages

- 51% Attacks are almost impossible and unreliable, attacker(s) need to own more than 51% of the total cryptocurrency of the network.
- Lower energy costs and lower computational power, meaning less need for mining.
- Increased distributed consensus as more peers can participate.
- Better long term solution then Proof of Work.
- The bigger it gets the safer it gets

Disadvantages

- Dishonest and malicious validators.
- Requires investment to contribute to the generation of block instead of just contributing with computational power (PoW)
- Exclusive to digital currency based consensus.
- May disincentive small stakeholders.

### 3.8.3 Delegated Proof of Stake

Delegated Proof of Stake while similar to Proof of Stake, there are some differences. It is intended to incentive smaller stake holders to continue to participate in the voting process. This happens because larger stake holder have essentially a higher chance of generating blocks, which means smaller holders may not be so interested.

This new system essentially becomes more democratic. In Proof of Stake, stakeholders use their stake to gain a chance to create a block, in Delegated Proof of Stake, stakeholders use their stake to vote on a user that will act as a validator on their behalf and from a list of top electors, someone gets chosen to become the current block creator and the rewards coming from the block are then shared among with those who voted for the creator. A user can essentially always be chosen as the current block creator but as the networks grows larger it becomes

increasingly more difficult to be a creator. This also disincentives users to do malicious actions because trust it a big factor, if the voters don't trust a user, he will never be picked as a creator. i.e: for censoring transactions by not including them in the next block. But the elector can always get voted off if malicious intentions are actions are detected.

This system holds a problem because it enables everyone to from small to big stakeholders, it centralizes the network.

#### Advantages and Disadvantages of Delegated Proof of Stake

Although it may seem as an improvement to Proof of Stake because it is more democratic it holds its problems.

Advantages

- Better distribution of rewards.
- Every can participate. (from small to large stakeholders)
- As it works by voting, even small stakeholders can a top elector.
- Democratic system so it incentives users to behave in order to get voted.
- Scalability and no powerful hardware is required.
- Easily avoid malicious users.

Disadvantages

- Can lead to more centralization.
- Censorship of transactions.
- Needs a big network for the system to work viably. Therefor not useful in private networks.

### 3.8.4 Hyperledger Fabric Protocol

Hyperledger Fabric is a private and permissioned blockchain, so different rules apply when compared to public blockchains. Hyperledger Fabric goes by identity over anonymity. Conditions apply in order to become a peer. The same going for participants. This network uses a different way of achieving consensus because trust is a factor for its peers. Because of the size of the network, proof of work or proof of stake are not considered as possible consensus choices because are not ideal for smaller networks. Hyperledger Fabric is the blockchain framework to be used in the work described in this dissertation. Therefore this section will be a simple introduction to the Hyperledger Fabric consensus protocol, Chapter 5 will correspond to a detailed view of the Hyperledger Fabric and all its components.

#### Consensus on a private blockchain like Hyperledger Fabric

A complex system of consensus takes places where it has multiple levels of confirmation before it reaches the blockchain. In public blockchains the usual case is having a peer which is called a miner, or any other appropriate designation based on protocol, like validator in Proof of Stake. In Hyperledger Fabric there are two major

types of peers, **Endorser peers** and **Orderer peers**. Endorser peers endorse transactions and orderer peers receive endorsed transactions and create blocks. So these Orderer Peers are the consensus. Consensus comes in different supported algorithms, Simplified Byzantine Fault Tolerance, Kafka with Zookeeper and Solo, although Solo consensus is for development purposes only.

Every request must strictly follow certain rules and must come from a registered user within the Certificate Authority. Transactions need to be endorsed by Endorser peers which determine if the transaction follows the rules. After being endorsed individually needs to be submitted to the Orderer Peers which check if the transactions follow the rules and then bundled into a block.

## 3.9 Forking

Forking is common on blockchain, because of the nature of the Longest Chain Rule, such as competing chains or majority attack. Besides cases, forking may also happen due to changes entirely on how rules are validated or addition of new rules among others. This leads to a **Soft Fork** and **Hard Fork**. Various blockchain projects are direct forks from other blockchains projects which occurred for many reasons, leading to a change of rules where the old peers no longer recognized blocks from the new peers.

Previously mentioned cases of blockchain forking, which do not relate to changes in software:

- **Competing Chains.** When peers receive different valid blocks, roughly the same time, causing competing chains until a new higher block is found.
- **Majority Attack.** Causes blocks to be modified, since it is not possible, the resulting chain is a fork of the original chain. But by the **longest chain rule** it is considered the valid one.

### 3.9.1 Soft Fork

Soft fork is a change of rules in how new blocks are generated but are still seen valid by the same software, making it backwards compatible. Soft forks can also lead to hard forks if the way blocks are generated is no longer accepted by the same unaltered software. An obvious example is when the new rules for generating blocks want to be reversed. Peers following old rules will still see the new blocks as valid ones, but not the old blocks.

### 3.9.2 Hard Fork

Hard fork in a blockchain is when a change of rules in how new blocks are generated occurs and the current software does not consider it valid anymore, in technical terms, its a change of current software.

Figure 3.12 represents a hard fork. A hard fork occurs at a certain block height and the chain is ”split” in two, where the peers who contribute have a choice of whether follow the rules or continue to follow the old rules, but the more peers follow the new rules the more secure the blockchain ”post-fork” is. The blocks with ”old rules” tag continue to abide the old software, the blocks with the ”new rules” abide by the new software, these blockchains are completely independent from the other, as in, doesn’t affect the original. Simply at some point in time, these two were one.

Hard forks occurs by a number of reasons. Most obvious cases is to attempt to fix problematic issues that can only be fixed by reverting transactions in blocks. Since the power is not within the developers but the commu-

nity, it is the community that must reach consensus on whether the problem or solution or upgrade or any other reason must occur. A famous example is the DAO incident in the Ethereum blockchain, where it led to Ethereum and Ethereum Classic.

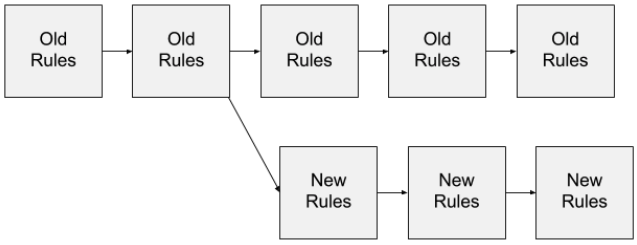



Figure 3.12: Hard Fork

**Example of Hard Fork: The DAO attack**

The Decentralized Autonomous Organization (The DAO), was a project, which expectations for it were higher than predicted, and during its Initial Coin Offering (ICO) raised 150 million dollars worth of Ether, Ethereum cryptocurrency. [Amm16] The DAO running on Ethereum, handles everything by code, but the smart contract suffered from bad engineering design. This bad design led to a malicious user exploiting it. The design flaw was due to updating records only after the currency was moved. This lead to a recursive exploitation where the attacker was withdrawing currency and before the smart contract would update the records the attack would interrupt and withdraw again. The issue with the developers of the DAO is that they are powerless, in the sense is that the community of Ethereum must reach consensus on the decision to make regardless of what happened. [dbc]

Soft forks proposals were introduced to serve as temporary solutions but the solution led to a hard fork on the Ethereum blockchain in order to restore stolen funds from The DAO, and give back to the users. This lead to the new chain still named Ethereum and the old chain was named Ethereum Classic. [Amm16]

This issue is rather controversial because the contract operated in its capabilities and did exactly what it was told to do, so it can be said that the attack was completely legit, it was not a problem of Ethereum but a problem of The DAO. This ”history rewrite” caused several criticism because if other contracts have the same fate then other users will want the same treatment if other problem occurs.

## 3.10 Blockchain Use Cases and Benefits

Blockchain can be used when the properties of a tamper proof shared ledger among participants can be taken advantage of. Its first use case was in financial situations, which was the objective of Bitcoin, to have a decentralized currency without the need of a middleman.

Besides financial uses, blockchain has many other important applications in today’s society. Areas like Healthcare [XSA+17, XSA+17], Decentralized Storage [sia, sto], Media, Internet of Things [CD16], Professional Services, Public Administration, Real Estate, Supply chain, Identity Management, Fraud Prevention are some of the areas blockchain can be taken advantage of. [CPVK16, Gup17]

Blockchain can also be beneficial to all sort of business applications. For business applications, a private and permissioned blockchain is more ideal, as identity and permissions permits only authorized users to interact with the network. [Gup17]

# 4

# Smart Contracts

*In this Chapter, tends to show a broad understanding of what smart contracts actually are and how blockchain permits smart contracts to have an environment where they shine. Not only limited to a definition but understanding how crucial and the importance of a well defined and tested smart contract. Additionally this Chapter describes the difference between smart contracts and applications, and how they are different in behaviour and functionality.*

## 4.1 Definition

Smart contracts are agreements between entities that are translated into code. Described by Nick Szabo smart contracts can *"secure many algorithmically specifiable relationships from breach by principals, and from eavesdropping or malicious interference by third parties"*, this because many *kinds of contractual clauses (such as collateral, bonding, delineation of property rights, etc.) can be embedded in the hardware and software we deal with* [Sza97].

They are to be "*run exactly as programmed without any possibility of downtime, censorship, fraud or third party interference.*" [eth] As such, smart contracts can be compared to vending machines. What this means is that smart contracts, like vending machines, have their entire logic written within. The input will decide without

third party interference what is the correct output. Based on this description it is clear that smart contracts contain the rules, conditions and penalties written in code, therefore once the smart contract is called in the sense of calling functions, they will automatically enforce, validate and verify the current conditions and output taking into consideration the input provided.

Ethereum Co-Founder Vitalik Buterin describes smart contracts as *"the program runs this code and at some point it automatically validates a condition and it automatically determines whether the asset should go to one person or back to the other person, or whether it should be immediately refunded to the person who sent it or some combination thereof"*

## 4.2 Smart Contracts without Blockchain technology

Smart contracts can exist without the blockchain technology, but it must be noted that blockchain technology allows for smart contracts to shine and taken the most advantage of. Smart contracts can be created on top of a traditional database, the problem with agreements is that smart contracts must be immutable, uncensored and unable to be interfered, which with a centralized entity it may pose contradictory. Blockchain provides this component to be check, as it is one of the major properties of the blockchain. Additionally no need for a central entity to keep tabs on these contracts due to the decentralized nature of the blockchain and thus ensuring the previous features are kept true.

## 4.3 History of Smart Contracts in Blockchain

The early phases of smart contracts started in the first blockchain, Bitcoin, which has the capability of built a limited number of scripts due to using a non turing complete language. [ABC+18] this is also due as it was designed with only a single purpose in mind and not to be to expand from that point. Ethereum and Hyperledger Fabric are blockchain, respectively, implementations, that permit unlimited creativity in their smart contracts development which allows for a much larger set of tools that permit developers to create applications that harness the power of the smart contracts. And allowing for decentralized applications.

## 4.4 Smart Contracts with Blockchain Technology

Smart contracts, shine with blockchain technology, because of the properties of the blockchain. Smart contracts provide **default consensus**, as explained in an interview with Gavin Wood in Dutch Blockchain Conference 2016 [dbc], they will act as they were programmed to act until the a majority of peers says different, but not a single entity and this is why smart contracts are ideal with blockchain in the mix.

When smart contracts are deployed on the *production* network it will include every issue not caught during development, so the developers of these contracts per say, are powerless if contract changes are required, there must be a consensus by the maintainers of the networks, this in private and public networks, that must decide if a contract is to be changed.

Blockchain properties also permit that any service that depends on these smart contracts to continue to exist. As they will be available for usage as long as the network continues to acknowledge their existence.

## 4.5 Importance of Security in Smart Contracts

The security of Smart Contracts when associated with Blockchain, have their security increased by blockchain properties, therefore they will operate without fraud, censorship, third party interference and also as equally important, constant availability, as long as there are peers maintaining the network, and able to execute it, then the contract swill always be available.

But the very nature of the smart contract can pose a potential security risk [DAK+16] to the application or applications that rely on these smart contracts to work. If the source code that reflects the deployment is visible or by means of reverse engineering then so its bugs, exploits and security vulnerabilities. This means if a smart contract has a severe security issue that was overlooked or ignored during development this may not be solved in a quick manner, and its effect can be permanent.

It must be noted that the blockchain eco-system is not the problem, the problem is with the developers of the smart contract that introduced code security flaws, in short, human error is the problem.

In sum, creating smart contracts require serious thought and the more complex the contracts are, the more attention to detail and testing it requires before being deployed on the production-ready blockchain network for its users to use.

### 4.5.1 The DAO incident

The DAO stands for Decentralized Autonomous Organization. And as previously explained of the hard fork on Ethereum that forked the blockchain into Ethereum and Ethereum Classic, the DAO incident was because of bad code on the smart contract. Since this was visible, a malicious user could replicate the contract in some way and use it to exploit the DAO contract in order to siphon a large amount of Ether. The contract did had some safety features which locks the currency for a total of 39 days, so a solution had to be found in less than that time. As stated by Gavin Wood on an interview in the Dutch Blockchain Conference 2016, ”the developers are powerless”, only the community can determine what action to take. Since the amount of Ether stored in the DAO blockchain address corresponded to 14% of the total circulating Ether at the time, the decision was to hard fork which led to revert transactions in order to recover the ”stolen” funds. This again led to controversy because the smart contract operated as programmed and the malicious user was simply taking advantage of it, so there is an argument to be taken. Nonetheless, the Ethereum community reached consensus and a hard fork occurred, reverting the transactions related to the DAO incident.

## 4.6 Decentralized Applications (DAPPs) vs Smart Contracts

Smart contracts are different from Decentralized Applications. Smart contract is an agreement written in code that resides in the blockchain or any other appropriate immutable support.

In the traditional sense, an application has a frontend and a backend which connects to some database in order to fetch and update information. But a traditional application has its backend running on a centralized servers.

A Dapp or Decentralized Application, on the other hand, has a front end and its backend runs on the blockchain by the means of one or various smart contracts invocations to fetch information from the blockchain or perform to transactions in it. Essentially a DAPP is an application that sits on top of the blockchain eco-system to thrive.

i.e: An application can be anything, Ethereum describes three different types of decentralized applications. The first are financial only applications, currency management, creation of cryptocurrency, among others. The

second type are applications that use currency but there is more to it, payment for services, rewards for doing something, etc.... And the third type is any application that doesn't fit into the previous two categories. [Woo14]

Decentralized Applications may rely on other components like decentralized messaging and decentralized storage.

## 4.7 A Practical Example

A smart contract for account balance would contain several functions related to balance. Some functions would be, `checkBalance()`, `send(toB)` The first function `checkBalance()`, the behaviour is obvious, but the second function `send(toB)` has different behaviour. When the user wants to transfer value from A to B, certain conditions must be checked by the smart contract. Is user A an existing and valid user? Does user A have permissions to call this function of this smart contract? Is the user B an existing and valid user? Can user B receive balance? Does user A have the sufficient balance to send to user B? This last question is the most important one

If all answers are true then the contract will automatically transfer value from A to B and reduce the current account balance of A and increase the balance of B. If any step fails, the contract will have different outputs. But the idea is that the smart contract automatically determines what is the appropriate action to take based on a set of variables provided so no third party will interfere with its behaviour. For example if user A does not sufficient balance, then the transaction ignore, or return the funds back to user A.

Smart contracts are made to be always available and cannot be shutdown and cannot be changed by anyone other than by consensus.

In networks like Ethereum, smart contracts cannot be changed, once deployed they are to be in the blockchain as long as it exists. The only way to make a change is to upload a new contract and have the community accept the new smart contract as the valid one and forget the old one. But these smart contracts can implement `selfdestruct`, a function that "kills" the smart contrac.

In networks like Hyperledger Fabric, smart contracts do not technically reside in the blockchain, every peer that can hold the smart contract, holds a copy but these execute in an isolated manner to not be interfered by the peer. In order to update the contract, an upgrade must be requested and consensus must be achieved, which replaces the old with the new contract.

# 5

# Hyperledger Fabric

*In previous Chapters, Hyperledger Fabric was referenced several times. The reason is because this blockchain framework is the one chosen to develop the work described in this dissertation. Since it is the basis of the work, needs to be clearly described, but in less technical terms. Because this Chapter objective is a description of what Hyperledger Fabric and its various components, such as peers, Blockchain and Consensus are.*

## 5.1 Hyperledger Fabric Business Network

Hyperledger Fabric is a private and permissioned blockchain for blockchain oriented business network which primary aspects are **trust, identity over anonymity, shared ledger and privacy.** Providing the ability for private transactions only executed by registered and known participants.

Hyperledger Fabric is a private, permissioned network thus permits the creation of business networks composed of multiple organizations, which can be suppliers, manufacturers, banks, from small commercials shops to large ones, among others. These participate in multiple channels where they interact in private channels independent from others where each shares a unique single ledger common to all participants.

Consensus takes a different shape as well. It does not uses Proof of Work or Proof, but a special mechanism.

## 5.2 Clients or Users

Clients are users of the network who interact with the blockchain via external programs. These clients must be registered within the Certificate Authority of an Organization in order to participate. If the user is accepted and successfully enrolled, these users can only interact with the network via programs built with the Fabric SDK, which means conducting transactions proposals to update the blockchain or simply querying it.

## 5.3 Peers

Instead of having one type of peer, such as miner or forger, Hyperledger Fabric is compromised of various peers which together form the whole network, where each peers holds certain responsibilities. There are peers, endorsers, anchor, leading and orderer peers.

### 5.3.1 Peers, Endorsers, Leading and Anchor Peers

Peers simply receive updates from the ordering service and maintain the blockchain, assuming default network configurations where peers can hold the blockchain and forward requests to other peers.

Endorser Peers, receive requests in the shape of transactions proposals where they endorse it. In other words, providing a valid signature which is required for query or writing to the blockchain. In order to provide a valid endorsement a certain number of steps must be checked. This is done locally without the interaction of other peers, but the procedure to provide an endorsement must be equal to all other endorser peers.

Anchor Peers, are the communication bridge between different organizations. They can be endorser peers or leading peers.

Leading Peers: each channel must have a leading peer. these peers are the communication bridge between peers and the orderer peers. The orderer service, which is composed of orderer peers, creates and broadcasts a new block to the leading peers and the leading peers distribute the block to the peers in the channel where they belong to. They can be endorsers and anchors.

### 5.3.2 Orderer Peers

Orderer peers control the Ordering Service. This serving is a complex system and pivotal to the stability and consistency of the Hyperledger Fabric network, which in others words mean consensus. In general terms, these peers and service are a must in the network, as they provide the consensus for other peers to maintain their blockchain copy. In less general terms, the orderer service provides a guarantee to the delivery of blocks to peers, so they can update and maintain their blockchain copy. These peers receive the transactions with endorsement, which then must follow a certain number of steps to prove endorsements and transaction are also valid, and if valid create blocks of transactions to be broadcasted to the other peers.

Orderer peers allow for consensus in the blockchain network. They guarantee messages are broadcasted and received by the peers and will also guarantee that received transactions are either correct or incorrect.

## 5.4 Organizations

Organizations are the members of a Hyperledger Fabric business network. Peers are associated with a single organization which is identified by the membership providers of that organization or a common membership service provider between organizations. When a network starts, it starts with main channel and every organization is present. Organizations have their anchor peers which serves as the communication bridge between other organizations.

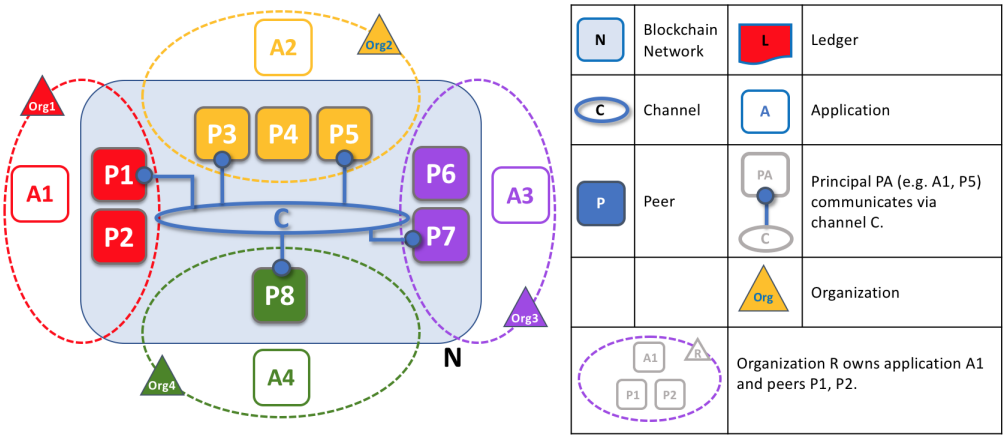



Figure 5.1: Diagram of a network with multiple Organizations [Hyp18]

Figure 5.1, by *Hyperledger Fabric Official Documentation* [Hyp18], represents a channel in a Hyperledger Fabric business network with multiple organizations in a single channel with multiple peers in each Organization and their own peers communicating with the channel and external programs and which peers is it associated with. Additionally each organization has one or more Membership Service Providers (MSP) that issues and manages the existing and revoked identities of its participants.

## 5.5 Membership Service Providers

Membership Service Providers is an abstract interface providing the ability to identify its participants in the network. All participants, users and peers, must be identified in order to be in the network, since Hyperledger Fabric network provides identity over anonymity. There can be multiple Membership Service Providers in a network not being limited to one network wide service.

The Membership Service Provider has the following functionality:

- Create credentials.
- Revoke credentials.
- Generate signatures and verification of the same.
- Provide identity to participants.

Mappings for the Membership Service Provider:

- **One-to-one**. One Organization to one MSP.

- **One-to-many**. One Organization with multiple MSP.
- **Many-to-one**. Many Organizations to one MSP.

Hyperledger Fabric Developers recommends the approach of using one-to-one mapping of Membership Service Provider and Organization, but organizations are free to choose they own mapping.

Although many MSPs in one organization is useful, this can come into practice when organization needs to define multiple levels of access on their participants. Must be noted that a peer can only have one identity independent of how many MSP exists within the organization. So this can lead to issues of organization participants not recognizing participants of the same organization. [Hyp18]

### 5.5.1 Certificate Authority

Certificate Authority, CA, is the Hyperledger Fabric default implementation of the Membership Service Provider interface named Hyperledger Fabric CA.

Implementation of a Membership Service Provider mapping is usually composed of the following components:

- **Root Certificate Authority**: manages the identity and certificates, which are, closely guarded not to be leaked. Used to sign intermediate certificates issued by the Intermediate Certificate Authority.
- **Intermediate Certificate Authority**: manages and issues certificates and identity signed by the root certificate. Intermediate Certificate Authorities can link to other Intermediate Certificate Authorities.
- **Client Certificate Authority**: used to connect to the Fabric CA Server which correspond to the Intermediate Certificate Authority
- **Fabric SDK for CA**: used instead of Client CA. Uses the Fabric SDK to build applications to communicate with the Intermediate CA.

## 5.6 Channels

The Hyperledger Fabric network is composed of channels where organizations interact each other, and channels are a ”sub-net” of the Hyperledger Fabric business network. Multiple channels may exist at the same time and peers may belong to as many channels as possible, assuming they are invited or they created it. Each channel is independent from other channels and have separate blockchains, so, information is contained only in a channel and no leaks can occur. This method allows for clients and peers of organizations to have increased privacy in transactions without the knowledge of others.

Setting up a channel requires additional maintenance, and since these are independent, channel configuration is required. When a channel is created, the anchor peers, the policies and membership among other requirements must be determined and then stored initially in the genesis block of that channel. Besides channel configuration issues, channels have their own blockchain visible only to the channel which means peers of said organizations have to maintain additional ledgers if they belong to more than one channel.

Additionally smart contracts also have to be deployed in the channel. but the versions and contents of the smart contract is independent from other smart contracts deployed in other channels, so a change in one channel only affects that channel.

As an example considering smart contracts: Channel C1 and channel C2 and both have smart contract S. If S requires a change in C1, and if the change is approved by the endorser peers and Orderer Service then the changes in S only affect C1. C2 still goes by the older unchanged version.

#### Analogy - Channels and Group Chats

Channels work similar to social platforms or business social platforms like **Discord**[1] or **Slack**[2], where you engage in private conversation with a user or a group of users, users outside of this "chat room" cannot see the data in it. In Hyperledger Fabric channels is the same, the data is only visible within that channel and is not visible to other peers or users that have not been invited to that channel or are even aware of the existence of the channel. This mechanism allows a level of confidentiality and increased privacy between two or more entities, for example, enabling file sharing between two entities without the slightest knowledge of others.

## 5.7 Transactions

Transactions in the Hyperledger Fabric are of two types. Transactions that affect smart contracts and transactions that interact with the smart contracts.

Transactions that affect contracts are called **deploy transactions**, these transactions install new smart contracts in the corresponding channel or upgrade the existing smart contracts to a newer version.

Transactions that interact with smart contracts, called **invoke transactions**, since they call `invoke` method of a smart contract, are able to query the blockchain and/or update the blockchain. Transactions ready-only are different from transactions that update the blockchain. Read-only only requires endorsement from endorser peers. If a read-only transaction attempts a write this will be ignored but the transaction can still succeed just ignoring the write. This happens because a transaction proposal for query is different than a transaction proposal to write, requiring the Orderer Peers approval to update.

Transaction that updates the blockchain also requires the endorsement from endorser peer(s) to be submitted to the orderer peer(s). These peers, controlling the Orderer Service, will update the blockchain by creating new blocks. Must noted that query transactions do not show up in a block because it does not affect the blockchain as it only reads from it so endorser peers just serve the request.

Additionally, transactions life cycle are either valid or invalid. The following enumeration describes the difference, in HF:

- Valid transactions will update the world state.
- Invalid transactions are still logged in the blockchain but will not update the world state.

### 5.7.1 Endorsements

Endorsements corresponds to endorser peers signatures that a user submitting a transaction proposal must acquire if the transaction is to be considered valid when submitted to the ordering service. The amount of endorsements required for a transaction is defined by a policy criteria in the deploy transaction of a smart contract

[1] Gaming oriented social platform
[2] Business oriented project collaboration platform

but this policy criteria can change if there is consensus among the endorser peers in the channel(s) the change is taking place. Endorsers do not need to communicate with other endorsers when endorsing but will go by the same rules as all other peers and the ordering service will determine if the endorsements were correctly made and the necessary endorsements was achieved.

Endorsements policies can take different shapes but go by logical expressions, and each endorser peer may have a different endorsement "weight".

Assuming a channel with three endorser peers, we have an Endorser set, `E = {e1 ,e2 ,e3}`, where `e1` is Endorser 1, `e2` is Endorser 2 and `e3` is Endorser 3.

- `e1 AND e2 AND e3`
- `e1 AND (e2 OR e3)`
- `At least two of E`
- `At least one of E`
- `Any one of E`
- `Any one of E except e2`
- `e1 = 10, e2 = 20, e3 = 30`. And the endorsement must be `n >= 30`.

The process of getting an endorsement is a set of policy criteria that must be checked, with the aid of special smart contracts. This must be done before an endorser peer may provide a valid signature for the specific transaction proposal. The following steps the steps required for an Endorser peer to provide a signature to a transaction.

- Transaction comes from a valid user
- The same transaction has not been broadcasted before. This avoid duplicate transactions.
- User is allowed to perform the execute from the smart contract.
- User is allowed to interact with the channel.
- Simulation of transaction and return of result to peers.

All previous steps require usage of special smart contracts known to the Endorsers and Orderer in order to enforce the various steps endorsement of transaction proposals. If all the previous steps are valid then a signature is provided The following expression is a command that instantiates a smart contract :

```
cli peer chaincode instantiate -o orderer.example.com:7050 -C mychannel -n sc -v 2.0 -
-c '{"Args":[""]}' -P "OR ('Org1MSP.member','Org2MSP.member')"
```

The previous command starts the smart contract `sc` on the channel `mychannel` with version `2.0` with the policy criteria stating only one endorsement from a member of either organization 1 or 2 is required and the transaction is sent to ordering service endpoint `orderer.example.com:7050`

### 5.7.2 Privacy in Transactions

Private transactions is information only visible to a set of participants. In Hyperledger Fabric v1.1, as of this date, this is achievable multiple ways.

- Join a Hyperledger Fabric business network. By default all transactions are private to outside users not affiliated with the network.
- Restrict other users on read-only and/or limit what they can read or inspect.
- Joining a Channel or creating a channel and inviting peers to conduct transactions only visible within the channel.
- Private transactions. These can take place in any channel where data or some part of it is kept private. This is achieved by special smart contracts functions that handle private data similar to channel visible data is done and providing additional details that specify who can have access and store the private data. Private transactions are special because they are sent to a private database on the peer and the ordering serving does not see the output, but in order to prove the transaction happened the data of the transaction is hashed.

Privacy is one of the main aspects of a Hyperledger Fabric network. Creating channels has its benefits because it enables two or more entities to engage in transactions and a dedicated blockchain for it without the knowledge of other entities present in the network. Although it does have to deal with additional configurations such as setting up smart contracts, peers role and maintain a new blockchain. Because of this, engaging in private transactions without creating new channels for it can be a benefit.

Any of the two approaches will entirely depend on the case at hand.

## 5.8 Block

Figure 5.2, Parth Thakkar,Senthil Nathan and Balaji Vishwanathan in *Performance Benchmarking and Optimizing Hyperledger Fabric Blockchain Platform* [TNV18], represents the structure of a block in Hyperledger Fabric version 1.0. Description over a block is described in detail in Chapter 3. Each block in HF is composed of multiple components. The blockchain header, list of recent transactions and block metadata.

#### Block Header Component

Block header is composed by 3 components: 1) The **Number** which corresponds to the block height[3]; 2) The **Previous Hash** which corresponds to the hash of the previous block; 3) The **Data Hash** corresponding to the hash of all transactions, which is similar to the Merkle Tree used in Bitcoin and other blockchain implementations.

#### Transactions Component

Transactions are composed of three main components. 1) **Transaction proposal**, is divided into 3 sub component. The first is the header of a transaction, containing data about the chaincode version, channel id, among

[3] Number of blocks starting from Genesis Block

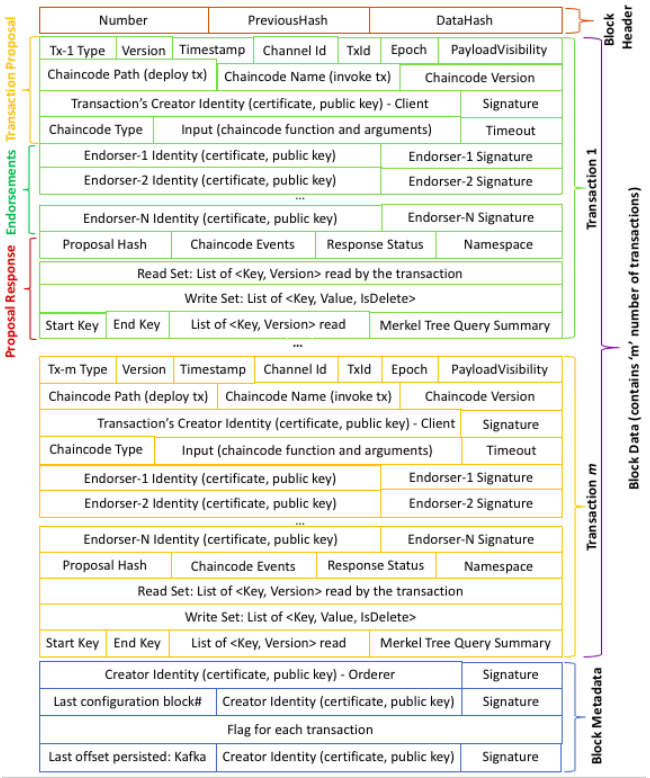



Figure 5.2: Hyperledger Fabric blockchain block structure [TNV18]

others. The second is the signature of the submitter of the transaction. Third is input parameters which is used create or update a entry in the world state; 2) **Endorsements**, corresponds to the endorser peers signature, as explained in Section 5.7.1; 3) **Transaction response**, corresponds to the before and after values of the world state caused by a smart contract. If the transaction is flagged as valid, then the after values will be applied to the blockchain.

### Block Metadata Component

Data in this component is not hashed, but added when a block is created. [Hyp18] It is related to who created the block, as well as the flags for every transaction in order to tell which transactions in the block are valid and invalid. The difference from invalid and valid, is that valid transactions will affect the world state as explained in Section 5.9.1. Invalid transactions will not update the world state in any way but stored in the block for record keeping.

## 5.9 Blockchain Ledger

The blockchain has two components. the **world state** and **blockchain**, as described by Hyperledger Fabric Documentation [Hyp18]. The world state component is the most most current state of every entry in the blockchain, the Blockchain is transaction valid and invalid. Figure 5.3, by *Hyperledger Fabric Official Documentation*, is a diagram representing both the components. The components together are called **Ledger** or more accurately **Blockchain Ledger**.

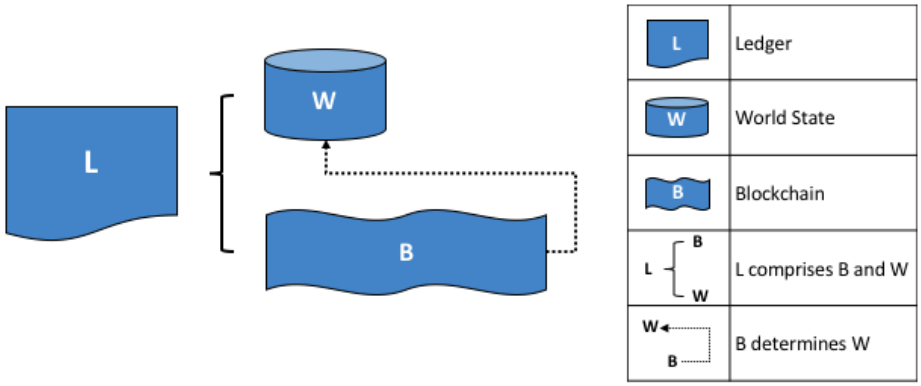



Figure 5.3: Blockchain structure in Hyperledger Ledger Fabric. [Hyp18]

### 5.9.1 World State

The world state component contains the most recent state for any given entry in the blockchain for every valid transaction, which is key-value pair. The world state is in constant change as it only keeps one record for any given key.

For example: User A registers a file in the blockchain with a unique key `FILE01`, with a status of `Available`. The same user queries blockchain with with the key `FILE01`, the information returned will be the same he entered, with status `Available`. User A then updates the status to `Deleted`. If user A of the file or any user (with given permission by the user A) queries with the same file key (`FILE01`), the information returned will be different were the status is now `Deleted`. But, if another user attempts to change any field identified by key `FILE01` other than owner the transaction will fail, but will get registered in the blockchain as an invalid transaction but won't update the world state.

By definition the world state contains only the successful transitions of state for any given key, translating to transactions flagged as valid.

Must be noted that since the state component is subset of the blockchain, the existence of this the state component will depend on the deployment configuration of the network. It is up to the developers to determine if a world state is necessary. Additionally, it is fairly simple to reconstruct this component because it is a subset of the blockchain.

The state component resides on the peers, anchor peers, endorser peers, leading peers and is also maintained by the same. The orderers on the other hand do not hold or maintain this component, only the ledger component, for fault tolerance.

### 5.9.2 Blockchain

The **Blockchain** component, contains all the successful and unsuccessful state transitions for any given key, translating to valid and invalid transactions, it works as a record keeper for everything that undergoes in the channels of a Hyperledger Fabric network. By logic, the blockchain is the only component that is actually required, because the world state is a construct from the blockchain, therefore the blockchain serves as the single shared truth for all records.

Where the blockchain resides depends on the configuration of the network. By default every peer holds a full copy of the blockchain, but optionally only the orderers or a subset of orderers maybe be the ones who hold the full copy of it.

Depending on the type of peers (Peers or Orderers) the blockchain may have a different name designation and a different ”function”. Blockchain that resides on Orderer peers are named Orderer Ledger and blockchain that resides on peers are named Peerledger. But the blockchain are the same and have the same contents the difference is due to the state component.

- OrdererLedger, is maintained by orderers for fault-tolerance and availability.
- PeerLedger, mechanism to tell a part valid transactions from invalid transactions. Used to build and maintain the world state component.

## 5.10 Consensus

In order for a block to be created, a number of rules must be enforced, just like any other Blockchain technology, as presented in Section 3.8. Bitcoin relies on Proof of Work, where computational power dictates how consensus is achieved. In Peercoin or any other Proof of Stake network, stake determines how consensus is achieved. All of these networks have something in common which is being public. Hyperledger Fabric is a private blockchain and these methods may not be the most efficient way of achieving consensus. Proof of work and Proof of Stake requires a vast number of peers to be most efficient and secure, but require monetary investment and incentive to keep the security and stability.

Hyperledger Fabric is targeted at business networks, so Proof or Work or Proof of Stake may not be ideal due being targeted at public networks, and getting more secure as the network grows, among others details explained in Chapter 3. Consensus on the Hyperledger Fabric is achieved by the Ordering Service which works by creating new blocks with valid endorsed transactions to be submitted to the leading peers of the channels so these can distribute the blocks to other blockchain holding peers. Hyperledger Fabric as of version 1.1 supports three different consensus implementations.

- Kafka and ZooKeeper, presented in Hyperledger Fabric v1.0, allows crash fault tolerance. But not good against malicious actions.
- Solo, solo ordering, should only be used in development environment.
- Simplified BFT is the first ordering algorithm introduced. Allows the use of Byzantine Fault Tolerance.

The ordering service must provide delivery guarantees of all messages as well as consistency of messages. Additionally the orderer service may provide ordering service to multiple channels, not being limited to one. Figure 5.4, by *Hyperledger Fabric Official Documentation*, presents the consensus mechanism.

### Ordering of transactions

When a transaction has achieved the number of correct endorsements, the client applications must then send the transaction proposal to the ordering service in order to be committed to the application.

The process of ordering a transaction is presented below. This is done with the aid of specials smart contracts to enforce and secure:

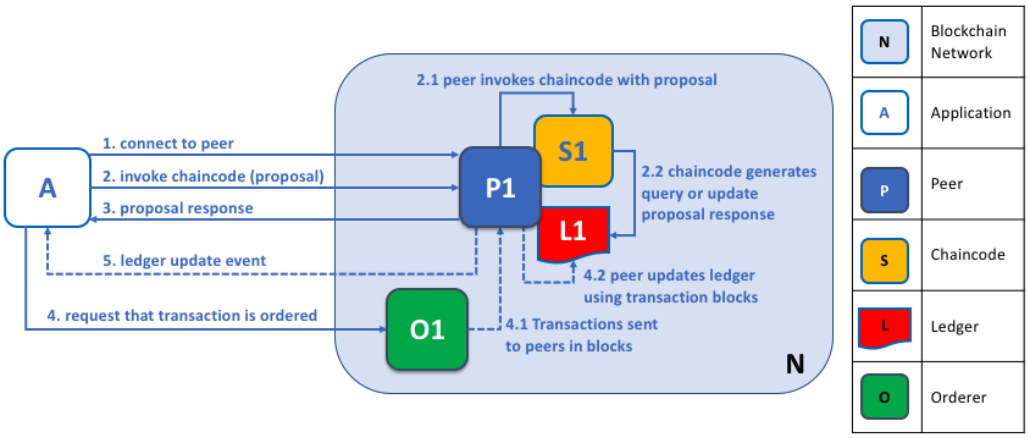



Figure 5.4: Transaction flow [Hyp18].

1. Orderer service receives transaction proposals;

2. Validates endorsements and verify if transactions meet the stated endorsemennt policy criteria;

3. Compare smart contract output from all endorsements. If equal, the transaction may succeed else, it fails;

4. Transaction is either flagged valid or invalid;

5. Bundle sufficient transactions into a block or wait for a timeout and then broadcast the newly created block to the leading peers of a channel;

### Orderer Peers and Transaction Data

Orderers do not inspect the transactions but only the endorsements and the necessary information to check its validity like the output data. Input parameters can be included in a transaction, but this field can be turned off to endorsers only although the output data must be visible in order for the orderer service to achieve consensus from all the endorsements response. Workarounds can be made to a certain extent, like hashing the data or encrypting it. Although in Hyperledger Fabric newer versions, the private data feature can bypass the previous condition.

## 5.11 Chaincode

*"A chaincode typically handles business logic agreed to by members of the network, so it may be considered as a "smart contract""* [Hyp18]. These programs are supported in one main programming language called Golang, other programming languages include peer.js and Java. There are two types of chaincode, system and user. Chaincode does not reside on the blockchain, system chaincode resides on the peers machine. User chaincode resides on an isolated Docker containers so not to be interfered associated with each peer that can instantiate it, but in order to deploy, upgrade, initialize a user chaincode, it generates transactions that are recorded in the blockchain, so these must be endorsed and ordered.

### 5.11.1 System Chaincode

System chaincode resides on peers and is deployed and register by the peer on start-up. These special chaincodes allow for privileged actions as it has more resources at disposal as opposed to the user chaincode. These chaincodes are used to evaluate transactions, achieve consensus and aid channel configuration. New system chaincodes can be added but follow a different procedure when compared to user chaincode because they reside on the peers machine instead in a isolated Docker container.

Pre-exisitng system chaincodes are:

- Lifecycle System Chaincode: handles the lifecycle of user chaincode. It packages, instantiates, installs and upgrades
- Configuration System Chaincode: handles channel configuration on the peer side [Hyp18]
- Query System Chaincode: query related interactions.

Changing existing system chaincode is not recommend by the Hyperledger Fabric developers and must proceed with extreme caution.

### 5.11.2 User Chaincode

User chaincode resides on isolated Docker containers in order not to be tampered or interfered with by any other peers. User chaincode can be implemented using multiple languages but mostly Golang and Node.js although there is support for Java, as of this date. All chaincodes have the `invoke` method in common, this is the first function to be called which will redirect to other functions called by the transaction proposal.

These contracts are called using a Software Developer Kit (SDK). Invocation by the SDK is exclusive to applications, as referenced in Section 5.12, and can perform multiple operations like read from and update the blockchain with new entries and update existing ones, among many others.

#### Install, Instantiate and Upgrade Cycle

Chaincode needs to be installed, instanteated and at some point in time upgrade because of unseen changes. This cycle only affects user chaincode, because system chaincode does not have the same behaviour or cycle. These chaincode operations only targeted at endorser peers.

- Install, install chaincode into the target peers. All peers within the channel must install the chaincode, in order to endorse transactions from it. Installing a chaincode is treated as a transaction therefore it needs to be endorsed and ordered.
- Instantiate, this enables the contract in the channel it resides on and starts the docker container enabling the simulation of transactions. It is treated as a transaction therefore needs to be endorsed and ordered
- Upgrade, replaces the old chaincode with a newer version to apply changes on the old chaincode. It is treated as a transaction therefore needs to be endorsed and ordered. New endorsement policy criteria may be specified, but in order for the transaction to succeed the current policy is what matters. [Hyp18]

#### Interacting with multiple Chaincodes

Since the state component is populated by key-value pairs, if a chaincode requires access to information, a simple fetch using the ”key” should suffice.

The previous statement will fetch the information stored in the state component or blockchain and if the information being fetched was previously entered in the blockchain by the chaincode that performed the write operation. This means, that if a chaincode requires access to information written by another chaincode, the return information will be an empty result set, but will not cause the transaction to fail. This is to not cause information leaks.

So the only way, at the moment, is to use special functions that call other smart contracts via the invoke method. When calling chaincode via another chaincode it will not require a new transaction, multiple chaincode calls happen in a single transaction.

This is not exclusive to read operations, the same goes for write operation. Only the chaincode can operate over the information written by that chaincode.

#### Random Numbers

Using random numbers is a common practice in software development. In this case, where the Orderers compare output results, using random numbers may lead to inconsistency. In the case of Hyperledger Fabric each endorser peer simulates the transaction outcome, and if the chaincode uses random numbers it means different endorser peers might have different results. This particular situation becomes a problem when the ordering service validates the received transaction proposal with the endorsements and notices, regarding this factor, that there is no consensus on results, therefore the transaction fails and does not update the world state.

Random numbers must have special attention because an initially correct transaction will most likely fail due to inconsistency of results.

## 5.12 Applications

Applications are programs built with the Hyperledger Fabric Client Software Developers Kit or, Fabric SDK for short. These are presented in a numerous languages but only Node.js has full support and documentation.

The purpose of the Fabric SDK is to provide easy to use APIs to interact with a Hyperledger Fabric blockchain via smart contracts, as presented in Figure 5.5, by *Hyperledger Fabric Official Documentation* . Applications are not unique, as there can be multiple applications, which can be seen in Figure 5.1, in a sense that an organizations creates their own applications to fit the needs of their services, essentially having multiple applications which are organization specific.

Applications are used by users which allows them to generate transaction proposals to be sent to endorser peers and orderer peers. Effectively reading from the blockchain or upgrade the blockchain.

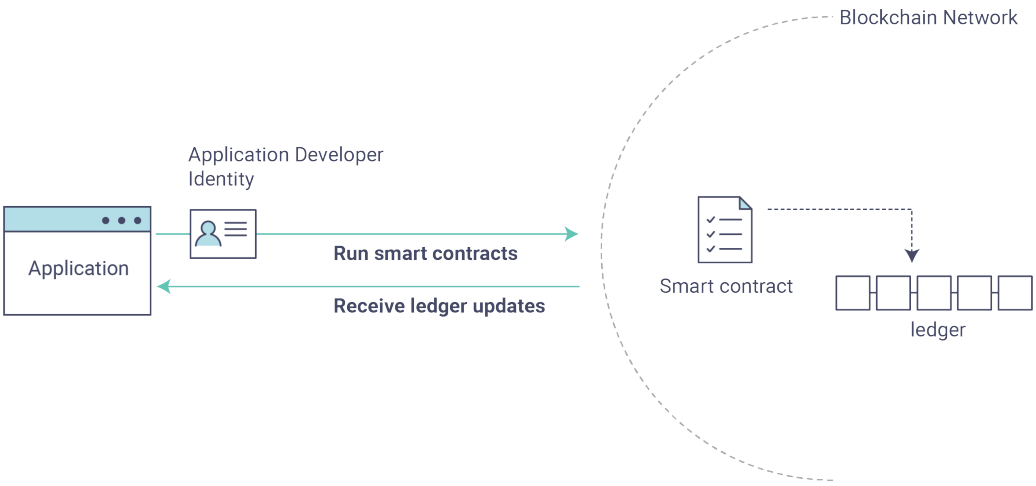



Figure 5.5: Application interacting with the Blockchain network. [Hyp18]

# 6

# Blockchain Data Sharing

*This chapter consists of the description of the components and decision process in the work done in this dissertation. Consists of architecture design process, description of the developed smart contracts and applications that interact with the Blockchain.*

## 6.1 Architecture

The architecture is focused on maximizing the properties of the blockchain technology. The concept behind the architecture is to have as much focus on blockchain and smart contracts as possible for the management and control of the data sharing mechanism. Focus on the implementation process was primarily on smart contracts because these describe the business logic of the data sharing mechanism. External applications that connect to the blockchain network are also important as they provide ways of interacting with the data sharing mechanism.

### 6.1.1 Data Storage

Data storage can take many forms, storage providers or hosting data on blocks of the blockchain. The concept of using blocks for data storage can pose few problems, the reason why is because of consistency, efficiency and security [LSZ15, SZ15]. Blocks by nature store only transactions, previous block reference, timestamp and additional important fields. Transactions are lightweight in size, this means a block can fit a large number of transactions, and the amount of transactions that reside in a block will also depend on the size of the block. Additionally the number of transactions allowed per block or timeout creates a block due to not receiving enough transactions.

Either hosting on blocks or using the blocks to store file data, leads to three approaches:

1. Unlimited block size
2. Maximum data share size on blocks
3. External storage provider approaches

#### Unlimited Block Size

This first approach is not recommended since blocks have a fixed size limit because of consistency, efficiency and security. The more blocks, the harder it is for a malicious user to disrupt the network, as explained in Section 3.8. Removing this factor enables blocks to have different sizes and destabilizes the growth of the blockchain as well its security.

Peers must have a a copy of the blockchain locally, if the blockchain is a dedicated decentralized repository machine, every peer holds a copy of all files of all participants, considering one time edits on files. If a file requires an update, this scales the size of the blockchain exponentially. Additionally, besides the file data, it also requires the transaction data. Another setback of this approach is the input data. This would have to be encrypted, otherwise, peers validating the transaction would be able to see contents of a file.

As an example, if the blockchain has 100TB of data, counting only file data in blocks, and every participants makes one small edit, the blockchain grows to 200TB.

#### Maximum Data Size

Data sharing directly on the block with fixed block size limits the amount of data and the kind of data able to be shared on the network. Bitcoin blocks size is around 1MB, this would limit the maximum size of data shared by 1MB. The blockchain would grow exponentially in the long term, because the number of files per block would be low, therefore the number of transactions would also be low. As an example, considering a network of 2 billion files, if every file had 1MB each that would mean two billions blocks not counting transaction data. In contrast Bitcoin does not have one million blocks as of this date since start [blo]. This approach holds the same problem as the previous, data would not be able to changed often because of the blockchain exponential growth, limiting the network for just very small and critical files with low probability of update. Additionally, input data is visible by certain peers, unless encrypted.

Stored files on the blockchain also poses an additional problem, even when considering small files with a very low probability of being modified. The file could never be deleted and a complete history of changes would be logged in the blockchain, which this last can be seen as positive. For sensitive or confidential data this would

not be the optimal approach, even by the properties of a permissioned and private blockchain network limiting what other users can visualize, In order to reach consensus input and output must be visible unless these are encrypted, in this case, the submitter of the transaction must be able to share the decryption keys. An alternative is to use private data functionality of Hyperledger Fabric.

#### External Storage Providers Connected to the Blockchain

This was the approached that was taken in the work described in this dissertation, that makes logical sense. This allows files to be registered as an asset in the blockchain network but not reveal its contents, providing enough information to trace it to the owner and allowing the storage provider to identify a given data based on requests validated by the blockchain network. Additionally it allows the use of smart contracts to control the flow of the data sharing mechanism.

This approach allows the blockchain to not grow out of proportions and blocks remain with a fixed size. Additionally, it allows for data to be deleted and use the blockchain to update the status of a given file and logs all events.

An obvious problem is that it requires users to have a certain level of trust regarding the storage provider if data is not encrypted before uploading, as explained in Chapter 2. Nonetheless each Organization of the Hyperledger Fabric network can have its own trusted storage providers.

### 6.1.2 Blockchain Decision Process

This section is focused on the decision process regarding the choice between implementation of a blockchain network or using an existing blockchain implementation. Based on the previous decision, choosing between a public or a private blockchain is a fundamental decision. The selection of the blockchain technology is clear, because it was mentioned several time in Chapter 4 and 5, which is Hyperledger Fabric.

#### Implementation vs Using a Blockchain Implementation

Developing a project based on blockchain technology leads to the question of implementing a blockchain from scratch, or use open source blockchain like Bitcoin or Ethereum as a basis for implementation, or use a popular blockchain framework like Hyperledger Fabric.

1. Implementation of a Blockchain Network from scratch.
2. Existing Blockchain implementation with turing or nearly turing complete smart contracts functionality.

The first approach is the more flexible approach, which allows the development of a blockchain network focused entirely around data sharing. This would mean a dedicated blockchain optimized for a single objective. The main obstacle with this approach would be the complexity of developing a fully working decentralized blockchain network with appropriate scripts for data sharing.

The second approach is to use a blockchain implementation that has smart contract capability with a turing complete or near turing complete language support. This would allow to use the blockchain and create business logic that would create the data sharing mechanism without implementing a full blockchain decentralized network from scratch. It is similar as to the previous approach, without being fully optimized for a single task

but would be considered as feasible proof of concept or even production approach of a secure data sharing mechanism.

In the context of this project, the second approach is the most obvious due to the nature of developing a proof of concept with less costs and a more realistic approach.

### Public Blockchain or Private Blockchain

Public or private blockchain was a fundamental decision in the development of this project, this would mean choosing a blockchain implementation like Ethereum or a blockchain framework implementation Hyperledger Fabric. From Chapter 1 to Chapter 5, the decision is clear on which approach was taken. But a fundamental aspect was to make a choice that could make the proof of concept closer to an actual reality. This means that the concept of blockchain framework or network would be around smart contracts, which means having the tools to build these with a turing or nearly turing complete programming language.

Using a large public blockchain, such as the Ethereum Project, has some advantages. The main advantage is a large community, strong and stable blockchain network and a consensus mechanism that ensures security, stability and consistency. But it holds a setback, transactions are made to be public and tend to favor anonymity instead of identity, this in order to increase security and confidentiality in the network. An additional point is the consensus mechanism. Public blockchains do not tend go by trust so cryptographic proof is the approach, usually a Proof of Work and Proof of Stake variations is common. This requires monetary investments on hardware especially in Proof of Work because of its nature. In order to execute smart contracts payment is required for the *miners* to execute the smart contract. Even without the needed hardware costs, computational costs are required. So setting up a sharing mechanism on these platforms would imply additional costs for organizations.

When focusing on business networks, involving multiple businesses this would not be a correct approach because of the infrastructure cost and maintenance associated with these public consensus mechanisms, and would require a great number of peers to keep it secure and tamper proof. Additionally, these networks tend to go by anonymity and being public permissionless, means anyone can join. This would be difficult or impossible to keep track of data sharing, unless some sort of identification, external to the network is used. Alternatively, the public network can be a permissioned one, but this still allows outside users of the network to access transactions, which would not be a desirable in terms of privacy.

Private blockchains, on the other hand, the situation is different. The consensus of Hyperledger Fabric does not require heavy computational resources, but requires some trustworthy peers at least regarding the Ordering Service peers. Hyperledger Fabric provides identity over anonymity, so participants are identified as well as allowing restriction over who can join, who can add content to the blockchain, among others, who can see what data, thus being permissioned. Transaction can be traced to the identity of a user and by inspecting the blockchain the owner of the data can easily identify who interacted with the data and it must have been from a user registered within the network. This guarantees only know and registered users can interact with the network.

An additional positive factor when choosing Hyperledger Fabric is that it could be deployed anywhere to serve the needs of the situation that requires it. There is no need for a single global network, but it allows the existence of several independent networks, serving different needs.

### 6.1.3 Role of Smart Contracts

Business logic has to describe data sharing solely based on the blockchain. This is where smart contracts are required and this is the reason why a turing complete or nearly complete is of most importance as it allows to develop the mechanism with a greater flexibility and little restrictions. Data sharing includes the following points that must be addressed.

- Who owns the data.
- Who has access to the data.
- When was data accessed.
- For how long can the data be accessed.
- How can data access be controlled.
- How to stop users from accessing the data.

#### First smart contract - File Management

In order to provide data sharing mechanism solely based on the blockchain, a file is to be considered an asset. From this decision, a smart contract to control and manage the asset is required.

The smart contract needs the necessary information to trace it to the owner but not reveal its actual contents, so, information like unique file identifier, publish date of file, file name, identity of owner, hash of contents and a list of users who have access was considered enough to trace it to the owner and be able for a repository to identity.

The process behind this smart contract is that the blockchain is tamper proof and incorruptible therefore, by using these assets the user knows all available files to him at all times, the client application would retrieve the information only from the blockchain and not ask the storage provider, meaning omitted data from a repository if remotely stored. If the user wants to fetch data from a repository then the hash regarding that file would be compared with the hash of the retrieved file in order to tell if the data was tampered with. Assuming the actual data has been uploaded to the repository using the unique file identifier, then when the user only needs to query the blockchain to determine what data access he currently has, there is no need to interact with the storage provider for this information.

#### Second smart contract - Share Agreements

The previous smart contract enables to trace data to the owner but it does not provide a secure enough mechanism for data sharing with other users. Another smart contract was necessary to control the access of data, handle and control the sharing of data between two entities. This would control who can access the data, for how long data can be accessed, who owns the data, and the status of the agreement and stopping a user from accessing.

**Third smart contract - Access Requests**

The two previous smart contracts are not enough to control the access of data and does not provide a log of data access besides the Share Agreements smart contract, as this only log at the time of creation and when status is updated. So a another smart contract was necessary. In the same sense that the interaction between user and repository to be as limited and provide as little details as possible without interacting with the blockchain doing most of the work.

This smart contract controls the access requests between users and repositories, and the blockchain would automatically manage and grant or deny data access by validating it without interacting with the actual repository. And it also log all access requests.

The objective of this smart contract, beside what has been stated, is to provides as little information as possible on request creation and deliver the file(s) in question with as minimal interaction as possible between repositories and users. The solution is via an Unique Identifier generated at time of a valid request by the smart contract. Since the repository is not able to determine what that unique identifier means, the repository must use the blockchain and returns the corresponding data to the user. This in the case of valid request.

### 6.1.4 External Applications

Applications were based on tutorial applications of Hyperledger Fabric. They are basic applications that provide interaction with smart contracts. There are two applications: client and repository. The Client applications provide interaction with the blockchain and are able to make and receive requests from repositories.

Repository applications, were made to be also simple, simply interacting with the blockchain upon receiving requests from clients and provide the most simple data storage and interaction between users. Additionally, besides confirming and validating unique identifiers they, also invoke Access Requests to register in the network.

Interaction with repository applications and client applications is by means of a unique identifier.

### 6.1.5 Storage Provider Peer restrictions

A storage provider as a peer, is a possibility but depends on the situation, as mentioned previously, peers, if possible, are able to endorse and order transactions. In the case of the development process, the repository is not a peer, but an application that connects to a peer. This is not unintentional due to simplicity of testing purposes. Although having a repository as a peer means it is part of the network, this is acceptable, but for security reasons it should not be an endorser peer or an orderer peer. The reason for this decision is to increase privacy and transparency so these cannot accept, deny endorsements or censor transactions from being in blocks, or even inspect input and output. The assumptions is that a repository may be considered untrusted by users and providing validating capabilities can imply additional risks to the users, for example censorship of transactions or unable to achieve consensus.

If a storage provider is to be considered untrusted at some level, then the data security must be taken special care, either by storage providers ensuring data security and privacy and/or users encrypting their data before uploading. In Section 6.5, a simple approach using Gnu Privacy Guard for encryption of data and sharing of the same will be described.

### 6.1.6 Encountered Problems and Solutions

Multiple problems were encountered in the development of this work, the most problematic in terms of implementation are presented in this section.

#### Random Numbers

Random numbers can be problematic in terms of consensus, this means will most likely fail due to inconsistency of results. This made relying on random numbers a more careful approach. Initially the unique identifier used for the interaction of clients and repositories was based on random numbers but was changed to a SHA256 because the same input will always lead to the same output. The input was chosen as the timestamp, file identifier, user public signature and operation. The larger the size of the hash the more likely a brute force attack will fail.

#### Uploads

Uploading files to a storage provider presented to be a difficult problem. Access requests must be confirmed by the repository and created by the user. This means the user can send data to repository and repository does not confirm the access request or confirms but does not send. Or another case of the user creating an access request but never uploading. For this issue, an assumption was made and was to consider the user as non malicious in terms of upload. The reason for this assumption is because the network is private and permissioned, meaning all users are registered and identified, and a user would not register himself in the network if not for secure data sharing. Additionally a third optional confirmation was implemented. When the user receives the data he has the option to confirm he received it, nonetheless if a confirmation is not provided after a certain period of time it is considered confirmed.

#### Validity based on Timestamp

Hyperledger Fabric chaincode does not have a clock in the chaincode so it cannot auto validate itself without being called by external entities, so, instead of discarding the idea of timestamp validation the focus was changed to compare timestamps at time of chaincode calls.

The idea revolved around the concept of validation by chaincode calls. If the transaction proposal is either a query or write then an attempt to write is made to validate all active Share Agreements, this because chaincode cannot tell the difference between query/write. The operation type does not matter because the return values will be correct, but if it is a query transaction then the contents of the blockchain will not be updated.

### 6.1.7 Structure

The architecture is composed by multiple layers, each representing a level of depth. There are four layers where the highest layer is composed of two components. Figure 6.1 represents all layers from highest level to lowest level.

The highest layer possible to the applications layers. These interaction is between a user of the network and a storage provider connected to the network via unique identifiers generated with the aid of the Hyperledger Fabric SDK layer. The storage provider maintains the files hosted by other participants and determine whether

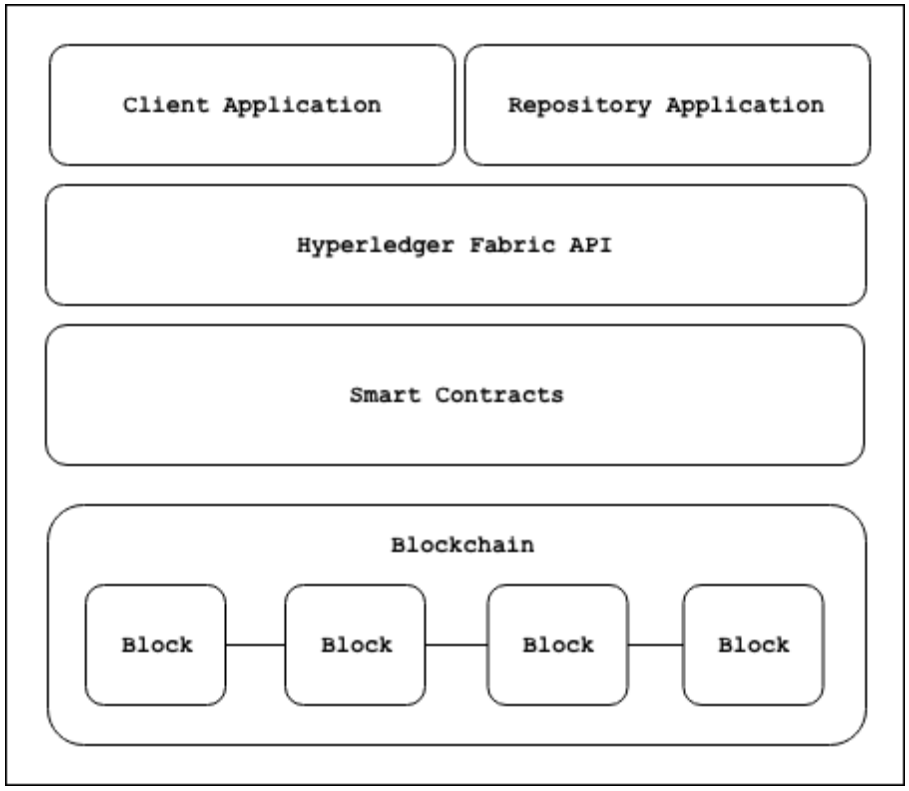



Figure 6.1: Diagram of the Layers of the Architecture

the request sent by a client application should be fulfilled or not. Whether is fulfilled or not depends on the Hyperledger Fabric SDK layer.

Hyperledger Fabric SDK level is the layer enabling the previously mentioned applications to communicate with the blockchain network. This encapsulates enrollment, registry of user and the ability to perform transactions proposals, in the sense of executing smart contract functions.

The Smart Contract level is where the smart contracts are deployed in the blockchain network which describes the business logic of the data sharing mechanism. During the development of this work and as mentioned previously, three smart contracts were created in order to devise a secure and controlled sharing mechanism.

- File Management: handles registry of files made by owners as file assets to be made available for sharing or just stored in the network.
- Share Agreement: creates, validates and manages the share agreements between owner of file and entity.
- Access Requests: creates, validates and manages all outgoing requests to the repositories.

The bottom layer is the Blockchain layer and together with the Smart Contract layer, compose the Blockchain network. This last layer corresponds to the immutable shared ledger containing all the information regarding the data written by smart contracts excluding information in Genesis block regarding channel configurations.

The architecture is described in Figure 6.2 which describes the data sharing flow over the blockchain.

In Figure 6.2 it can be seen the division between applications and the blockchain network. Applications submit transaction proposals against smart contracts, and blockchain network processes them and returns results.

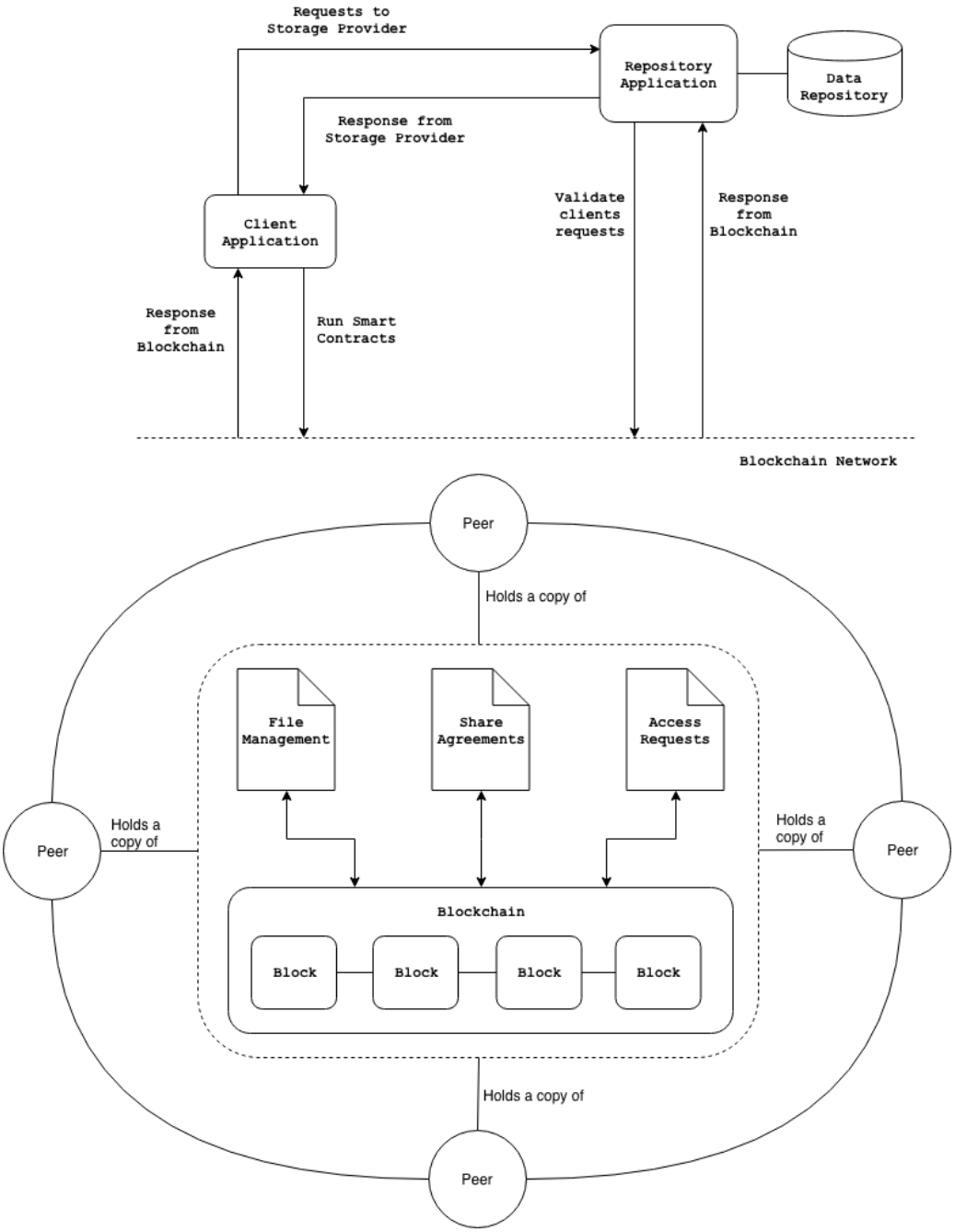



Figure 6.2: Diagram of the Architecture of the Network

Figure 6.1 represents a top level view of the network without regarding endorsement and orderer steps and using a single channel and presents just the interaction between applications, smart contracts and blockchain. Applications are external to the network and communicate with the blockchain network only by invoking smart contracts. Smart contracts query and update the blockchain based on input provided by the requests made by client/repository applications. The repository applications only use the blockchain network when client applications interact with it, except when registering, explained in Section 6.3.4.

## 6.2 Data Sharing Flow over the Blockchain

From uploading data to a storage provider and a second entity download the data there are several steps to follow, excluding any endorsements and orderer steps. Follows the details of how the data sharing procedure takes place.

1. Participant 1 registers and enrolls in the network and receives identity;
2. Participant 1 registers the file as an asset using `File Management` smart contract and creates an request for upload via Access Requests smart contract, pending confirmation from the storage provider;
3. An unique identifier is returned which is used to generate a request for uploading to the storage provider;
4. Participant 1 uploads the data along with the unique identifier.
5. Storage provider receives data and checks the unique identifier and signs the request as `ACCEPTED` or `ACCESS_DENIED`;
6. Participant 2 registers and enrolls in the network and receives identity;
7. Participant 1 creates a share agreement using `Share Agreement` smart contract with Participant 2 specifying that Participant 2 has a maximum of 24 hours to access the file;
8. Participant 2 makes an access request using `Access Request` smart contract and returns an unique identifier;
9. Participant 2 makes a request to the URL of repository with the unique identifier as a parameter;
10. Storage provider checks blockchain for a request that generated the received unique identifier and the smart contract handles the process of validation;
11. Storage Provider sends the file to Participant 2 and signs the requests;

From the previous steps, there are a total of 5 mandatory transactions and 1 optional executed from user chaincode. These transactions are not necessarily bundled into the same block.

- **Transaction 1:** File registered as an asset by Participant 1 by invoking File Management.
- **Transaction 2:** Participant 1 invokes Access Request for upload request. If not made in transaction 1 by a function that creates both in a single transaction.
- **Transaction 3:** Creation of Share Agreement by Participant 1 and 2 by invoking Share Agreements.
- **Transaction 4:** Participant 2 invokes Access Request for upload request.
- **Transaction 5:** Signing of access request by repository.
- **[Optional] Transaction 6:** Signing of access request by Participant 2 to confirm data was retrieved.

This is a simple case of just enabling the data sharing flow over the blockchain. This scenario expects Participant 2 to be within the limit stated in the share agreement made with Participant 1. Test cases are described in Chapter 7.

## 6.3 Smart Contracts and Business Logic

Following the architecture design process of the three smart contracts, this section is towards an accurate description and behaviour of the functionalities provided by each one of the smart contracts.

Smart contracts, or chaincode in Hyperledger Fabric terms, allows the manipulation of data stored in the blockchain, which operations are accessible via the Fabric SDK, used by applications. For the proof of concept, three different smart contracts were developed, as previously described. A smart contract may depend on other smart contracts, so internal calls in a single transaction occurs.

Business logic represents the data sharing mechanism managed by the smart contracts explained as Section 6.2. The smart contracts perform the following business logic:

- File Management, registers a file as an asset. Controls the management of all files owned by a participant hosted in a repository or participant's local machine.
- Share Agreements, controls the management of share agreements between the owner of the data and entities.
- Access Requests, management of access control of data between participant and repository. Purpose of this smart contract is the generation of unique identifiers and acceptance by the repositories.

### 6.3.1 Important Packages

For the development of the smart contracts, there are two packages, `cid` and `shim`, that are very useful and out of the two, the `shim` package is mandatory.

- `shim`. "Package shim provides APIs for the chaincode to access its state variables, transaction context and call other chaincodes" [shi]
- `cid`. Client identity, permits the manipulation of the invoker's signature, which contains organization, unique name, attribute and additional properties. [cid]

### 6.3.2 File Management

This smart contract written in Go, between the three smart contracts, the `file management` smart contract is the first the user should interact with. Following Section 6.2, in order to enable sharing of data in the network a user must invoke this smart contract first in order to register his data as an asset so the hosting and/or sharing of data can be set up in a secure way. The information regarding the data is not the data itself, as explained in Section 6.1. As such, it contains the information that can trace the data to the owner if hosted in a remote storage provider in the network. It also allows for querying information regarding data, such as history of changes in the fields presented below or which data a user has access to. By logic, when the user registers a file as an asset, it works as a shared ledger of all the data in all repositories. This in the case the user has data spanned across multiple storage providers connected to the network. Based on what has been stated, this smart contract contains the following information to be stored in the ledger.

- `File Identifier`, this value is the key parameter in the key/value pair. It is used to uniquely identify the contents in the blockchain, which links the data to the owner, and allows the management of data. Users can only change information if the unique identifier corresponds to the current owner.

- `File Name`, the name of the file provided by the owner.
- `Location of File`. Repository that holds the data.
- `Owner Identifier`, self explanatory, is an unique ID that identifies the user. In order to perform any operation, the user must be a registered user within the Certificate Authority.
- `User Permissions List`. A list of users with access permissions. Contains the user IDs of every user that can have access to this data. The list updates itself depending on the Share Agreements. If a Share Agreement expires or is terminated by the owner, it will modify the list. This field is necessary for Access Requests smart contract to determine if its the owner making a request. This information only visible to owner. It holds no purpose to show users a list of users with access permission.
- `Timestamp`, self explanatory, when was the file registered or modified in a readable format.
- `Status`, the user may disable access to this file at any given time. Status can hold two values "Available", "Unavailable". It will not terminate any active Share Agreements with the `FILE ID`, but no access will be given.
- `SHA512`, a SHA512 of the file. Useful when hosted in a remote repository. Used for comparisons at time of upload and download and share between users. SHA512 to ensure no likely collisions will happen.

No smart contract is complete without a set of functions that allows the manipulation of the data stored in the blockchain. All smart contracts in Hyperledger Fabric have an `invoke()` method which contains all functions publicly available for applications to use, as such, a wide variety of functions is required. Every data asset is firstly registered in the blockchain as a means to be able to control and keep record of all share and access history regarding that file.

Having a data asset enables to have no interaction with hosted repositories in order to visualize existing data and to provide a shared ledger between multiple repositories. This allows users to use the blockchain to keep control of all files scattered among multiple or single repositories.

Changes that happens in the file regarding the information stated above, either a file registration, an update in permissions, a status update, or a file delete will trigger a `timestamp` update. This only happens if the user has the permissions to do so: it is either an owner or is able to edit the file. This permits the owner to have a complete history of changes on the data asset across single or multiple repositories with little hassle.

This smart contracts allows users to check which files he owns, which files he has given share permissions on.

Not all functions will result in an update of the blockchain, users may just require the need for simple query. An example is querying all the data he owns.

### 6.3.3 Share Agreements

Share Agreements are written in Go and describes in code how data sharing interaction between two entities will occur against a file. This is the most important smart contract of the three previously stated as it ensures and enforces access control of data. This smart contracts creates, validates and maintains the state of all active share agreements between two entities. A share agreement between two entities can only be made if the user submitting the transaction proposal is the owner of the data, this means an internal call to the previous smart contract in order to determine if this condition is met. A contract contains the conditions of how a file is meant to be shared and with who. As such, it also writes to the blockchain a specific set of data in order translate the agreement into code.

Additionally only one active share agreement of `FILE ID` per user.

- `Share Identifier`, this is the primary key. Same as the `File ID`, used to uniquely identity a state in the blockchain.
- `Owner of File`, or creator of contract. This information is to help identity who owns the contracts without calling the previous contract to find who owns the data identified by the unique identifier.
- `File Identifier`, corresponds to the file being shared.
- `List of Users`, corresponds to the second party of the share agreement.
- `Conditions`, corresponds to the conditions of which determine the validity of the contract. i.e: User has access for 24 hours.
- `Timestamp`, corresponds to the exact moment when the contract was created or updated.
- `Terminated`, corresponds to the status of a contract. By the name it is of boolean type.
- `Can Modify?`, this corresponds to being able to modify the data. If this field is true then it will need to update data identified by `FILE ID`

Due to the nature of identity and smart contracts double checking identity, a malicious user cannot grant himself access to data owned by another user. This unless the owner of the data leaks the credentials.

Smart agreements cannot automatically validate themselves, as mentioned in Section 6.1.6, which is solved by validating share agreements by invocation.

### Validating a share agreement "automatically"

Validating share agreements is achieved by using the properties of the blockchain. Updates on the blockchain only occurs if the requests from the application are a transaction proposal that updates the blockchain.

Since chaincode cannot tell if a transaction is a write or read operation, the solution is to assume every request from applications are transaction proposals that update the blockchain. As such every `invoke()` method of every smart contract, not limited to the Share Agreements calls the function to validate every valid contract in the blockchain. This way, on every request from applications, an attempt to validate the agreements is made.

Figure 6.3 represents the process for every chaincode call, it first begins by attempting to validate the contracts and then proceeds to the actual chaincode call. If it happens to be a transaction that updates the blockchain then the agreements get updated. This will lead to two obvious questions:

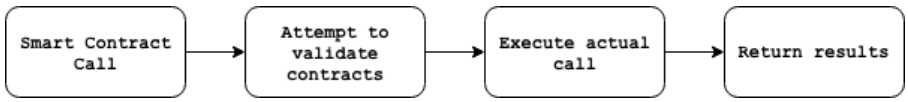



Figure 6.3: "Automatic" Validation Steps

- Question 1: Why not just check contracts if they are valid and not write to blockchain? Because that would imply scanning every single contract from the genesis block with the user id and recheck the validity, which although is an alternative, the previous implemented method is not that heavy, it performs less operations then the alternative method because it only needs to scan active contracts via `Terminated` field and only performs write operations when necessary.

- **Question 2: If there is not a write operation then how can the network know if the user still has access?** Because although it is assumed that every application request is a transaction proposal the smart contract only uses this assumption to update the status of the agreement on the blockchain, but can still say the contract is not valid because, if the `Terminated` returns false, a validity check must happen. The difference is it will not update the contents, but the return values will be correct because of the validity check.

Alternatively, instead of validating every Share Agreement, validation of only relevant Share Agreements to the user could reduce any possible block creation conflicts because of inconsistent states created by transactions.

#### Terminating a Share Agreement

Contracts are made to be able to share and control share permissions between two entities, and once created can be terminated at any given time but only by the creator of the contract, by the `Conditions` field not being met, or due to violations on the contract, for example attempting to write with only read access. If a contract is terminated, all access to the data is unauthorized, and once terminated, the `Terminated` field cannot be change, requiring a new share agreement.

Terminating a contract also updates the `users` list in fields of the data asset identifier by `FILE ID` of the File Management smart contract.

#### More Functionalities

This smart contract is not only to maintaining and validating share agreements. Queries in this contract are obvious, allowing to check available contracts, history of changes, query contracts by `FILE ID` and all contracts associated with the same `FILE ID`, terminating contracts at any time, querying by owner and also importantly validate contracts.

### 6.3.4 Access Requests

The Access Request smart contract is written in Go, is the last smart contract in the trio and is how users interact with the data spanned across multiple storage providers . It describes the logic of requesting access to data that is hosted on repositories. This means all requests of accessing a file must be channeled through the blockchain so it can be validated. As mentioned previously, it is the only way to retrieve a given file.

A request must contain enough information to determine who made the request, what data is the user requesting and when was the request made and the legitimacy. Requests created by Access Requests needs two validations, three optionally, to be considered valid, which translate into two or three transactions. The first transaction is by the client application, the second one is by the repository application to confirm the request, and the third is by client application to confirm the repository delivered the data. This smart contract requires interaction between all layers described in Figure 6.1. The information that is stored in the blockchain is the following:

- `Access Request Identifier`, same as `File ID` and `Share ID`. Used to uniquely identify the access request in the blockchain.
- `User Identifier`, identity of the user that made the request.

- `File Identifier`, file identified by `File ID` that user will want to access.
- `Timestamp`, timestamp of when the request was made.
- `Status`, status of the request. Values: `ACCEPTED` or `DENIED`.
- `Repository Identifier`, identity of the storage provider that checked this request and gave it a second confirmation.
- `Unique Identifier`, an unique identifier generated the smart contract when the request is made, valid only once, in order to download data.
- `Unique Identifier SHA256`, used to compare the input based on the unique identifier received on each `PENDING` request.
- `When Validated by Repository?` Timestamp but for repository validation of request.
- `Feedback on Receive`, only writable if the two previous fields have been updated by the Storage Provider. This is an optional third confirmation of the access request. Starts `null` but can be changed to two values. `DATA_RECEIVED` or `DATA_NOT_RECEIVED`. If field is not modified it is assumed delivered.
- `Has UID been used?`, filter out used `UID`.
- `Operation`, do determine what is the operation being executed.
- `Is Owner?`, determine if it is the owner of a user with permissions.
- `HTTP Request Method`, to determine if the `Operation` matched the correct HTTP Request Method. To be completed by second confirmation.

At time of file asset registration, the user should be aware of the storage provider in order to fill the information, as stated in Section 6.3.2

The information regarding the repositories is the following and is maintained by the Access Requests Smart Contract.

- `Repository Name`, the name of the repository.
- `Repository Identifier`, unique repository ID, achieved from the certificate. Same as the user ID.
- `Timestamp`, how long has this repository been around since registration.
- `Access`, how to access this repository. In the test environment this field is a `localhost:port`.
- `Is Repository?`, for query purposes, a filter for active repositories.

This smart contract is of great importance since it is the link between the user and the repository when fetching data. As mentioned, access requests are divided into two parts, and a third optional one. Figure 6.4 summarizes the two validation process of the generation unique identifiers, `UID`, for a request for any operation, either get or update or delete, to be submitted to a repository.

In a descriptive way, the all steps for access request validations found in Figure 6.4 are:

1. User makes a transaction proposal for a request.

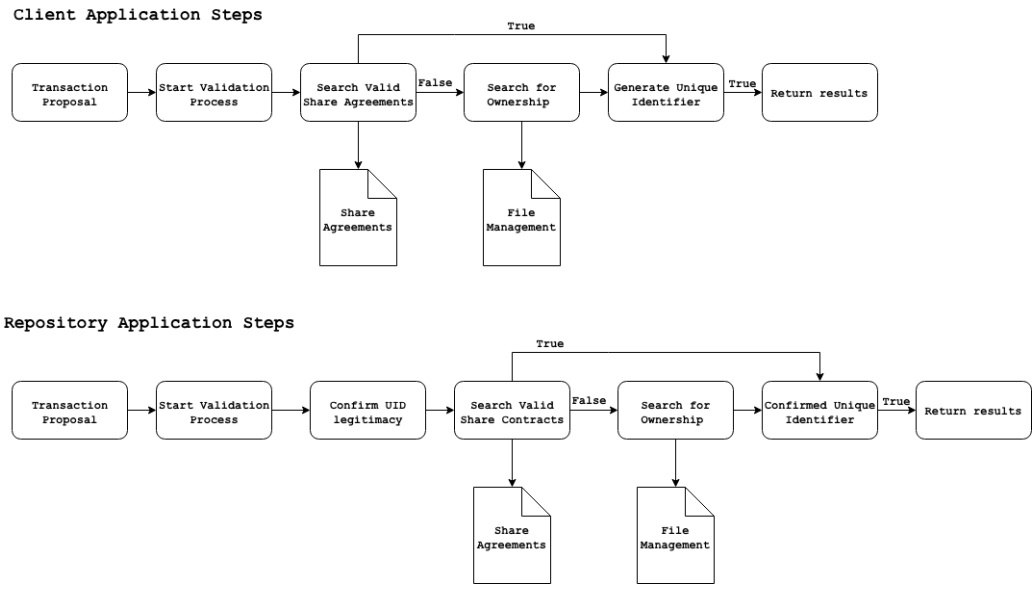



Figure 6.4: Access Requests steps

2. Access Requests checks for any valid Share Agreements, by invoking `Share Agreements`, for the given `FILE ID` in which the user is in the list of permitted users. If there are none, then it checks for ownership of the file by invoking `File Management` if the user is the owner of `FILE ID`.

3. If the request is valid, returns an `UID` and the status of the request is flagged `PENDING` else it gets flagged `ACCESS_DENIED`. If it is flagged `ACCESS_DENIED` the Repository Application Steps are not required.

4. User sends `UID` as a parameter for the repository URL.

5. Repository receives `UID` and checks the legitimacy of the `UID` by checking if it exists, has not been used and has not expired and the operation matches the correct HTTP request method.

6. If any of the conditions fail then the request is flagged as `ACCESS_DENIED`. And the process stops here.

7. If all conditions are true, then, in the same transaction proposal, it makes another invocation to Share Agreements to determine if the Share Agreement is still valid and who made the access request. If there is no Share Agreements and if the `UID` is valid by the business logic then it determines it was the owner of the data who did it.

8. If it is not the owner but a user with permissions and the contract is no longer valid then the request is flagged `ACCESS_DENIED`.

9. If validity check turns out to be valid, then an update on the request is flagged with `ACCEPTED` and request is signed by the repository.

10. Transaction proposal returns a set of values to determine the data the repository is going to send back to the user.

11. Repository sends the data back to the user.

12. Optionally User may sign again the request to confirm receive of data.

The formula for `Unique Identifiers`:

$$UID = SHA_{256}(timestamp_{raw} + file_{id} + invoker_{id} + operation_{id}) \tag{6.1}$$

The formula is composed of four elements wrapped in a SHA256.

Equation 6.1 is a the formula for generating an UID. In order to be easily verified that the input generated the hash, keeping it consistent. A hash of 256 characters long permits, along with the timeout given to the unique identifier permits brute force attacks to be much more difficult to achieve. It is then encoded into base64 for URL simplicity.

The first element in Equation 6.1 is the timestamp. For a timestamp based identifier, but in order to make consistent results due to the decentralized nature, nanosecond and seconds were removed from the equation due to possible inconsistency of results in endorsements.

The second element in Equation 6.1 is the `FILE ID`. The previous element is not considered enough because there could be multiple access requests in the same minute, so this field was added to avoid collisions.

The third element in Equation 6.1 is the `user ID`. Every user has an unique id, and there are no equal ids in the network, according to Membership Service Providers. The previous two elements are not considered sufficient to avoid collisions because of the one minute difference, because there could be same file id requests at in that time limit, the invoker user id was added to reduce the number of possible collisions. An obvious disadvantage is the user can only perform access requests once every 60 seconds.

The fourth element in Equation 6.1 is the operation. Operation ID holds three values `UPLOAD`, `DELETE`, `GET`. Since with the previous three elements a request can only be made every 60 seconds, using this element adds flexibility because permits different type of operations in 60 seconds instead of one. This element also allows the `UID` to identify the operation being executed, use it to compare with Share Agreements, as in, can the user modify the data or not. The owner of the file can easily verify if the correct operation was really conducted by doing the SHA256 of the input, although the type of operation is also logged.

### Additional UID confirmations

From the current setup, there is no way to tell if the `UID` sent by URL parameter is a malicious action, accidental action or legitimate action. Because of this uncertainty, it is treated as a possible malicious action therefore, once the storage provider application receives the `UID` as parameter, it makes a transaction proposal to update the access request using `UID` and the HTTP request method, either GET, POST or DELETE. Figure 6.4 sums the following steps for the repository application.

A second `UID` confirmation is deemed necessary to enforce that a repository will process the request. Figure 6.5 represents a flow chart representing all the steps and decision process for a `UID` confirmation.

A third confirmation is optional by the user, mostly useful to confirm data was downloaded or not. When the operation is finished, a transaction proposal can be made to the blockchain to update the access request in order to confirm if the operation was conducted or not by the repository. This to ensure a storage provider signs the access request and does not deny for whatever reason. Because there can be other reasons for the request to fail, such as lost of internet access, it was deemed optional and therefore after a certain period of time the signing of the request becomes irrelevant.

A timeout is necessary and the reason is because although only the owner, the user that submitted the requests and the repositories see this `UID`, not taking into consideration orderer or endorser peers, it is not wise to have

a non expiring `UID`, this because of possible human error.

An example why the timeout was deemed necessary: the user is not malicious, gets a valid `UID` and the second confirmation will make it valid as well. The good user loses internet access for a brief period of time and forgets about the request he made. This leaves a valid `UID` waiting to be used for as long as the channel exists or until the user remembers. In the meantime, a malicious user in the meantime gets hold of the `UID` and sends `UID` via parameter to the storage provider and gets access to the data. Unless the user can get hold of the `UID` and make a request within the timeout time limit the malicious user will never get hold of the file.

#### Automatic Revocation of Access

Automatic revocation is based on the type of operation and the permissions the user has. If the user does not have the right permissions the share agreement is terminated. But it must be considered that files have different levels of sensitivity. As such, any user with the incorrect permissions trying to access data with high sensitivity should be blocked from interacting. Additionally, in the same case, a user with read permissions attempting a write permission should lose all access permissions. Essentially terminating the share agreements.

An HTTP Request Method is necessary to ensure the correct operation that a user submitted matches the correct HTTP Request method received. This is to avoid a user submitting a FETCH operation but sending a POST request, which may lead to updated data which he may have not had access to. If these two fields don't match, then a search on the share agreements is firstly made and if found the agreement is terminated then the request is flagged as `ACCESS_DENIED`.

## 6.4 Applications via Blockchain APIs

Applications are programs that use the Hyperledger Fabric Software Developers Kit to use blockchains APIs to interact with the network. Applications are used by participants of the network, which are registered within the Membership Service Provider of that organization. As referenced in Figure 5.1 and Figure **??**, each organization has its own applications. In the work a single organization was used thus providing two different applications. Applications act as wallets as they store the credentials of the user used to sign transactions.

### 6.4.1 Client Application

The client application acts as a wallet, in the sense that it stores credentials and has the ability to access the blockchain via those credentials. By holding the credentials, it gives access to the data stored in the blockchain which allow for remote storage access. And as long as the participant does not loose access to his credentials, then data will never be lost, as long as the storage provider persists.

Additionally, besides blockchain level interaction, it provides interaction with the repository hosting the data, via three http requests methods. In the test environment this is done outside the application.

- `POST`, upload data to the repository. Data should be registered as an asset in the blockchain in order to increase overall security.
- `DELETE`, delete data from the repository. Data must have been uploaded first and unique link identifier must match one in the blockchain. If data does not match or does not exist no valid link is created. So, malicious or careless actions are flagged as invalid.

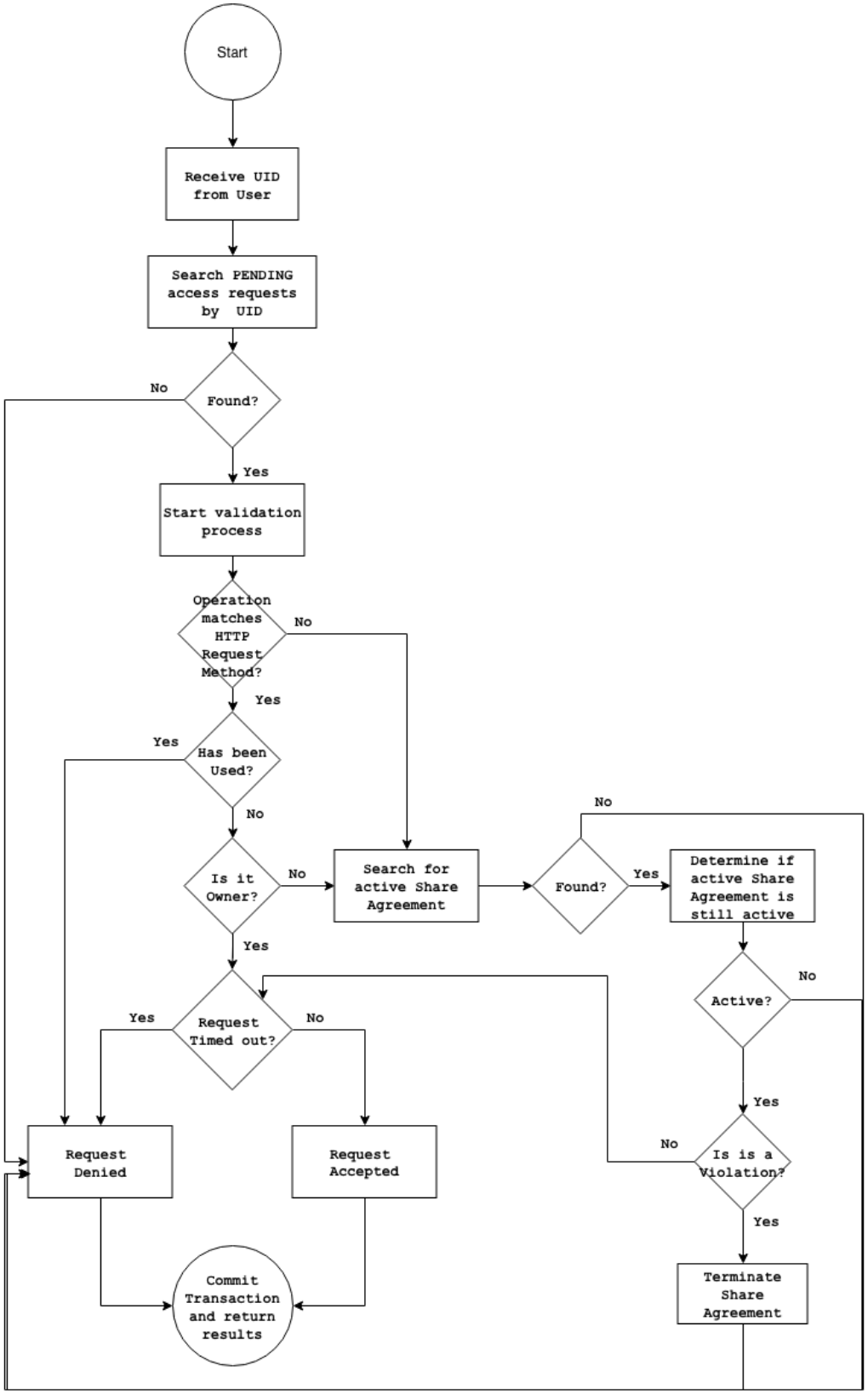



Figure 6.5: Decision process for `UID` confirmation

- `GET`, get data from the repository. Data must have been registered and unique identifier must match one in the blockchain. But if data does not match or does not exist, the requests are are flagged as invalid.

### 6.4.2 Repository Application

The repository application is used by storage providers, and like the client application it acts as a wallet, storing credentials and allowing access to the blockchain via those same credentials. In the current implementation there is nothing that differentiates a participant from an actual storage provider at certificate level. Only a list kept by the `Access Requests` smart contract stating who is registered as a storage provider and who is not. Adding storage provider specific attributes when registering using the Membership Service Providers is an alternative to clearly differentiate, removing the need for previous list.

The repository application is always on hold for requests and waits for requests at two different routes. The repository at the highest level of the architecture does the following steps

- `/upload/`, receive data from participants. This takes shape of a `POST` request and the fields are `UID` and the data.
- `/delete/{UID}`, delete data identified by unique link based on `UID`. This takes the shape of a DELETE request.
- `/fetch/{UID}`, fetch data identified by unique link based on `UID`. This takes the shape of a GET request.

In short, the repository application does the following:

- Identifies data identified by unique identifier.
- Compares the client submitted operation with the http request method.
- Update status and sign access requests.

## 6.5 Storage Providers and Data Hosting

In Chapter 2, it was presented several approaches in order to secure data in a cloud repository, not necessarily using blockchain. This section provides an approach which was not implemented, using Gnu Privacy Guard as a method of privacy and security. [gnu] Gnu Privacy Guard enables multi-key encryption, which works by encrypt a symmetric key with asymmetric keys. The primary issue with this approach is that it the owner of the data would be the only one performing updates on the data because of the re-encryption.

#### Client Setup for Secure Data Upload

In order to send data to the storage providers, it must be encrypted first by the user. This is the objective, but is not enforced and the cloud provider can still add some security. Any of the two options still allows the owner to control access to his data by other users in the network. Data encryption is not implemented, merely it was merely research. Nonetheless for data encryption a setup is needed.

Using Gnu Privacy Guard the user must generate a key using the identity `identifier` provided by the Hyperledger Fabric network, which can be obtained by querying the blockchain for a specific function of the `cid`

package, `GetID()`. This will would make re-uploading data an easier task for the user. This action, makes the sharing of public keys an easier task for the owner and the client application automating the process. An ideal scenario would be: at time of upload, checks the existence of valid share agreements and returns the users id, assuming the owner still has valid keys, and encryption follows.

### 6.5.1 Data Upload

Data upload is done in two parts, as described in Section 6.3.4. In the client side, data is encrypted using asymmetric key encryption and uploaded to the repository, where the repository flags the request as `ACCEPTED`. Data can be re-encrypted, before being sent, with a shared secret if the owner chooses it.

Assuming a single copy of the data, any time a new user needed access, re encryption with the public keys of every user would be required. This can all be automated with the blockchain, due to the setup and share agreement automatic validation at request creation and request validation.

### 6.5.2 Optimal Scenario for Sharing

When it comes to sharing of files, with this method, the best case scenario is uploading the encrypted data with Gnu Privacy Guard with all the users who need access, using multi-key encryption. This would require every party to be in the network before hand, and knowledge of every user requiring access.

#### Data in the repository

If the data is already hosted in the repository and a group of users require access to the data, the owner will follow the same procedure explained previously. The next step is to re-encrypt the data using the client application with the target recipient public keys, which may require to download the data once again. Since its just one copy, the client application queries the contracts with by `FILE ID`, returning every user's identity, so the client application can re-encrypt with all the recipient keys and re-upload.

#### Increased Protection

The first encryption uses GNU Privacy Guardian to encrypt the data. This encryption method allows for multi-key encryption, and was ideal for data sharing for multiple users and groups of users that do not interact with each other. This allows for data to be secure between users and the untrusted repository with the addition of the blockchain eco-system.

Additionally, the storage provider should provide its own encryption method as expected from cloud storage providers in order to ensure increased privacy and confidentiality. This pre-upload encryption is to ensure the owner has more control over the security and confidentiality of the data in case of untrusted repositories.

Additionally the cloud storage provider could perform a shared secret encryption, which would be generated by the Share Agreements and applied by the storage provider. This system would work at time of an access request validation by repositories, instead of just returning the `FILE ID` of the data. It would also return the shared secret, and the cloud storage provider would re-encrypt with the shared secret.

# 7

# Experimental Evaluation

*In this chapter we present some experimental test cases with the test setup. This chapter provides detail on the test environment setup and the test cases detailing the flow of permissions and file asset registration and malicious test cases.*

## 7.1 Test Environment

The test environment is built on top of a Hyperledger Fabric Samples tutorial, Write Your First Application Tutorial [hfw], that is composed of the essentials to run an application on the network, and make the network function. The components are the following:

- A single organization.
- An ordering service with one Orderer peer.
- Consensus Mechanism: `solo`.
- A single Certificate Authority.

| Request Fields | Request Values |
|---|---|
| Chaincode Identifier | Target Chaincode |
| Function | Target Function |
| Function Arguments | [arg0, arg1, ..., argN] |
| Channel Identifier | Target Channel |
| Transaction Identifier | txId |

Table 7.1: Request Payload Fields and Values

- A single channel.
- Three Smart Contracts.
- Three users. One of these acts as a repository.
- One administrator.

Peers and Certificate Authority are Docker containers from various Hyperledger Fabric Docker images. Peers and Certificate Authority are part of a Docker network in order to communicate.

The network is booted up by running a script that runs the `docker-compose.yaml` and a bash script to setup up the network. This includes setting up channel configuration, joining the peers into the channel, install and instantiates the smart contracts. When smart contracts are instantiated, it also boots up an individual docker container that is connected to peers who have installed and instantiated the smart contract for all the peers in the channel. After the script is finishing running, an administrator must be registered first before other users are allowed to join.

## 7.2 Test Cases

Test cases are divided into three sections, each one for each developed smart contract, and within these sections there are tests for intended behaviour of the network without malicious interference and for accidental or malicious behaviour of the network. Regarding the three sections: the first section of tests are for file asset management, including registration and querying available files. The second section are tests for Share Agreement creation and management. Including creation, unauthorized creation, and termination of the same. The third section are tests for access requests that range from authorized to unauthorized and timeout and revocation of access.

### 7.2.1 Transaction Proposal Request Payload

In order to use the properties of the smart contracts, a request must be filled that must specify the target contract, target function of the contract, arguments, target channel and finally a transaction identifier which will identify the transaction proposal.

Table 7.1 presents the format of a request payload and all the necessary fields to perform a transaction proposal. If it is a query proposal then `Transaction Identifier` is not required. Table 7.2 presents an example of an actual request payload for a transaction proposal on smart contract `File Management`, to query a file by the id of `FILE_01` on channel `mychannel` and the transaction id, to identify the transaction proposal. This last is not required unless for the updating of share agreements as it was previously stated in Chapter 6.

| Request Fields | Request Values |
|---|---|
| chaincodeId | 'file_management' |
| fcn | 'queryFile' |
| args | ['FILE_01'] |
| chainId | 'mychannel' |
| txId | txId |

Table 7.2: Example of a Request Payload Fields and Values

| Request Fields | Request Values |
|---|---|
| chaincodeId | 'file_management' |
| fcn | 'queryAllFiles' |
| args | [] |
| chainId | 'mychannel' |
| txId | txId |

Table 7.3: Request Payload for querying all files

### 7.2.2 File Management Tests

In order to have a valid share agreement between two entities, a valid asset registration must occur beforehand. Therefore the first step is to ensure a valid registration must occur. In order to query for shared files it must check the `share_agreements` smart contract.

#### Test 1: File Asset Registration

Pre-Requisite: Fill request payload with appropriate field values.
Expected Result: Commit of file asset.

In order to create an asset, an user must have a valid set of credentials which is generated by the Certificate Authority connected to the Organization. A user fills up the request form. If the request payload is well formed, and the user has a valid set of credentials the registration will always be accepted, assuming `FILE ID` is not already taken and there 4 arguments in the `args` parameter. `FILE ID, Location of File, Name of File, SHA512` If the `FILE ID` is not available or there are not 4 arguments in the `args` field of the request, the chaincode will perform a run time error and the transaction will fail.

#### Test 2: Inspect all Files with access

Pre-Requisite: File Asset registration of `FILE ID` or valid Share Agreements of `FILE ID`.
Expected Result: Return all file information.

The request payload for this test is represented in Table 7.3. For all query related operations, a `txId` is not required because it does not need to generate a transaction for just read operations. Although for validation of contracts, it needs to have a `txId`.

For this test, the `args` field is empty because the only information needed is the `userId` of the invoker which will be determined with the `cid` package at runtime, so the user does not need to provide it. The smart contract scans the file assets for `userId` and valid Share Agreements. Bundles up the results and returns to the applications which in turn show the results to the user.

| Request Fields | Request Values |
|---|---|
| chaincodeId | 'file_management' |
| fcn | 'queryFile' |
| args | ['FILE_01'] |
| chainId | 'mychannel' |
| txId | N/A |

Table 7.4: Request Payload for querying all files

| Request Fields | Request Values |
|---|---|
| chaincodeId | 'share_contracts' |
| fcn | 'createContract' |
| args | ["FILE_01", "user2, user3"', "60","TIME", "true"] |
| chainId | 'mychannel' |
| txId | txId |

Table 7.5: Request Payload for Share Agreement creation

#### Test 3: Inspecting a `FILE ID` with no access

**Pre-Requisite:** File Asset registration of `FILE ID`
**Expected Result:** Failed Transaction.

The request payload used for this test is presented in Table 7.4. If the `userId` of the invoker does not match the `ownerId` or an id in the `User Permissions List` of the data asset regarding `FILE ID` then the transaction will always fail. But the error message will not say if the given `FILE ID` exists or not.

### 7.2.3 Share Agreement Tests

A Share Agreement can only be created if, and only if the user submitting the transaction proposal matches the ownership field in the `FILE ID`. This implies `FILE ID` must exist beforehand. But does not require to be hosted in a remote storage provider.

#### Test 1: Valid Share Agreement Creation

**Pre-Requisite:** File Asset registration of `FILE ID`.
**Expected Result:** Commit of Share Agreement.

The request payload for this test is presented in Table 7.5. If the user has successfully registered a file with a valid `FILE ID`, it enables the sharing mechanism.

The first arguments in `args` parameter is the `FILE ID`, the second argument is a list of the target users, the third one is the time limit, 60 minutes in this case, and the last is to determine what type of contract is this (time based in this case). Since the current implementation only handles time based contracts, the forth argument can be considered useless for now, the final argument states if the user can modify the data.

The following enumeration describes the steps for a successful Share Agreement creation:

1. Internal invoke to `file_management` using the function `query_file` with the `FILE ID` provided in the

| Request Fields | Request Values |
|---|---|
| chaincodeId | 'share_contracts' |
| fcn | 'terminateContract' |
| args | ['FILE_01'] |
| chainId | 'mychannel' |
| txId | txId |

Table 7.6: Request Payload for Share Agreement creation

payload.

2. Determine if the `userID` using `GetID()` from the `cid` package from the invoker matches the `ownerID` in the file Registration. Assuming it found `FILE ID`.
3. Create a new Share Agreement based on the arguments provided in `args`.
4. Update `Users Permissions List` field by invoking `file_management` of the file `FILE ID` with the users provided in the `args` parameter.

If all these steps occur without interruption, the transaction is considered valid and commited on the blockchain.

#### Test 2: Invalid Share Agreement Creation

Pre-Requisite: No File Asset Registration or not enough arguments.
Expected Result: No commit of Share Agreement and failed transaction.

The request payload for this test is presented in Table 7.5. If the user does not provide a valid number of arguments from the `args` parameter, the transaction will fail because the first check on every function of every smart contract is to verify the number of arguments in `args` parameter.

Invalid creation of a Share Agreement occurs by two methods.

1. The user provides a correct number of arguments and an existing `FILE ID` either by an accidental action or malicious intention. This will cause the transaction to fail on step 2, since the smart contract will not grant unauthorized alterations on file data or access control over it. In order to do so, the malicious user would require to have ownership over file asset identified by `FILE ID` which must be done by the smart contracts with the original owners credentials or the owner changing the ownership. Additionally the error message is simply a transaction fail message. It will not explicitly say if the `FILE ID` exists or not.
2. The user submits an non existing `FILE ID`, which will make the transaction to fail at step 1. Additionally the error message is simply a transaction fail message. It will not explicitly say if the `FILE ID` exists or not. The error message is the same as the previous method.

#### Test 3: Automatic and Manual Revocation of Access

Pre-Requisite: File Asset registration of `FILE ID` and a Share Agreement of `FILE ID`
Expected Result: Commit of updated share agreement with `Terminated` field set to true.

Termination of a contract is one of two methods: manual or automatic. Manual termination requires a transaction proposal with the request payload described in Table 7.6, where it performs a validity check before granting

| Request Fields | Request Values |
|---|---|
| chaincodeId | 'access_requests' |
| fcn | 'createRequest' |
| args | ["FILE_01", "GET"] |
| chainId | 'mychannel' |
| txId | txId |

Table 7.7: Request Payload for Access Request creation

a revocation, which is made by an internal call to `file_management` to determine if `FILE ID` exists and if the invoker is the owner of said file. If it is not true, the transactions fails. But if these steps succeed then `Terminated` is set to true and the transaction is flagged as valid.

The second method is automatic. In table 7.5, the contract created has a timespan of 60 minutes. After 60 minutes it is no longer valid, and, when possible, change `Terminated` field from false to true. This requires a transaction proposal to change this field. But it is not of vital importance, because the timestamp comparisons will always be checked until an update occurs.

### 7.2.4 Access Requests Tests

In order for an access request to be processed it must come from a registered user. The `FILE ID` must be registered and the transaction proposal for an access request must be submitted by either a owner of the file asset or user with access permissions. Follows a series of test cases that range from expected behaviour to unexpected behaviour.

#### Test 1: Valid Access Request Creation

Pre-Requisite: File asset registration and/or Smart Contract registration if user is not an owner.
Expected Result: `UID` is returned and `status` is pending confirmation from repository.

Table 7.7 presents the request payload for the creation of an access request in order to fetch data from a storage provider. Upon request payload, when received by the smart contracts the following checks occur:

1. Check valid Share Agreements with the user `userId` for `FILE ID` in question and if found check the operation `GET` matches the value in `Can Modify` in the share agreement.
2. If no Share Agreement are found, it may mean the user is the owner of the file.
3. Check for ownership of the file using `file management` smart contract.

If step 1 or step 3 returns true, a valid `UID` is generated and the `status` of the access request is modified to pending. Awaiting storage provider confirmation.

#### Test 2: Invalid Access Request Creation

Pre-Requisite: File asset registration missing, and/or Smart Contract expired or was terminated.
Expected Result: Transaction succeeds but logs `status` field with value of ACCESS_DENIED for the request.

| Request Fields | Request Values |
|---|---|
| chaincodeId | 'access_requests' |
| fcn | 'validateRequest' |
| args | [uid, HTTP_REQUEST] |
| chainId | 'mychannel' |
| txId | txId |

Table 7.8: Request Payload for Access Request validation

Table 7.7 presents the request payload for the creation of an access request. An access request creation fails due to two reasons:

- There is no valid smart contract at the time corresponding to `FILE ID` in which the user is part of.
- The operation does not match value in `Can Modify?`.
- User is not the owner of `FILE ID`.

At the moment of creation of an access request, the smart contract will determine first if there are any valid Share Agreement with the `FILE ID` and `userId` that matches the invoker and the operation. If no contract is found, the contract checks for ownership of file via `file_management`. If no ownership is found then it flags the request as invalid but validates the transaction for the owner to know who has been illegally attempting to access his file.

#### Test 3: Valid Access Request Confirmation

Pre-Requisite: Users sends `UID` using request payload presented in Table 7.7.
Expected Result: `status` field value is modified to ACCEPTED and file is returned.

Table 7.8 presents the request payload for validation of an access request. A repository application upon receiving a valid request three conditions must be verified.

1. Check if the operation submitted matches the http request received.
2. Check if `UID` exists and has not been used before.
3. Check if `UID` is not over the time limit.
4. Check if `UID` still matches an owner or valid Share Agreement with the `is Owner?` flag.

If any of those steps fails, the smart contract modifies `status` to `ACCESS_DENIED`, the transaction is committed and the request process stops there.

#### Test 4: Timeout

Pre-Requisite: Users sends `UID` 60s after it was generated.
Expected Result: `status` field value modified to ACCESS_DENIED and no file is returned.

| Request Fields | Request Values |
|---|---|
| chaincodeId | 'access_requests' |
| fcn | 'createRequest' |
| args | ["FILE_01", "UPLOAD"] |
| chainId | 'mychannel' |
| txId | txId |

Table 7.9: Request Payload for Access Request creation

| Request Fields | Request Values |
|---|---|
| chaincodeId | 'access_requests' |
| fcn | 'validateRequest' |
| args | [uid, "GET"] |
| chainId | 'mychannel' |
| txId | txId |

Table 7.10: Request Payload for Access Request validation

The request for this test is presented in Table 7.8. Timeout only occurs with a valid `UID` had been generated. Upon generation a user has 60 seconds to use it or at the time of usage, the smart contract `access requests` is automatically determine if it has been over time limit or not.

When the user submits the `UID` via `URL` parameter, it is received by the storage provider which in turn checks the blockchain with the previous request payload. The `access_request` smart contract will fail at step 2 of the previous enumeration and flag the `status` field value with ACCESS_DENIED.

#### Test 5: Revocation of Access

Pre-Requisite: Valid Share Agreement of `FILE ID` but with read-only access.
Expected Result: The request is denied and the share agreement is terminated and transaction is flagged as valid.

Revocation of access can happen either when the client applications submits a transaction proposal with the payload presented in Table 7.9 or when a Repository Application receives the `UID` and compares the operation submitted by the client application presented in Table 7.7 does not match the HTTP request of GET presented in Table 7.10, as presented in Figure 6.5.

An update on the data can only occur if the share agreement for the given `FILE ID` has the field `Can Mofify` to true. If the request payload presented in Table 7.9, the `access_requests` smart contract will firstly determine if there is a valid share agreement for `FILE_01`, assuming this returns true, it then checks if the operation being conducted is possible. If it is not, the smart contract determines a violation and terminates the contract.

## 7.3 Evaluation

Various tests were conducted which approached several situations, the results from these tests were as expected which proved to be satisfying results.

Although these tests only assumed one storage provider registered in the network, which in turn facilitated the testing process because of only focusing on one storage provider instead of many, it was considered to be an

overall success where expectations were achieved. Adding more storage providers would certainly increase the testing.

# 8

# Conclusions and Future Work

*In this chapter we present the conclusion of the work done in this dissertation as well as the future work.*

The work described in this dissertation implements a data sharing mechanism solely based on the blockchain, introducing new concepts that are deemed disruptive and are the focus of the development of this work. The concepts are Blockchain and Smart Contracts, which are explained in depth in Chapters 3 and 4. Due to the nature of the applications of the project and the need for an identity based blockchain, to identity participants so the owner of the data can be clearly identified as well as as the users with access permission. A private blockchain implementation was the basis for this project, as explained in Chapter 5.

The ultimate goal of this work was to create a proof of concept where users are able to share the data in cloud storage providers having as little interaction as possible in order to shift the focus, decision power and business logic to the blockchain network, but must be stated that during the development and as stated in Section 7.3, only one storage provider was considered and a very simple one. Instead of having an interaction with the storage provider and having the storage provider handling every aspect related to the data of the user, or have additional services controlled by the provider, the user has an interaction with a decentralized network which provides uncensored, tamper proof information about the data sharing access and additionally, controlled by

the user and enforced, maintained and validated by the smart contracts. As importantly, besides the user controlling the access control via a decentralized network, it also provides a complete record of every interaction made with the data by any user approved solely by the owner, which both the data and users must be registered and identified in order to interact with it.

In Chapter 6 we described the decision process while developing a data sharing mechanism controlled by the user and validated by the smart contracts of the network. To do so there was the need for three smart contracts to describe the logic of creating an asset, creating a share agreement and finally create an access request for interaction between users and storage providers. Aside from the smart contracts, applications that interact with these smart contracts were also created and developed which allow users to freely interact with it.

Various tests cases were described in Chapter 7 showing the results of the developed project, which provides only authorized modifiable states and access control based on blockchain. Providing output for the owner or any other user to see who is accessing the data, who is altering the data, for how long can other users access the data, and with what permissions. This is achieved with a decentralized blockchain network and smart contracts to handle all the business logic. The objectives stated in Chapter 1 can be deemed fulfilled. This project was more shifted to the understanding of the Blockchain and Smart Contracts capabilities as the two most important components. This also allowed to reduce the centralization of the data mechanism from the storage providers and putting into the hands of the blockchain network.

Comparing the initial version and the final version of this dissertation the focus was shifted more towards Blockchain and Smart Contracts then that of Clouds, where in testing, as described in Chapter 7, it was used an application with very simple storage capabilities. However, the data sharing mechanism worked as expected and the goals of this work were achieved, although certain additional points are in need of some improvement, which are described in Section 8.1.

## 8.1 Future Work

Although the work describe in this work achieve the proposed goals, it can improved in many areas. Some of this areas include:

- Implementation of data encryption method for cloud hosting. It was studied a few approaches on data sharing encryption for cloud hosting and sharing.
- Hyperledger Fabric is all about identity over anonymity, a concept to improve would the in the area of the Membership Service Providers in order to improve overall identity and distinction between users, storage providers and any other identity deemed necessary.
- Incorporate private transactions as these were introduced in a later version of Hyperledger Fabric.
- Additional channel setup. In this dissertation a single channel was setup. An obvious improvement would to be to have applications with the option of setting up more channels, regarding the applications stated in Chapter 6.
- Expand to multiple Storage Providers. In the describe work, only one storage provider was considered.
- Optimization and Code Quality. regarding smart contracts efficiency this would permit serving a great number of requests and query optimization by using indexes and other database optimization tools to make the network scale in greater numbers. And as important, increased testing in order to discover and fix possible loopholes and security flaws missed during the development process.

# Bibliography


[ABC+18] Nicola Atzei, Massimo Bartoletti, Tiziana Cimoli, Stefano Lande, and Roberto Zunino. Sok: unraveling bitcoin smart contracts. In *International Conference on Principles of Security and Trust*, pages 217–242. Springer, 2018.

[ADK+17] Mazhar Ali, Revathi Dhamotharan, Eraj Khan, Samee U Khan, Athanasios V Vasilakos, Keqin Li, and Albert Y Zomaya. Sedasc: secure data sharing in clouds. *IEEE Systems Journal*, 11(2):395–404, 2017.

[AEVL16] Asaph Azaria, Ariel Ekblaw, Thiago Vieira, and Andrew Lippman. Medrec: Using blockchain for medical data access and permission management. In *Open and Big Data (OBD), International Conference on*, pages 25–30. IEEE, 2016.

[Amm16] Saifedean Ammous. Blockchain technology: What is it good for? 2016.

[B+02] Adam Back et al. Hashcash-a denial of service counter-measure, 2002.

[B+13] Vitalik Buterin et al. Ethereum white paper, 2013.

[BG17] Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget. *arXiv preprint arXiv:1710.09437*, 2017.

[blo] Blockchain. `https://www.blockchain.com/pt/`. Accessed October 2, 2018.

[Cac16] Christian Cachin. Architecture of the hyperledger blockchain fabric. In *Workshop on Distributed Cryptocurrencies and Consensus Ledgers*, 2016.

[CCT+14] Cheng-Kang Chu, Sherman SM Chow, Wen-Guey Tzeng, Jianying Zhou, and Robert H Deng. Key-aggregate cryptosystem for scalable data sharing in cloud storage. *IEEE transactions on parallel and distributed systems*, 25(2):468–477, 2014.

[CD16] Konstantinos Christidis and Michael Devetsikiotis. Blockchains and smart contracts for the internet of things. *Ieee Access*, 4:2292–2303, 2016.

[cid] GoDoc - cid package. `https://godoc.org/github.com/hyperledger/fabric/core/chaincode/lib/cid`. Accessed July 10, 2018.

[CPVK16] Michael Crosby, Pradan Pattanayak, Sanjeev Verma, and Vignesh Kalyanaraman. Blockchain technology: Beyond bitcoin. *Applied Innovation*, 2:6–10, 2016.

[DAK+16] Kevin Delmolino, Mitchell Arnett, Ahmed Kosba, Andrew Miller, and Elaine Shi. Step by step towards creating a safe smart contract: Lessons and insights from a cryptocurrency lab. In *International Conference on Financial Cryptography and Data Security*, pages 79–94. Springer, 2016.

[dbc] Interview with Gavin Wood. Accessed September 16, 2018.

[dpo] What is delegated Proof of Stake? Accessed September 16, 2018.

[eth] Ethereum Project. **https://www.ethereum.org/**. Accessed 15 February, 2018.

[gnu] Guard, GNU Privacy. Accessed October 5, 2018.

[Gua] GNU Privacy Guard. Encrypt files on Linux.

[Gup17] Manav Gupta. *Blockchain for dummies*. John Wiley & Sons, 2017.

[hfw] Hyperledger Fabric Write Your First Application. **https://hyperledger-fabric.readthedocs.io/en/release-1.0/write_first_app.html**. Accessed April 10, 2018.

[Hyp18] Hyperledger. hyperledger-fabricdocs documentation. Technical report, Linux Foundation, 2018.

[Kas17] Preethi Kasireddy. Ethereum how it works, anyway? **https://medium.com/@preethikasireddy/how-does-ethereum-work-anyway-22d1df506369**, 2017. Accessed 15 February, 2018.

[KN12] Sunny King and Scott Nadal. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. *self-published paper, August*, 19, 2012.

[Lew15] Antony Lewis. A gentle introduction to blockchain technology. *Brave New Coin*, 2015.

[LSZ15] Yoad Lewenberg, Yonatan Sompolinsky, and Aviv Zohar. Inclusive block chain protocols. In *International Conference on Financial Cryptography and Data Security*, pages 528–547. Springer, 2015.

[Nak] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system.

[pro] Ethereum Proof of Stake FAQs. **https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQs**. Accessed October 17, 2018.

[shi] GoDoc - shim package. **https://godoc.org/github.com/hyperledger/fabric/core/chaincode/shim**. Accessed February 15, 2018.

[sia] Sia. **https://sia.tech/**. Accessed February 15, 2018.

[sto] Storj. **https://storj.io/**. Accessed 15 February, 2018.

[SZ15] Yonatan Sompolinsky and Aviv Zohar. Secure high-rate transaction processing in bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 507–527. Springer, 2015.

[Sza97] Nick Szabo. Formalizing and securing relationships on public networks. *First Monday*, 2(9), 1997.

[TNV18] Parth Thakkar, Senthil Nathan, and Balaji Vishwanathan. Performance benchmarking and optimizing hyperledger fabric blockchain platform. *arXiv preprint arXiv:1805.11390*, 2018.

[VC14] David Vorick and Luke Champine. Sia: Simple decentralized storage. 2014.

[Vuk17] Marko Vukolić. Rethinking permissioned blockchains. In *Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts*, pages 3–7. ACM, 2017.

[WLB14] Shawn Wilkinson, Jim Lowry, and Tome Boshevski. Metadisk a blockchain-based decentralized file storage application. *Technical Report. Technical Report*, 2014.

[Woo14] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151:1–32, 2014.

[XSA+17] Qi Xia, Emmanuel Boateng Sifah, Kwame Omono Asamoah, Jianbin Gao, Xiaojiang Du, and Mohsen Guizani. Medshare: Trust-less medical data sharing among cloud service providers via blockchain. *IEEE Access*, 5:14757–14767, 2017.

[XSS+17] Qi Xia, Emmanuel Boateng Sifah, Abla Smahi, Sandro Amofa, and Xiaosong Zhang. Bbds: Blockchain-based data sharing for electronic medical records in cloud environments. *Information*, 8(2):44, 2017.

[ZN+15] Guy Zyskind, Oz Nathan, et al. Decentralizing privacy: Using blockchain to protect personal data. In *Security and Privacy Workshops (SPW), 2015 IEEE*, pages 180–184. IEEE, 2015.

[ZRL+10] Gansen Zhao, Chunming Rong, Jin Li, Feng Zhang, and Yong Tang. Trusted data sharing over untrusted cloud storage providers. In *2nd IEEE International Conference on Cloud Computing Technology and Science*, pages 97–103. IEEE, 2010.