POLITECNICO DI TORINO

Corso di Laurea in Ingegneria Informatica

Tesi di Laurea

Providing trust to multi-cloud storage platforms through the blockchain

Relatore
Cataldo Basile

Claudia Fiore

Aprile 2019

To my grandparents

Summary

Cloud storage services are currently a commodity that allows users to store data persistently, access the data from everywhere, and share it with friends or co-workers. The number of cloud services is growing rapidly but with low interoperability between them; consequently, managing and sharing files between users of different cloud storage is very difficult. To address this problem, specialized cloud aggregator systems emerged that provide users a global view of all files in their accounts and enable file sharing between users from different clouds. To remove the need to trust the cloud providers, Crypto Cloud solution provides the full encryption of stored data, allowing to use multiple cloud storage providers to securely store files.

However, in Crypto Cloud, there is a central server which is responsible for managing metadata about users, clouds, files, and permission. The general problem is that if the server is attacked, the integrity of files and public keys can be compromised. The Crypto Cloud system was created with the assumption that the server does not act maliciously. In this dissertation, we propose a solution that, through the use of the blockchain, is able to provide integrity of metadata without relying on the server. This is achieved by extending Crypto Cloud with secure metadata management using the blockchain. We focused on the management of the users' identities and how to provide metadata integrity without relying on the central server. We built a prototype and tested it with real use cases such as the addition of a user, or creation/reading of files. While more complex, the new client removes the trust from the central server and is, therefore, a step towards more decentralized and secure cloud storage systems.

Acknowledgements

First and foremost, I would like to express my genuine gratitude to my academic supervisors Cataldo Basile, Ricardo Chaves and Miguel Matos for giving me the opportunity to work on this project and for the continuous support and guidance throughout this research. Their patience, motivation and immense knowledge provided me the proper guidance during the research and writing of this thesis.

I would like to thank my parents and my sister for their deep and unconditionally love and in particular for the support in this Erasmus experience. Finally, I would like to express my gratitude to my friends Andrea, Camilla and Cristina for their support and constant presence during these months spent in Lisbon.

Contents

Chapter 1

Introduction

In recent years the use of cloud storage services such as Dropbox or Google Drive has increased exponentially. These services offer a remote storage space on which it is possible to save important data, accessible at any time. The use of cloud storage services also allows to drastically reduce the risk of data loss, in fact, the cloud provider periodically creates backup copies in a completely transparent way for the end user. Moreover, Cloud Storage Services provide new features to manage files, such as file sharing, file versioning, concurrent access or disaster recovery. However, the interoperability between providers and platforms is very low. To address this problem, specialized cloud aggregator systems emerged that provide users a global view of all files in their accounts and enable file sharing between users from different clouds. Such systems, however, have limited security: not only they fail to provide end-to-end privacy from cloud providers, they also require users to grant full access privileges to individual cloud storage accounts.

In this work, we focus on Crypto Cloud [1], a privacy-preserving cloud aggregation service. It allows for using multiple cloud providers without renouncing privacy, guaranteeing the confidentiality and integrity of managed files. Crypto Cloud enables file sharing on multi-user multi-cloud storage platforms. The Crypto Cloud application relies on a central server which manages all the information (metadata) related to users, files, permission and cloud storage. In Crypto Cloud, there is the assumption that the server does not modify the metadata actively. He could access them, but it acts honestly. If malicious users attack the server, the integrity of metadata could be compromised. Therefore, the goal of this thesis is to extend Crypto Cloud with metadata management that ensures that security sensitive operation can be performed only on trusted client endpoints. The resulting system should leverage blockchain technology to enforce decentralized metadata management securely; it manages the integrity of metadata and management of users' identities through the blockchain.

1.1 Motivation

Blockchain technology offers new tools that prevent the need for central administrators. Storing blocks of information across the network, the blockchain cannot be controlled by any single entity and has no single point of failure. Whatever is on the blockchain is not modifiable.

Moreover, the use of a distributed architecture implies performance advantages compared to a centralized solution, including greater scalability and availability. The main components of Crypto Cloud are: the Client Application, the Directory Server and the Cloud Stores. The Crypto Cloud Directory Server acts as a metadata repository of users, clouds, files and permission information. As an assumption, this server is considered *"honest but curious"*. It means that the server can listen to the exchanged messages but follows the system's protocol and does not launch active attacks [1]. The server can read the metadata but does not modify them. In order not to trust the server anymore, we must provide a mechanism to protect the metadata integrity. This functionality can be delivered with the blockchain, whose main property is the integrity of the stored information. In this research project, we explore the potentiality of the blockchain technology to reduce the trust on the central server of the application Crypto Cloud. By integrating this technology with Crypto Cloud, it is possible to implement authentication and provide integrity of the metadata without relying on the server. There will be no central point of failure, and the assumption of the trustworthy server will not be longer necessary because with the blockchain we do not need to trust in any third party.

The main goal of this thesis is to extend Crypto Cloud with metadata management that ensures security sensitive operation relying on the blockchain and develop a Proof of Concept to prove the feasibility of the designed solution. The actual implementation of Crypto Cloud Application manages the user's cryptographic keys and their certification relying on an external service. It is possible to modify this functionality, managing the users' identities both with the blockchain and adding an application logic on the client; in this way we can achieve a robust identity management but in a decentralized and secure way. Moreover, Crypto Cloud guarantees the integrity and versioning of files but, as mentioned before, the integrity of the metadata is not guaranteed. The hash of files, the versions, and the Access Control Lists are critical metadata to provide validity of files, so we must protect them from tampering. Moving that metadata to the blockchain we can ensure integrity and freshness of that information. In addition, we developed a control on the client in which, every time an operation is performed, there is a check between the server's metadata and the blockchain's metadata; in this way, we guarantee the system correctness.

The new version of Crypto Cloud should have the same functionality as before. It must also provide the following functionalities:

1. the system must provide integrity of the metadata without trusting the server;
2. the system must provide a mechanism to validate the public keys and associated identities of the users.

1.2 Thesis Outline

The rest of this document is organized as follows:

- **Chapter 2** describes the current Crypto Cloud system;
- **Chapter 3** presents background about blockchains and its main characteristics;

- **Chapter 4** presents an overview of the blockchain state of the art and the choice of the blockchain system to use in our Proof of Concept;
- **Chapter 5** describes the proposed solution, detailing its architecture and security models;
- **Chapter 6** describes the implementation details of the Proof of Concept;
- **Chapter 7** presents the evaluation of the proposed solution and the analysis of the obtained results from performance and security perspectives;
- **Chapter 8** concludes this document, summarizing the developed work and presenting improvements that should be taken into account for future work.

Chapter 2

Related work

In this chapter, we detail the Crypto Cloud system, a privacy-preserving cloud aggregation service that enables file sharing on multi-user multi-cloud storage platforms [1]. The Crypto Cloud application is based on Storekeeper's approach [2] and extends it by introducing new management of users' identities, authentication and integrity protection over files. The goal of this thesis is to integrate Crypto Cloud with the blockchain. Therefore, it is essential to provide an overview of the system to understand its functionalities.

2.1 Storekeeper

Storekeeper [2] is a cloud storage aggregator that guarantees data confidentiality without compromising the user's credentials. Storekeeper presents the different cloud stores as a single storage workspace and enables secure file sharing between users without requiring individual cloud accounts. Figure 2.1 depicts the application architecture.

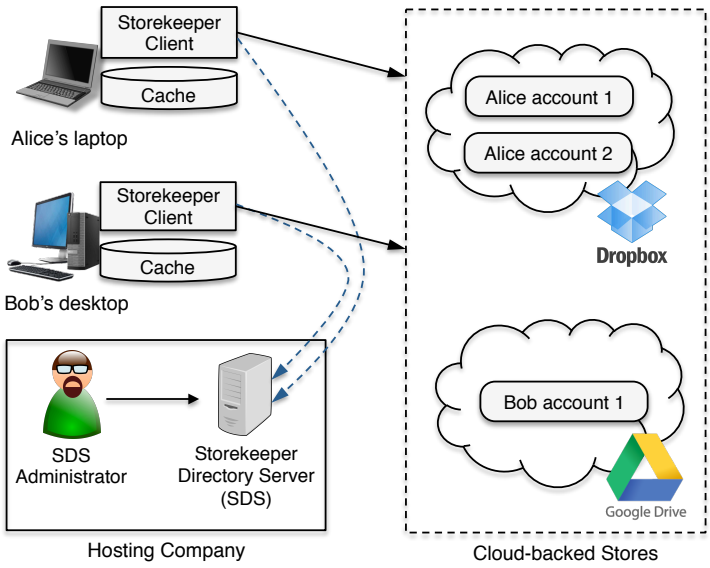


Figure 2.1: Storekeeper Architecture [2]

Storekeeper is composed of two main components: a client application and the Storekeeper Directory Server (SDS). The client application is installed on the user's device and is responsible for accessing the cloud and maintaining a local cache of the user files. The server is responsible for managing metadata, users, files and cloud services. The Storekeeper Directory Server (SDS) is assumed to be "*honest but curious*" [2]. Therefore, the server can listen to messages and learn information, but follows the protocol and does not launch active attacks. So, it does not access cloud storage spaces, nor store user files.

The Storekeeper system does not preserve data integrity and availability but, in terms of consistency semantics, the system allows file versioning.

2.1.1 Identity Management

The system generates and associates a pair of RSA asymmetric keys to the user, used in data protection mechanisms. The user's correspondent private-key pair is protected with a cryptographically secure password (or key) that is only known to the user. This encrypted credential is then stored at the SDS.

2.1.2 Access Control Model

Storekeeper implements an access control mechanism based on ACLs where users can have three privileges over a file: read, write, and share. Read privileges only allow the user to read the file. Write privileges allow both read and write operations such as create, update and delete the file. Share privileges allow reading, writing and sharing a file with other users and change permissions of the file. Exceptionally, the file's owner has full access privileges. The SDS will do Access Control List (ACL) checks on every request to enforce access control policies.

The system guarantees end-to-end confidentiality by encrypting files at the client side with an encryption key (KF). KF is then encrypted using the owner's public-key pair of RSA asymmetric keys. In this way, Storekeeper assures that only the owner can access the file. The file sharing service is implemented by following a simple protocol: when a file owner wants to share a file with another user, a symmetric key Read Key is generated and used to encrypt KF. The Read Key is then encrypted using the other user's public key, allowing the user to decrypt Read Key (RK) and KF, and consequently decrypt the file's content.

Whenever a writer submits a file update, a new file encryption key is generated and used to encrypt the new content of the file. This key is then encrypted with the current RK and distributed among the group's users using their private key. Every time a revocation occurs, a new Read Key has to be generated and distributed among the new set of users using their public keys; a revoked user will not be able to read future updates. The system also has to update the ACLs at the SDS on every file's group members change.

2.2 Crypto Cloud

Crypto Cloud is a distributed system that provides a secure cloud aggregation service for multi-user multi-cloud storage platforms [1].

Crypto Cloud users register their individual cloud storage accounts from Dropbox or Google Drive. Through a unified file namespace, users can seamlessly browse files across different cloud accounts and share files with other users. Crypto Cloud includes mechanisms to handle user identities from different cloud providers and provides key management and storage solutions that are easy to use by users. The system provides adequate security model and enforcement mechanisms that mask the diversity of file permission models across the cloud provider. It enables secure read and write operations to shared files located in different cloud accounts.

The Crypto Cloud system architecture is depicted in Figure 2.2. In the following sections we explain the components of the platform.

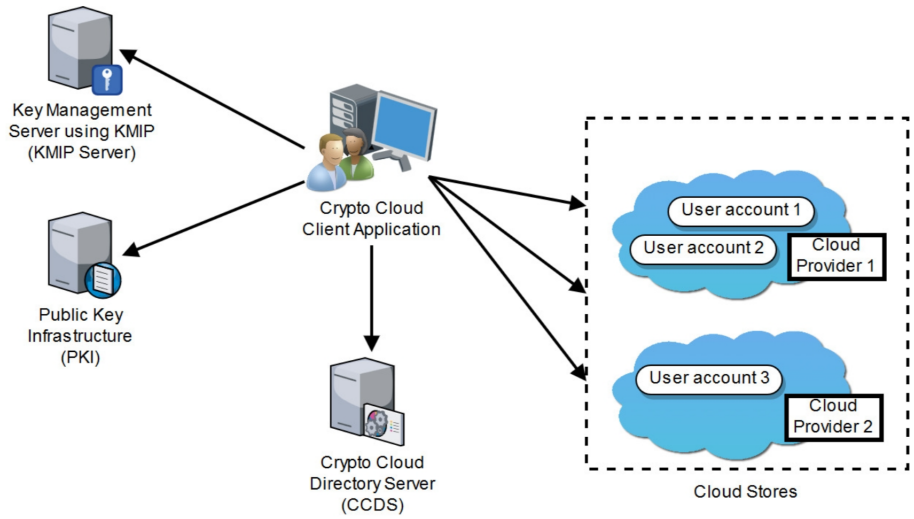


Figure 2.2: Crypto Cloud Architecture [1]

2.2.1 Crypto Cloud Client Application

The Crypto Cloud (CC) Client Application is the central component of the system. This component manages the user's files, and it interacts with all the other components of the system. The application provides the following services:

- upload and download the files of the authenticated users;
- allows performing special operations, such as the generation and replacement of the cryptographic key pair of the user;
- request for the registration of new users, requiring the permission to the administrators.

The client is composed of different modules. The first module is the Session Manager which preserves the local state of the app, after the correct login of the user. It maintains info about the user (username, cloud account, etc.) and the managed user's file (filename, version, share revision, etc). Moreover, there is the Communication Manager which communicate with the Crypto Cloud Directory Server (CCDS). During the login, it requires the server to authenticate the user and receives from it the access token to future access.

Another module is the Key Manager which controls the key pair of the users, collaborating with Key Management Interoperability Protocol (KMIP) Client Application Programing Interface (API) to access to the cryptographic key and to execute the cryptographic operations. Finally, the File Manager maintains the workspace which contains the local copy of the user's files. This module is capable of creating, reading and writing files. It also manages two temporary directories: one for the temporary files (e.g., downloading files) and one for conflicts (like name conflicts or versioning conflict).

2.2.2 Crypto Cloud Directory Server

The CCDS serves the client application. As an assumption, the CCDS server do not act maliciously.

It is possible to summarize the main features of the CCDS in:

- Metadata management. The metadata associated with: users, files, shares, and clouds;
- Access Control Model based on authentication;
- Version control and Integrity of files.

Its main function is maintaining the metadata to manage the files, keeping track of them on cloud stores. All the metadata are stored in a database shown in Figure 2.3.

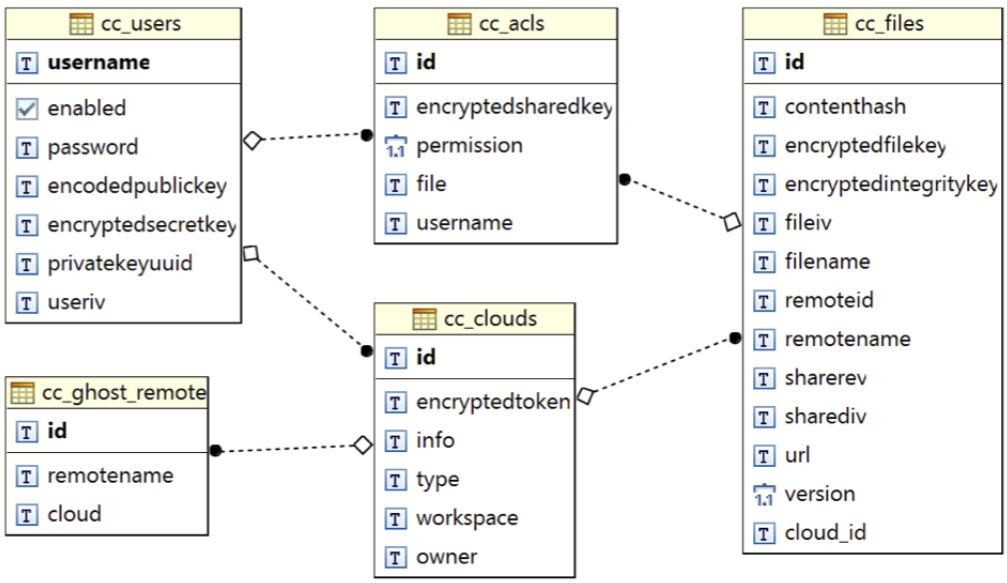


Figure 2.3: Database of Crypto Cloud Directory Server [1]

The server also implements an Access Control Model, storing an Access Control List that contains a unique binding user-resource (file), the permission granted and the wrapped Read Key, which was wrapped using the user's public key during

sharing algorithm execution. The access control mechanism is verified at every request.

One of the most important aspects of Crypto Cloud is the protection of managed files, which guarantees the confidentiality and integrity of users' sensitive files on public clouds. This mechanism is implemented relying on a Message Authentication Code (MAC) function. It is calculated using a symmetric key, called Integrity key (IK), over the content of the plaintext file, producing a fixed-size hash result. The produced result is stored concatenated in plaintext alongside with the ciphered content, taking the firsts bytes of the encrypted file. Every time a user reads a file, the HMAC is recalculated and verified with the one retrieved from the server.

Finally, the central server also implements a Version Control Mechanism, which relies on two metadata elements: the file content hash and the file version. The file content hash represents the hash result of the content of the file before encryption. This result is calculated by the client's application and sent to the CCDS. The file version is a number managed by the CCDS, which is incremented every time a user performs an update operation over the respective file. The association of both these elements ensures that a certain file's content represents a specific file version.

2.2.3 Key Management Server and PKI infrastructure

The Key Management Server is responsible for the access and protection of the cryptographic keys of the users. It follows the KMIP Protocol [3], allowing the users to generate, obtain and destroy the keys. The KMIP Server is responsible for managing cryptographic keys, which must be deployed in a secure environment. When performing the operations' actions, this component interacts with the HSM and database components to perform cryptographic operations and access persistent data. The HSM component is responsible for performing cryptographic operations (e.g., generate/export cryptographic keys and cipher/decipher data). We also have the KMIP Client, which consists of a simple component that provides access to the remote cryptographic resources from the KMIP server.

The Crypto Cloud Application also includes a Public Key Infrastructure. It is a third trust party which manages (emits, execute binding and revoke) digital X.509 certificates to validate the public key of the users. It also maintains a CRL to keep track of the revoked certificates. The certificates are signed with the public key of the infrastructure, becoming trusted certificates.

The Key Management Server and the Public Key Infrastructure are responsible for implementing the Key Management Interoperability Protocol (KMIP) standard, which allows the remote access and management of user's cryptographic keys, and certification of the users' identity, guaranteeing proper authentication while protecting and sharing sensitive files.

2.2.4 Cloud Stores

They represent the cloud account registered on the system by the users. They are passive stores, meaning that they store the files without executing any application logic. The Client App authenticates the user to the cloud systems, receiving an access token for the future accesses.

2.2.5 Base algorithm

This Section explains the basic algorithm implemented by the system. The functional process involves the management and protection of users' files. Consider only read and write operations, and not the sharing operations.

Figure 2.4 depicts the steps executed when a user wants to write a file. Therefore, the application does the following steps:

1. generates a new symmetric cryptographic key, called File Key (FK) and another symmetric cryptographic key, called Integrity Key (IK);
2. performs the encryption of the user's file, using the previously generated FK;
3. use IK to perform a Hash-based Message Authentication Code (HMAC) calculation over the content of the plaintext file, producing a fixed-size hash result;
4. store the HMAC in plaintext alongside with the ciphered content;
5. upload the encrypted file to the user's cloud storage and its metadata to the CCDS. FK and IK are wrapped using the user's public key and the wrapping result stored at the CCDS.

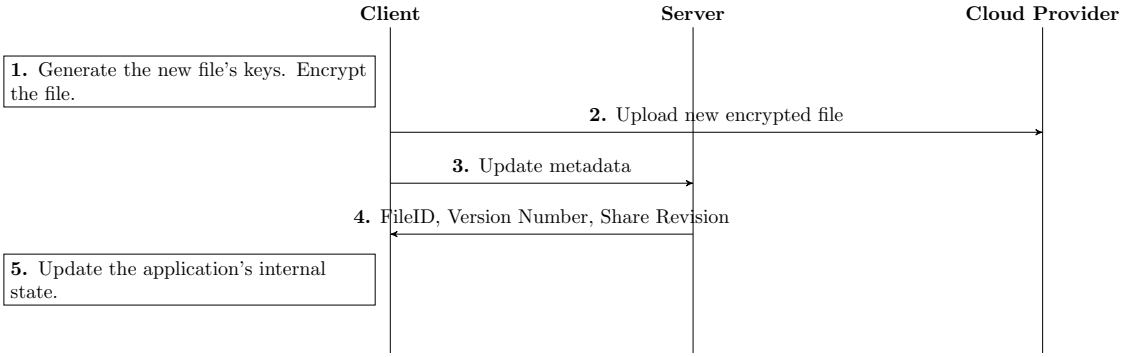


Figure 2.4: Write operation

As depicted in Figure 2.5, if a user wants to read that file the user's client:

1. downloads the encrypted file and retrieves the HMAC;
2. fetches the wrapped FK and IK from the CCDS;
3. obtains FK and IK by unwrapping them using his PK;
4. decrypts the content of the encrypted file using the obtained FK;
5. the application performs the HMAC calculation over the decrypted content, using the obtained IK and compared it with the retrieved result from the CCDS. If both results match, it is considered that the file was uncorrupted. Otherwise, it means that the content of the file was tampered and compromised.

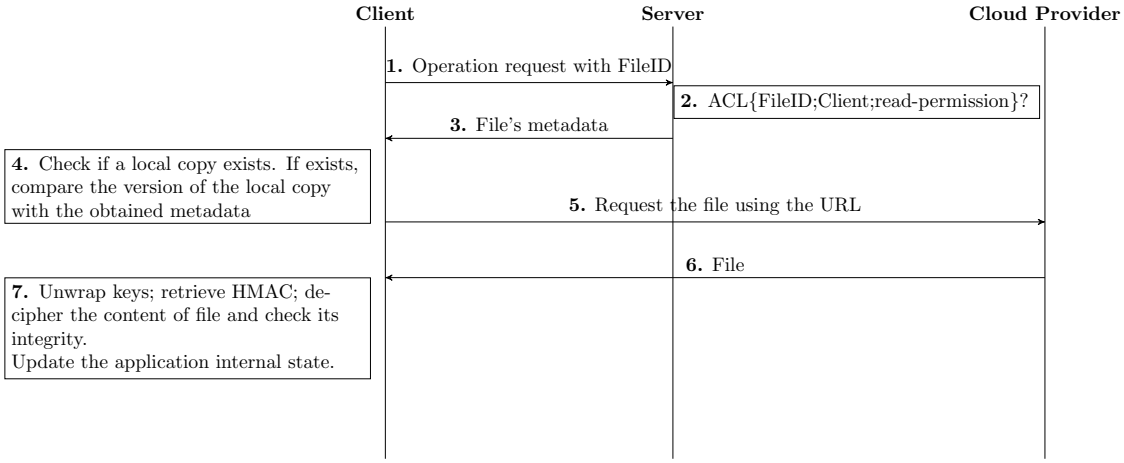


Figure 2.5: Read operation

2.2.6 File Update

This Section describes how the CC Application works in case of file update.

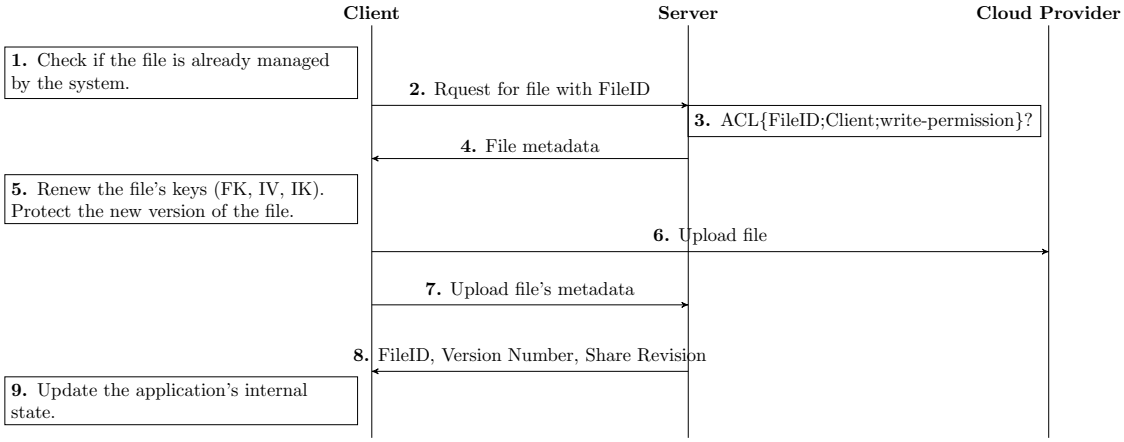


Figure 2.6: Update operation

To perform a file update (see Figure 2.6), the client Application does the following:

1. checks if the system already manages the file;
2. consults the ACLs' entries related to the user and checks if the user has permission to update the file;
3. obtains the file's metadata from the CCDS;
4. renews the file's keys, which comprises the FK, its IV, and the IK;
5. protects the new version of the file using the keys generated on the previous steps;
6. uploads the newly encrypted file to the cloud storage and update the corresponding metadata on the CCDS and updates the application's internal state with the new file's metadata.

2.2.7 File Sharing

In addition to the basic algorithm, the Crypto Cloud application provides the sharing functionality. The share operation allows a user to share a certain file with another user, called grantee user. Figure 2.7 depicts this mechanism.

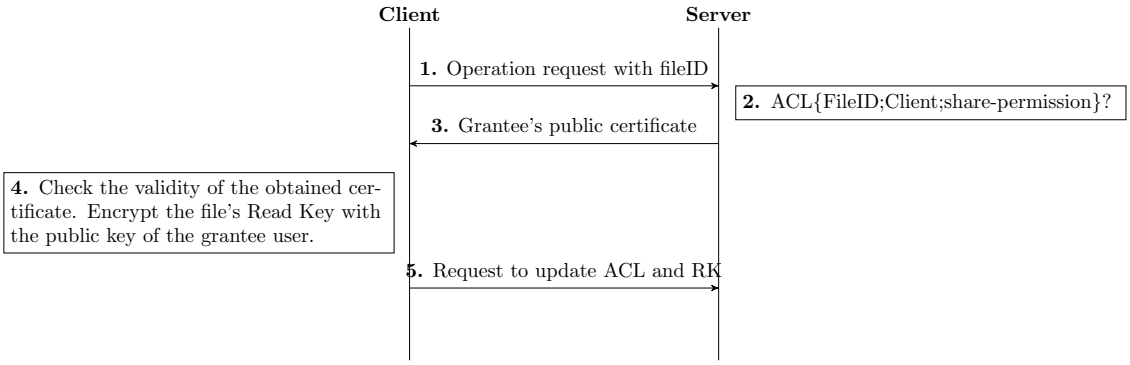


Figure 2.7: Share operation

When a user wants to share a file with another user, the user's client:

1. sends a request to the CCDS that accesses the ACLs entries and verifies if the user has, at least, share permissions on the selected file;
2. obtains the grantee user's public certificate from the CCDS;
3. checks the validity of the grantee's digital certificate;
4. encrypts the file's current Read Key (RK), using the public key obtained from the grantee's certificate;
5. updates the file's ACL, creating a new entry for the grantee user, with the assigned permission and previously encrypted RK.

2.3 Summary

Table 2.1 summarizes the features of the previous presented approaches. These systems are compared based on their data and metadata protection properties, sharing and file versioning mechanisms.

System	Data Sharing	Data Confidentiality	Data Integrity	Data Versioning	Metadata Confidentiality	Metadata Integrity
Storekeeper	✓	✓	✗	✓	✗	✗
Crypto Cloud	✓	✓	✓	✓	✗	✗

Table 2.1: Comparison of existing solutions for secure cloud systems.

Storekeeper [2] supports multiple cloud providers and is supported by a directory server to store metadata. It is assumed that this server does not launch active attacks, thus metadata protection was not part of the project's scope.

The system guarantees data confidentiality using symmetric keys, that are distributed to users using public-key pairs. The approach also provides sharing and access control mechanisms. However, this approach does not consider data integrity and employs an inefficient key management mechanism that performs an extensive number of requests to the SDS, resulting in a decrease in the application's performance.

Crypto Cloud system consists of an augmented version of Storekeeper, a system capable of providing cloud's features without trusting on its providers [1]. In addition to the Storekeepers functionalities, the Crypto Cloud system guarantees the integrity of files and provides robust key management. To address the key management, Crypto Cloud implements the KMIP protocol, which provides remote management of the users' cryptographic keys. Moreover, Crypto Cloud guarantees proper certification of the users' keys and assures their identity when accessing the managed files relying on a Public Key Infrastructure, which acts as a trusted third-party entity.

Chapter 3

Background

This chapter presents the required background on blockchains. It introduces the most important concepts of the blockchain, which are essential for its applications. In Section 3.1 we present what is the Distributed Ledger Technology and the the definition of the blockchain. Section 3.2 and Section 3.3 present the basic concepts involved in the blockchain, such as blocks and transactions. Then, in Section 3.4, there is the classification of the blockchain systems. Finally, Section 3.5 shows the main existing consensus mechanisms.

3.1 Distributed Ledger Technology and Blockchain

The Ledger is the fundamental basis of accounting [4]. It can be said that the Ledgers represent one of the foundations of our civilization and of our way of interpreting and managing relationships and transactions between people and organizations. Therefore, a ledger can be defined as a register used to record assets, which can be financial, physical or electronic. It allows establishing a historical memory, to check, verify, manage the transactions and exchanges that have been carried out. The ledgers became digitized with the rise of computers in the late 20th century, although computerized ledgers generally mirrored what once existed on paper. For a long time the ledgers were interpreted with the same centralized logic that characterized the paper. There was someone who managed the systems and someone who centrally managed the extraction and processing of data.

21st-century technology has enabled the next step in record-keeping with cryptography, stronger compute power and near-ubiquitous computational power, making the distributed ledger an increasingly viable form of record-keeping. Therefore, it is possible to create digitally distributed ledgers that can be shared across a network of multiple sites, geographies or institution. In other words, a distributed ledger is a database that exists across several locations and among multiple participants. All participants within the network can have their identical copy of the ledger. This technology is used to process, validate and authenticate transactions. The records are only ever stored in the ledger when a consensus has been reached by the parties involved. All the participants on the distributed ledger can view all of the records in question. DLTs drastically reduce the cost of trust, in fact, the main feature of this technology is that the ledger is not maintained by any central authority.

A blockchain is a form of Distributed Ledger Technology. Data on the blockchain are grouped and organized in blocks, which are linked one to the other and secured using cryptography (see Figure 3.1). Every block holds a complete list of transaction records. Distributed Ledgers do not require to have a data structure in blocks, and this is why the blockchain is a type of DLT and not the same thing. In other words, all blockchains are distributed ledgers, but not all distributed ledgers are blockchains.

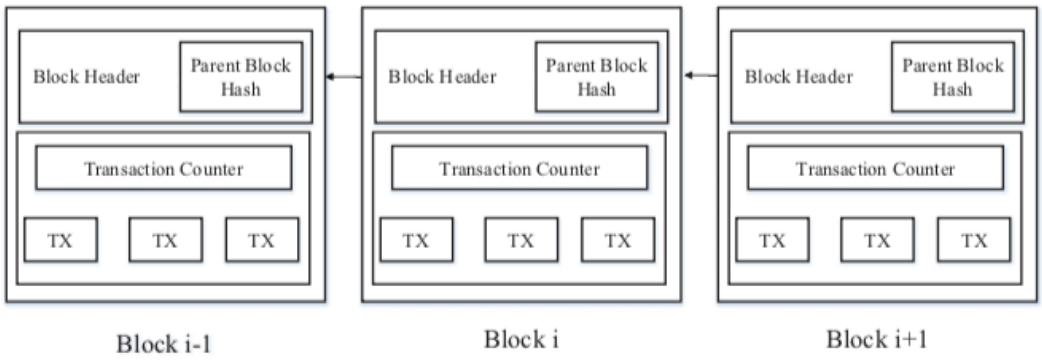


Figure 3.1: The blockchain [5]

The blockchain has the following main characteristics [6]:

- Immutability. The hash guarantees that the blocks are linked to each other, but it also guarantees the integrity of the blockchain. If one block is altered, the hash on the block that follows it stops matching the blocks;
- Transparency. Anyone can view the records and verify the authenticity of the data;
- Decentralization. Rather than relying on a central authority to securely transact with other users, blockchain utilizes innovative consensus protocols across a network of nodes to validate transactions and record data in a manner that is incorruptible.

In a blockchain network, generally, there are three types of nodes: miners, full nodes and light nodes. The nodes specialized in creating blocks are called *miners* and the activity of creating blocks is the "mining" operation. Instead, the task of a *full node* is to pass data to other nodes and check that the new blocks added are valid. This consists in verifying transactions, checking that the hash exists at the previous block, checking that the hashes of the new block are correct and other similar operations. Usually, altough not mandatory, a full node keeps a copy of the entire blockchain locally [7]. *Light nodes* are typically devices with limited resources, such as smartphones or devices of the Internet of Things. These nodes do not have the copy of the entire blockchain, and the data for processing are passed to the main nodes.

Nowadays most of the new blockchain technologies allow implementing real programs called "smart contracts". A smart contract is composed of code and data. Basically, it is a program that resides in a specific address of the blockchain.

From another point of view, it can be considered a contract between different parts, managed automatically, that specifies the conditions under which a transaction can be made. It is a computer protocol that facilities the transfer of digital assets between parties under the agreed-upon stipulations or terms. A smart contract is similar to a traditional contract in most ways including the definition of rules and penalties around the agreement except for the fact that it can also enforce the agreed-upon obligations automatically [8].

3.1.1 Security Characterstics

This section discusses the most important security properties in a blockchain system:

- Integrity: for a message to have integrity it means that cannot have been modified during its transmission. Traditional authentication mechanisms rely on digital signatures that allow a party to digitally sign its messages. Digital signatures also provide guarantees on the integrity of the signed message.
- Authentication: it requires that parties who exchange messages are assured of the identity that created a specific message so to be sure you're communicating with the real person rather than an impersonator;
- Authenticity: it is the assurance that a message, transaction, or other exchange of information is from the source it claims to be from. Authenticity involves proof of identity.
- Confidentiality: it means limiting the access to information. In other words, the only authorized person can access the information;
- Privacy: it means that the identities of participating nodes are protected.

3.2 Transactions

One of the purpose of transactions in a blockchain is to transfer ownership of a digital asset from one person/part to another person/party without the need for validation by a third party (a government or an external authority). Transactions are not encrypted, so it is possible to browse and view every transaction ever collected into a block. Once transactions are confirmed, they can be considered irreversible. To confirm a transaction it is necessary to verify that who sends a certain amount of asset must be able to prove that only he/she has control over that. A transaction changes the state of the agreed-correct blockchain. In a transaction there is the address of who sends and the other of who receives assets. Usually, the address is derived from the public key. Using this kind of mechanism, the system guarantees what is called pseudo-anonymity: in the first instance we do not know whom an address belongs to, but by analyzing the traces left on the network it is possible to trace back to the user who used that address. Figure 3.2 depicts the structure of typical transactions. Transactions contain an input that is a reference to an output from a previous transaction (a hash of the previous transaction).

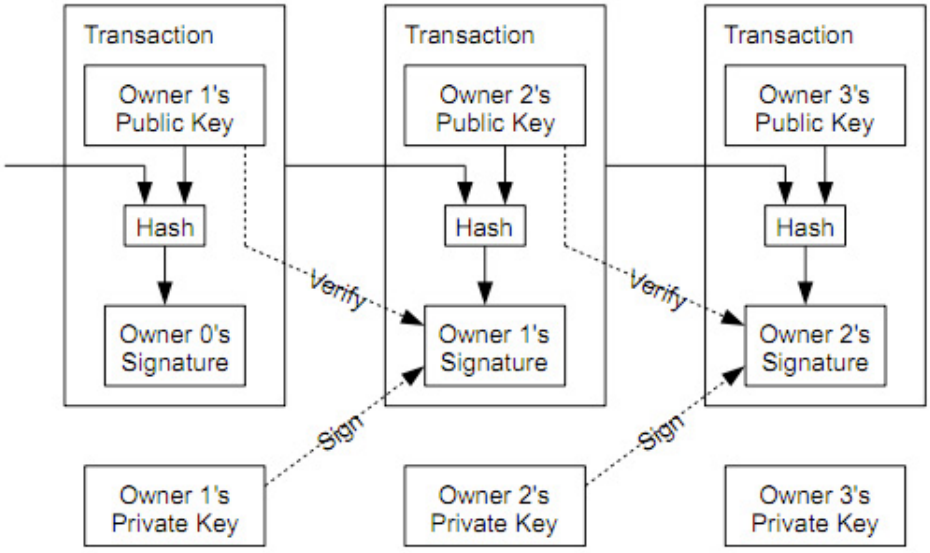


Figure 3.2: Structure of transactions [9]

In the input, there are also a signature and a public key. The public key must match the hash given in the script of the redeemed output. The public key is used to verify the redeemers signature, which is the second component that proves the transaction was created by the real owner of the asset in question. This continual pointer of inputs to previous transactions outputs allows for an uninterrupted, verifiable stream of value amongst addresses. Transactions contain also an output that specifies an amount and an address. Transactions are bundled and delivered to each node in the form of a block. As new transactions are distributed throughout the network, they are independently verified and "processed" by each node.

3.3 Blocks

Using the concept of the distributed ledger, it is possible to see a block as a page of this accounting register, in which transactions are marked, step by step, during a certain period of time. Each block contains, in addition to transaction data, a header with metadata namely time stamp and the hash of the previous block, as shown in Figure 3.3.

The block header is the metadata that helps verify the validity of a block. Each block has its "fingerprint" represented by the hash of the data it contains. The next block contains this hash: it is this "backward" reference that creates the chain of blocks. This order is very useful for verifying the internal consistency of the chain. If someone wanted to change the details of a transaction within a block, he/she would change the hash, so he/she would then need to recalculate a valid hash for the next block. This mechanism is necessary because the hash of the next block is based on the original hash of the previous block. By changing the hash of the current block and the one of the next block, every block after them is instantly invalid. It would need to recalculate every hash to consider the change valid. Every block's state of validation depends on all the blocks before it. This mechanism determines the immutable character of the blockchains [11].

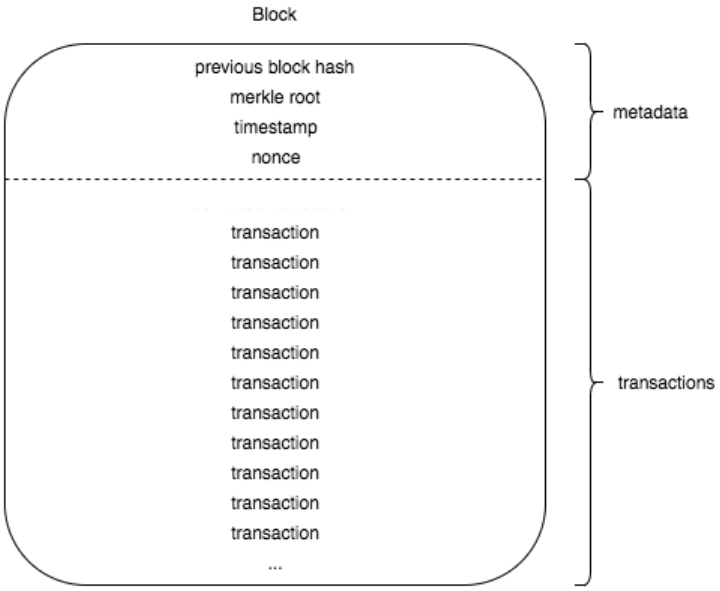


Figure 3.3: General block structure[10]

The longer the blockchain gets, the harder it is to change.

3.4 Classification of a blockchain system

There are various categorizations of blockchain types, according to whether authorization is required for network nodes which act as verifiers, and whether access to the blockchain data itself is public or private. Permission refers to the authorization for verification, and anybody can join the network to be a verifier without obtaining any prior permission to perform such network tasks [12].

The first distinction is between Permissionless and Permissioned blockchains:

- Permissionless blockchains are systems in which anyone can participate in the verification process. No prior authorization is required, and a user can contribute his/her computational power, usually in return for a monetary reward;

- Permissioned blockchains are systems in which the miner nodes are preselected by a central authority or consortium.

For the second categorization we have:

- Public blockchains, in which anyone can read and submit transactions to the blockchain

- Private blockchains in which the permission of read/write is restricted to users within an organization or group of organizations

The distinction between public and private blockchain is related to who is allowed to participate in the network, execute the consensus protocol and maintain the shared ledger. One of the drawbacks is the openness of the public blockchain, which implies no privacy for transactions. A private blockchain network requires an invitation according to a set of rules put in place by an administrator. In a private blockchain, there are restrictions on who is allowed to participate in the network and transactions. A regulatory authority issues licenses for participation. Once an entity has joined the network, it will play a role in maintaining the blockchain in a decentralized manner.

In conclusion, it can be noticed that most permissionless blockchains feature public access, while most permissioned blockchains intend to restrict data access to the company or consortium of companies that operate the blockchain.

3.5 Consensus protocols

To ensure that only legitimate transactions are recorded into a blockchain, the network confirms that new transactions are valid, given the history of transactions recorded in previous blocks. A new block of data will be appended to the end of the blockchain only after the nodes on the network reach consensus as to the validity of all the transactions that constitute it. Thus the transaction only becomes valid ('confirmed') once it is included in a block and published to the network. In this way the blockchain protocols can ensure that transactions on a blockchain are valid and never recorded more than once, enabling people to coordinate individual transactions in a decentralized way, without the need to rely on a trusted authority [12]. The blockchain and its data exist in a peer-to-peer network, and as such, it is stored and extended by the nodes of the network, which form a topology between them. All the nodes, if they are not malicious and actively attempting to change the contents of the chain, contain the same blockchain structure and information, as they all agree on its contents through the consensus algorithm [13]. The main Consensus Protocols are described below.

3.5.1 Proof of work

PoW is the consensus used in the Bitcoin and Ethereum network [14]. To submit new transactions they are bundled together into a block and the miners calculates a hash value of the block header. Figure 3.4 depicts how Proof of Work (PoW) works. The block header contains a nonce and miners would change the nonce frequently to get different hash values. The consensus requires that the calculated value must be equal to or smaller than a certain given value ("target" value). When one node reaches the target value, it will broadcast the block to other nodes, and all other nodes must mutually confirm the correctness of the hash value. All the network miners compete to be the first to find the solution because the first miner who solves each blocks problem earn a reward. If the block is validated, other miners will append this new block to their blockchains.

In the decentralized network, valid blocks might be generated simultaneously when multiple nodes find the suitable nonce virtually at the same time. As a result, forks may be produced. However, it is unlikely that two competing forks

will generate the next block simultaneously. In the case of a fork, the rule is that the longest chain wins: after a certain period of time, the block that has the largest number of other blocks subsequently hooked is accepted. The other chains, with the other concurrently created blocks, are discarded, and their miners are not rewarded.

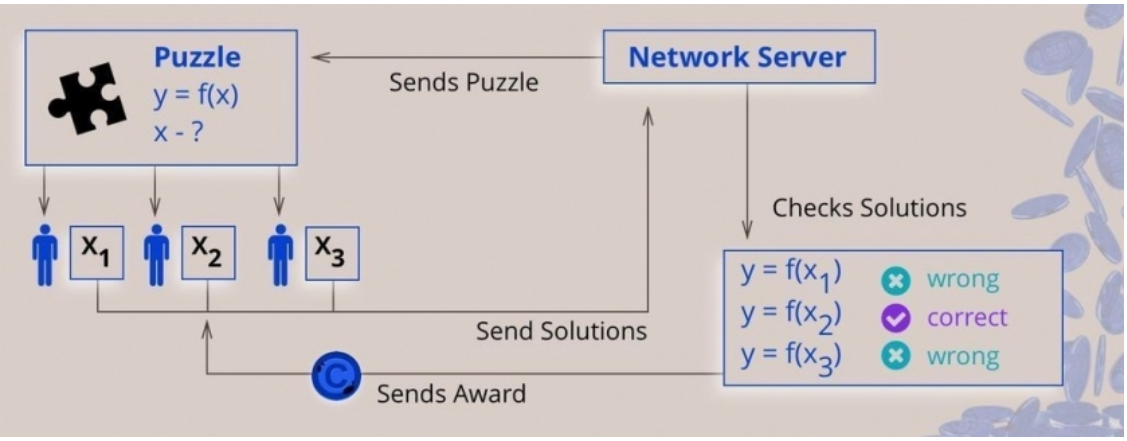


Figure 3.4: How PoW works [15]

3.5.2 Proof of stake

PoS is an energy-saving alternative to PoW [14]. Miners in PoS have to prove the ownership of the amount of currency, the "stake". It is believed that people with more currencies would be less likely to attack the network [16]. The selection based on account balance is quite unfair because the single richest person is bound to be dominant in the network. Therefore, the Proof of Stake attributes mining power to the proportion of coins held by a miner. This way a PoS miner is limited to mining a percentage of transactions that is reflective of his or her ownership stake. The creator of a new block is chosen in a semi-random, two-part process, depending on its wealth (stake). In the PoS system there is no block reward, so, the miners take only the transaction fees.

3.5.3 Practical Byzantine Fault Tolerance

PBFT is a replication algorithm to tolerate Byzantine faults [17]. The Byzantine Generals' Problem is a situation where involved parties must agree on a single strategy to avoid complete failure. However, it assumes some of the involved parties might be corrupt or otherwise unreliable [17]. The Practical Byzantine Fault Tolerance (PBFT) protocol guarantees the ability of a distributed computer network to correctly reach a sufficient consensus despite malicious nodes in the system failing or sending out incorrect information. The goal of PBFT is to protect against catastrophic system failures by reducing the influence of these malicious nodes.

In private networks, where the participants are whitelisted, costly consensus mechanisms such as proof-of-work are not needed, practically removing the need for an economic incentive for mining.

PBFT focuses on providing a practical Byzantine state machine replication that tolerates Byzantine faults (i.e., malicious nodes) by assuming there are independent

node failures and manipulated messages sent through specific nodes. Nodes in a PBFT system are sequentially ordered with one node being the leader, and others referred to as backup nodes. All nodes in the network communicate with one another with the goal being that all honest nodes will come to an agreement of the state of the system using a majority rule. Communication between nodes has two functions: nodes must prove that messages came from a specific peer node, and they must verify that the message was not modified during transmission. For the PBFT system to function, the number of malicious nodes must not equal or exceed one-third of all nodes in the system. The more nodes there are in a PBFT network, the more secure it becomes.

PBFT requires that every node is known to the network. The consensus rounds are four [18]:

1. A client sends a request to the leader node to invoke a service operation;
2. The leading node broadcasts the request to the backup nodes;
3. The nodes execute the request, then send a reply to the client;
4. The client awaits f+1 replies from different nodes with the same result, where f represents the maximum number of potentially faulty nodes.

In each phase, a node would enter the next phase if it has received votes from over 2/3 of all nodes. The leading node is changed during every view and can be replaced with a protocol called a view change if a certain amount of time has passed without the leading node broadcasting the request. Also, a supermajority of honest nodes can determine when a leader is faulty and replace them with the next leader in line.

Hyperledger Fabric uses the PBFT since it can handle up to 1/3 malicious byzantine replicas. Hyperledger Fabric is permissioned blockchain which is private and operated by invite with known identities which means that there is no need of algorithms like PoW.

3.5.4 Conclusions

Using consensus mechanism like PBFT, temporary forks in blockchain are not possible. In private blockchain which uses consensus algorithms like PBFT the problem of forks is avoided [13]. It is possible to highlight the main differences among the protocols. The first is the node identity management: PBFT needs to know the identity of each miner to select a primary in every round, while in PoW and PoS nodes could join the network freely. Moreover, it can be noticed that there is a big difference from the energy consumption point of view: in PoW, miners hash the block header continuously to reach the target value. As a result, the amount of electricity required to process has reached an immense scale [13].

Chapter 4

Blockchain: State of the Art

In this chapter, we analyze three different blockchain systems: Hyperledger Fabric, Ethereum, and Filecoin. We choose Hyperledger Fabric that implements a private blockchain, and Ethereum and Filecoin which implement a public blockchain. This chapter discusses the main characteristics of these three different systems, their architecture and their features from a security point of view. Section 4.1 explains Hyperledger Fabric, Section 4.2 describes the Ethereum platform, and Section 4.3 presents Filecoin. After that, Section 4.4 summarize the main characteristics of the analyzed systems and Section 4.5 explains the security issues which could exist in blockchains. Section 4.6 shows the main parameters used to evaluate the performance of a blockchain system. In Section 4.7 there is a comparison between Hyperledger and Ethereum both from the performance and security point of view, with a final discussion of the advantages and disadvantages of each approach. Finally, Section 4.8 explains why we chose Hyperledger Fabric as blockchain platform.

4.1 Hyperledger Fabric

Hyperledger is an open-source project hosted by The Linux Foundation, which aims to improve the concept of blockchain further in many different contexts [19]. The Hyperledger platform is an open source framework to build permissioned blockchains usable in business. Hyperledger Fabric can leverage consensus protocols that do not require a native cryptocurrency to implement costly mining. Avoidance of a cryptocurrency reduces some significant risk/attack vectors, and the absence of cryptographic mining operations means that the platform can be deployed with roughly the same operational cost as any other distributed system. Figure 4.1 depicts the Hyperledger Fabric network. The Hyperledger Fabric network is formed by a set of nodes. Nodes are the communication entities of the blockchain. Nodes are grouped in trust domains and associated to logical entities that control them.

There are three types of nodes:

1. **Client**: network entity that submits a transaction-invocation to the endorsers, and broadcasts transaction-proposals to the ordering service. It stands between the network and the end-user. It must connect to a Peer to communicate with the blockchain.

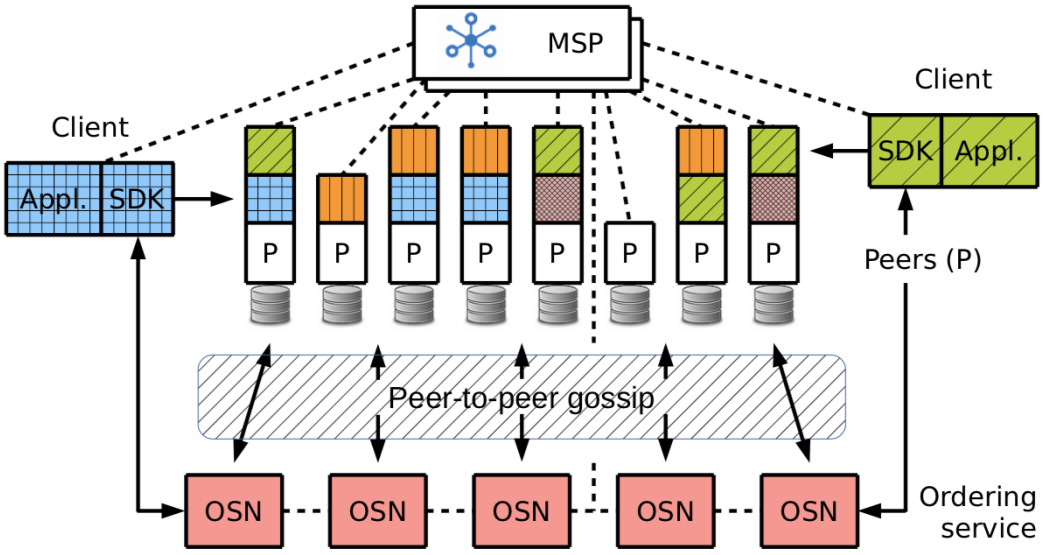


Figure 4.1: A complete Fabric network [20]

2. **Peer node**: network entity that maintains a ledger and runs chaincode containers to perform read/write operations to the ledger. Peers can be:

   - Endorsing Peer: it is a node that simulates smart contract transactions and returns a proposal response endorsement to the client. The endorsement policy specifies the set of Peers that need to simulate the transaction and endorse or digitally sign the execution results;
   - Committing Peer: it validates blocks of ordered transactions and appends the blocks to its local copy of the ledger;
   - Anchor Peer: it is the first Peer that will be discovered on the network by other organizations within a channel. Anchors Peers enable communication between Peers of different organizations and allow discovering all active participants of the channel;
   - Leading Peer: is a node which takes responsibility for distributing transactions from the Orderer to the other committing Peers in the organization.

   A Peer can be a committing Peer, endorsing Peer, leader Peer and anchor Peer all at the same time. Only the anchor Peer is optional. There must always be a leader Peer and at least one endorsing Peer and at least one committing Peer.

3. **Ordering Service node** or orderer: a node running the communication service that implements a delivery guarantee, such as atomic or total order broadcast. The Ordering Service ensures totally ordering of the delivered blocks on one channel. The hash chain imposes the total order of blocks in a ledger, and each block contains an array of totally ordered transactions. The Ordering Service provides a shared communication channel to clients and Peers, ordering a broadcast service for messages containing transactions.

Clients connect to the channel and may broadcast messages which are then delivered to all Peers.

The Ordering Service API consists of two basic asynchronous events:

- broadcast(blob): a client calls this function to broadcast an arbitrary message blob for dissemination over the channel;
- deliver(seqno, prevhash, blob): the ordering service calls this function on the Peer to deliver the message.

The ledger contains all data output by the ordering service. It is a sequence of deliver events, which form a hash chain according to the computation of the previous hash. The Ordering Service will group the blobs and output blocks within a single deliver event. The Ordering Service guarantees liveness delivery and safety consistency.

Two or more network members communicate through a *channel* which allows private and confidential transactions. A channel is defined by members, anchor Peers per member, shared ledger, chaincode applications and ordering service nodes. Each transaction on the network is executed on a channel, where each party must be authenticated and authorized to transact. Even if anchor Peers can belong to multiple channels and maintain multiple ledgers, no ledger data can pass from one channel to another.

4.1.1 Identity

Peers, Orderers, Client, and administrators are active network elements able to consume services. All these entities have a digital identity encapsulated in an X.509 digital certificate [21]. These identities determine the exact permissions over resources and access to information that actors have in a blockchain network. A digital identity also has some additional attributes used to determine permissions; the union of identity and attributes is called "principal". Principals can include a wide range of properties of an actor's identity, such as the actor's organization, organizational unit, role or even the actor's specific identity [22]. A verifiable identity must come from a trusted authority: the Membership Service Provider.

4.1.2 Membership Service Provider

The Membership Service Provider (MSP) identifies specific roles an actor might play within the scope of the organization the Membership Service Provider (MSP) represents (e.g., admins, or as members of a sub-organization group), and sets the basis for defining access privileges in the context of a network and channel (e.g., channel admins, readers, writers).

The Membership Service Provider identifies which Root CAs and Intermediate CAs are trusted to define the members of a trust domain, e.g., an organization, either by listing the identities of their members, or by identifying which CAs are authorized to issue valid identities for their members, or through a combination of both. An organization is a managed group of members and is often divided into multiple organizational units, each of which has a set of responsibilities [23].

The default MSP implementation in Fabric uses X.509 certificates as identities, adopting a traditional Public Key Infrastructure hierarchical model.

MSPs are mandatory at every level of administration: they must be defined for the network, channel, Peer, Orderer, and users.

- Network MSP: defines who are the members in the network as well as which of these members are authorized to perform administrative tasks (e.g., creating a channel);
- Channel MSP: define who can participate in certain activities on the channel;
- Peer MSP: is defined on the file system of each Peer and there is a single MSP instance for each Peer;
- Orderer MSP: is also defined in the file system of the node and only applies to that node. Orderers are owned by a single organization and therefore have a single MSP to list the actors or nodes it trusts.

4.1.3 Consensus

Hyperledger Fabric introduces a new architecture for transactions called execute-order-validate. Peers execute a transaction and check its correctness thereby endorsing it. Then, the Orderer Service order transactions via a consensus protocol and validate transactions against an application-specific endorsement policy before committing them to the ledger. The workflow of the consensus, shown in Figure 4.2, operates as follows:

1. the client transmits the transaction to the endorser nodes;
2. the endorser nodes simulate the transaction and choose whether they endorse it or not. If they do, they sign the transaction and send the endorsement back to the client;
3. the client broadcasts the endorsement to the Ordering Service;
4. the Ordering Service gathers incoming transactions, sorts them into blocks and then broadcasts the transactions by order to all the Peers (Orderers and committers). The Peers then validate the transactions and verify their endorsements, applying only the transactions which fulfill the endorsement policy.

4.1.4 Smart Contract

A smart contract, known in Fabric as chaincode, is code invoked by a client application external to the blockchain network. It manages access and modifications to a set of key-value pairs in the World State (the current values of all ledger states) [24]. The chaincode is installed into Peer nodes and instantiated to one or more channels. Users can use chaincode (for business rules) and membership service (for digital tokens) to design assets, as well as the logic that manages them. Fabric supports smart contracts written in general-purpose programming languages such as Java, Go and Node.js.

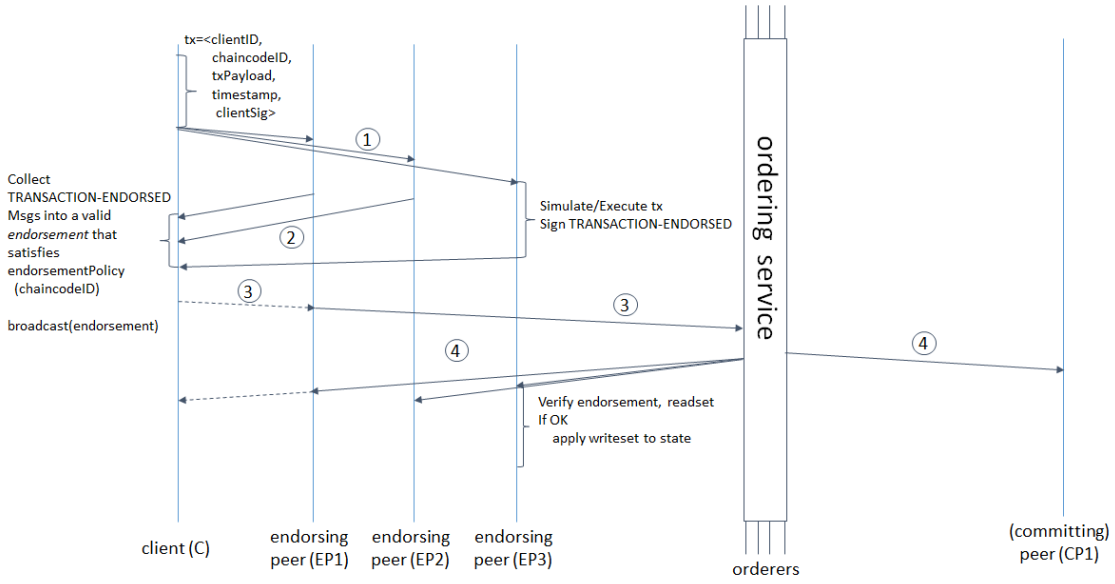


Figure 4.2: Transaction flow [23]

4.1.5 The ledger and the World State Database

The ledger is the sequenced, tamper-resistant record of all state transitions (see Figure 4.3). State transitions are a result of chaincode invocations (transactions) submitted by participating parties. Each transaction results in a set of asset key-value pairs that are committed to the ledger as creates, updates, or deletes. There is one ledger per channel. Each Peer maintains a copy of the ledger for each channel of which they are a member.

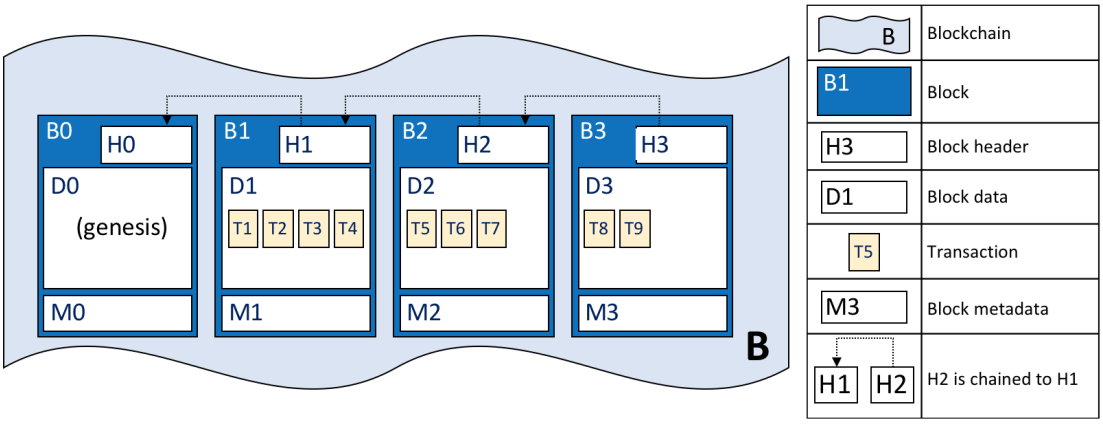


Figure 4.3: Ledger structure in Hyperledger [23]

Hyperledger Fabric has a ledger subsystem comprising two components: the World State and the transaction log. The World State component describes the state of the ledger at a given point in time, as depicted in Figure 4.4. It is the database of the ledger. The transaction log component records all transactions which have resulted in the current value of the World State. The ledger, then, is a combination of the world state database and the transaction log history. The ledger

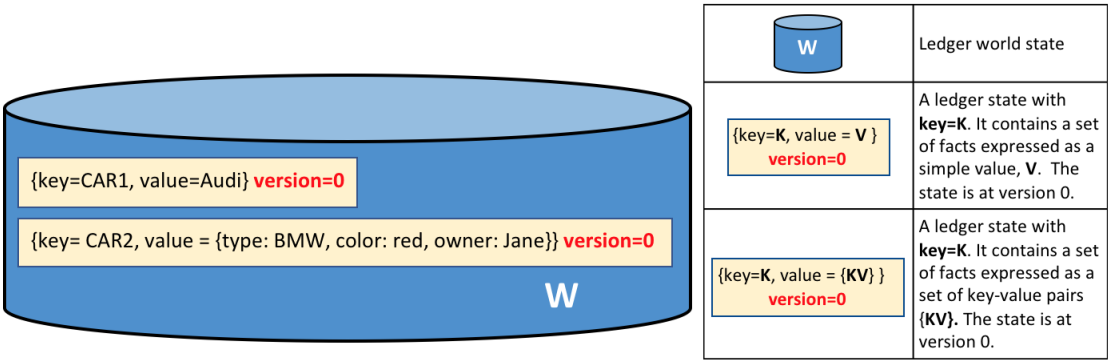


Figure 4.4: Example of world database in Hyperledger (the assets are cars)[23]

has a replaceable data store for the World State. By default, this is a LevelDB key-value store database. The transaction log does not need to be pluggable. It simply records the before and after values of the ledger database used by the blockchain network. The World State represents the current values of all ledger states; states have a key and a value. Physically, the World State is implemented as a database and provides a rich set of operators for the efficient storage and retrieval of states. In Hyperledger Fabric only transactions that are signed by a set of endorsing organizations will result in an update of the world state. The version number of a state is incremented every time the state changes. Because any transaction which represents a valid change to World State is recorded on the blockchain, it means that the World State can be re-generated from the blockchain at any time.

4.1.6 Security analysis

The *authentication* mechanism relies on digital signatures that allow a party to sign its messages digitally. Digital signatures also provide guarantees on the *integrity* of the signed message. Digital signature mechanisms require each party to hold two cryptographically connected keys: a public key that is made widely available and acts as authentication anchor, and a private key that is used to produce digital signatures on messages. Recipients of digitally signed messages can verify the origin and integrity of a received message by checking that the attached signature is valid under the public key of the sender.

In Hyperledger Fabric there is the possibility to create the signatures created with the Identity Mixer Protocol [25]. The Identity Mixer is a cryptographic protocol suite for privacy-preserving authentication and transfer of certified attributes. It allows user authentication without divulging any personal data. Thus, no personal data is collected that needs to be protected, managed, and treated according to complex legal regulations. Nevertheless, service providers can rest assured that their access restrictions are fully satisfied. These zero-knowledge proofs prove that the signature on some attributes is valid and the user owns the corresponding credential secret key. Only the user who knows the credential secret key can generate the proofs about the credential and attributes. This mechanism provides strong

authentication.

There are different *confidentiality* mechanisms to accommodate degrees of managing *privacy*, depending on the use case:

- Segregate the network into channels, where each channel represents a subset of participants that are authorized to see the data for the chaincodes deployed to that channel;
- Visibility setting: used to determine whether input and output chaincode data is included in the submitted transaction, versus just output data;
- Use private-data to keep ledger data private from other organizations on the channel. A private data collection allows a defined subset of organizations on a channel the ability to endorse, commit, or query private data without having to create a separate channel. Other participants on the channel receive only a hash of the data;
- Hash or encrypt the data in the client app before calling chaincode.
- Access Control into the chaincode logic to restrict data access
- Ledger data at rest can be encrypted via file system encryption on the Peer, and data in transit is encrypted via TLS;
- Use the Identity Mixer to provide no linkability. Using the default MSP implementation with X.509 certificates, all attributes have to be revealed to verify the certificate signature: this implies that all certificate usages for signing transactions are linkable. To avoid linkability, fresh X.509 certificates need to be used every time, which results in complex key management and communication and storage overhead. Identity Mixer avoids linkability for both the CA and verifiers since even the CA is not able to link presentation tokens to the original credential. Neither the CA, no a verifier can tell if two presentation tokens were derived from the same or two different credentials.

4.2 Ethereum (Public)

Ethereum is a blockchain platform to create decentralized applications. It is built on a public blockchain in which users, network nodes, currency, and markets are free to enjoy the network [26]. In Ethereum there are three types of nodes:

- Full nodes are entities that verify the blocks; they have the full copy of the blockchain;
- Light Nodes download block headers by default and verifies only a small portion of what needs to be verified;
- Miners are nodes that validate the new blocks by solving a crypto problem.

Full nodes verify block that is broadcast onto the network. They ensure that the transactions contained in the blocks, and the blocks themselves, follow the rules defined in the Ethereum specifications. They maintain the current state of

the network. Transactions and blocks that do not follow the rules are not used to determine the current state of the Ethereum network. For example, if A tries to send 100 ether to B but A has 0 Ethers, and a block includes this transaction, the full nodes will realize this does not follow the rules of Ethereum and reject that block as invalid. In particular, the execution of smart contracts is an example of a transaction. Whenever a smart contract is used in a transaction (e.g., sending ERC-20 tokens), all full nodes will have to run all the instructions to ensure that they arrive at the correct, agreed-upon next state of the blockchain. Full nodes that preserve the entire history of transactions are known as full archiving nodes.

Light nodes, in contrast, do not verify every block or transaction and may not have a copy of the current blockchain state. They rely on full nodes to provide them with missing detail. The advantage of light nodes is that they can get up and running much more quickly and can run on more computationally/memory constrained devices. Instead of downloading and storing the full chain and executing all of the transactions, light nodes download only the chain of headers, from the genesis block to the current head, without executing any transactions or retrieving any associated state. On the downside, there is an element of trust in other nodes (it varies based on client, and probabilistic methods/heuristics can be used to reduce risk). Some full clients include features to have faster syncs.

4.2.1 Accounts and Addresses

The global "shared-state" of Ethereum is comprised of many small objects ("accounts") that can interact with one another through a message-passing framework. Every account is defined by a pair of keys, a private key, and a public key. Accounts are indexed by their address which corresponds to the last 20 bytes of the public key.

Every User has an Account which can be Externally Owend account or Contract account. Externally Owned Accounts perform all the actions in the blockchain. They are controlled by private keys and have no code associated with them. Contract accounts are smart contracts, in fact, they are controlled by their contract code.

Any action that occurs on the Ethereum blockchain is always set in motion by transactions fired from externally controlled accounts. An externally owned account can send messages to other Externally Owned Accounts or other contract accounts by creating and signing a transaction using its private key; on the other hand, contract accounts cannot initiate new transactions on their own.

4.2.2 Transactions

The Ethereum blockchain has a native currency called Ether. One fundamental concept in Ethereum is the concept of fees. Every computation that occurs as a result of a transaction on the Ethereum network incurs a fee. For every executed operation there is a specified cost, expressed in a number of gas units.

Gas is the name for the execution fee that senders of transactions need to pay for every operation made on an Ethereum blockchain. Gas is purchased for Ether from the miners that execute the code. Gas and ether are decoupled deliberately since units of gas align with computation units having a natural cost, while the price of

ether generally fluctuates as a result of market forces. The price of gas is decided by the miners, who can refuse to process a transaction with a lower gas price than their minimum limit. The Ethereum protocol charges a fee per computational step that is executed in a contract or transaction to prevent deliberate attacks and abuse on the Ethereum network.

In the most basic sense, a transaction is a cryptographically signed piece of instruction that is generated by an externally owned account, serialized, and then submitted to the blockchain. With every transaction, a sender sets a gas limit and gas price. The product of gas price and gas limit represents the maximum amount of Wei that the sender is willing to pay for executing a transaction. The Wei is the base unit of Ether [27]. A Transaction contains the recipient of the message, the signature of the sender, the VALUE field which represents the amount of Wei to transfer, an optional data field, the "start gas" and "gas price". The start gas parameter represents the maximum number of computational steps the transaction execution is allowed to take, while the gas price represents the fee the sender is willing to pay for gas. The total ether cost of a transaction is: Total cost = gasUsed * gasPrice.

4.2.3 Smart contract

A contract is a collection of codes and data that resides at a specific address on the Ethereum blockchain. They are stored in a binary format: Ethereum Virtual Machine bytecode. The Ethereum Virtual Machine executes the contract code on each node participating in the network as part of their verification of new blocks. Contracts are typically written in Solidity and then compiled into bytecode to be uploaded on the blockchain. Contracts can send messages to other contracts which are virtual objects that are never serialized and exist only in the Ethereum execution environment. A message is like a transaction, except it is produced by a contract and not by an external actor.

4.2.4 Mining

Mining is the way to secure the network by creating, verifying, publishing and propagating blocks in the blockchain. Ethereum uses an incentive-driven model of security. The consensus is based on choosing the block with the highest total difficulty, which is used to enforce consistency in the time it takes to validate blocks. The difficulty parameter is a scalar value corresponding to the difficulty level applied during the nonce discovering of this block. It defines the mining target, which can be calculated from the previous block's difficulty level and the timestamp. The higher the difficulty, the statistically more calculations a miner must perform to discover a valid block. This value is used to control the block generation time of a blockchain, keeping the block generation frequency within a target range.

Miners produce blocks which the others check for validity. Among other well-formeness criteria, a block is only valid if it contains proof of work of a given difficulty. Generating this Proof of Work is known as mining. The Proof of Work consensus used is called *Ethash* and involves finding a nonce input so that the result is below a certain difficulty threshold. Any node participating in the network can

be a miner, and their expected revenue from mining will be directly proportional to their mining power or hash rate, i.e., the number of nonces tried per second normalized by the total hash rate of the network. Generating this proof of work does not itself require executing the transactions; miners can connect to a full node and get the work to be completed which is only dependent on the contents of the block to be added to the blockchain, not the execution of the contents. Each block represents a snapshot in time. Because there is often short-term disagreement (soft forks) as to what the current state of the blockchain is, after a transaction is confirmed in a block, it is often recommended to wait for a few more blocks to be built on top of that block before assuming that the transaction is final. The longest chain is the agreed upon state of the chain. The more blocks that have been added after a particular block, the less likely it is that the block will be "undone". The purpose of choosing the longest chain to add blocks to the blockchain is to arrive at the consensus. The reason why mining is necessary at the moment is so that there is a cost to extend the chain. Miners earn block rewards and transaction fees paid by people who wish to have their transactions added to the blockchain.

4.2.5 Security Analysis

Every account is defined by a pair of keys, a private key, and public key; *Authentication* and *Integrity* of messages rely on digital signatures. There is no *Confidentiality* and *Privacy*: data and contracts on the Ethereum network are encoded, but not encrypted. Everyone can audit the behavior of the contracts and the data sent to them. However, it always possible to encrypt data locally before broadcasting it to the network to preserve the confidentiality of data.

4.2.6 Ethereum Parity (Private)

Parity is an Ethereum client. Parity supports private chain and private network configuration via Chain specification files [28].

In addition to the Ethereum Public components the system contains:

- A public contract containing the encrypted code of a private contract;
- Secret Store nodes: used to create and provide access to the encryption key used to encrypt the state of the private contract and any message exchanged between the nodes. The exact Secret Store URL is specified for every Parity node participating in a private transaction system. A permissioning contract may be used to describes which account has access to which contract's key.

Parity supports state snapshotting which reduces sync-times and database pruning which reduces disk requirements. Snapshotting allows for a fast synchronization that skips almost all of the block processing, simply injecting the appropriate data directly into the database. It is also possible to prune the Parity's database to maintain only a small journal overlay reducing the disk space required.

Pluggable Consensus

Parity supports different consensus engines. The main one is the Ethash Proof of Work [29] but the most interesting is Proof-of-Authority consensus (e.g., Aura, in

which each validator gets an assigned time-slot, determined by the system clock of the validator, in which it can release a block). It can be used for private chain setups and uses a set of "authorities" which are nodes that are explicitly allowed to create new blocks and secure the blockchain. For each consensus engine, there are two main varieties of permissioned validation: Proof-of-Authority and Proof-of-Stake. In Proof-of-Authority, validators typically represent some real-world entities, which prevents Sybil attacks. These authorities can be added and removed according to a set of rules, such as via a voting process. The rules are specified in a smart contract on the blockchain. Proof-of-Stake, on the other hand, relies on security deposits. This means that validators are added after submitting a sufficient amount of valuable tokens, which can be taken away in the case of misbehavior.

The *validator set* is a group of accounts allowed to participate in the consensus; they validate the transactions and blocks. It can be defined statically in an immutable list or a contract. They validate the transactions and blocks to later sign messages about them. The chain has to be signed by the majority of authorities, in which case it becomes a part of the permanent record. This makes it easier to maintain a private chain and keep the block issuers accountable.

Private contracts and transactions

This feature allows storing encrypted data on the Ethereum blockchain. It is possible to create *private contracts* which are stored encrypted inside a public contract. The public contract specifies the accounts allowed reading and modifying the private contract's state. Only a validator account can allow changing the state of a private contract; the list of the validator is specified during the public contract's deployment. A private transaction is a message that contains encrypted data and can modify the state of the private contract; it requires the signature of all the validators.

Permissioning

Parity, differently to Ethereum, allows the network participants to different permission aspects of the blockchain. All permissioning is based on blockchain accounts, which means that permissions always correspond to an address. It is possible to introduce permissions at different layers such as:

- network: which nodes can connect to the network and communicate with the others;
- transaction: the ability to execute transaction types can be defined in a contract which implements a special interface;
- validation set: which parties can create new blocks and append them to the blockchain;
- gas price: it possible to create a whitelist of accounts for zero price transactions.

Security Analysis

The *Authentication* and *Integrity* are the same as in Ethereum. The main difference is about *Confidentiality* which is guaranteed by the encryption of Transactions and Contracts. There are still the same issues about Privacy as in Ethereum.

4.3 Filecoin

Filecoin is a decentralized storage network based on the blockchain. It is intended to be a cooperative digital storage and data retrieval method [30]. Filecoin's network is permissionless: it admits any aspiring participant without verification or centralized approval. It provides a system for guarantees based on the engineered incentives: Filecoin was designed with the goal that the most profitable choice of every participant is to act to improve the quality of service for the network.

4.3.1 The network and the participants

Filecoin is a decentralized storage network that turns cloud storage into an algorithmic market. The market runs on a blockchain with a native protocol token, which miners earn by providing storage to clients. The Filecoin Decentralized Storage Network (DSN) is auditable, publicly verifiable and designed on incentive. It aggregates storage offered by multiple independent storage providers and provides data storage and data retrieval to clients. Coordination is decentralized and does not require trusted parties: the secure operation of these systems is achieved through protocols that coordinate and verify operations carried out by individual parties.

Clients spend Filecoin hiring miners to store or distribute data. Filecoin miners compete to mine blocks with sizable rewards. Filecoin mining power is proportional to active storage, which directly provides a useful service to clients. This creates a powerful incentive for miners to amass as much storage as they can, and rent it out to clients. The protocol weaves these amassed resources into a self-healing storage network that anybody in the world can rely on. The network achieves robustness by replicating and dispersing content, while automatically detecting and repairing replica failures. Clients can select replication parameters to protect against different threat models.

4.3.2 Consensus

In the Filecoin protocol [30], storage providers must convince their clients that they stored the data they were paid to store; storage providers will generate Proofs-of-Storage that the blockchain network (or the clients themselves) verifies.

Proof-of-Storage schemes allow a user to check if a storage provider is storing the data at the time of the challenge. A verifier can check if a prover is storing her/his data for a range of time.

Proof-of-Replication is a Proof-of-Storage which allows the prover P to convince the verifier V that some data D has been replicated to its own uniquely dedicated physical storage. The prover P convinces the verifier V that P is indeed storing each of the replicas via a challenge/response protocol.

Proof-of-Spacetime is a scheme in which a verifier can check if a prover is storing her/his outsourced data for a range of time. The prover must generate sequential Proofs-of-Storage and has to compose the executions to generate short proofs recursively. The Filecoin protocol employs Proof-of-Spacetime to audit the storage offered by miners.

4.3.3 Participants

Any user can participate as a Client, a Storage Miner, and a Retrieval Miner:

- Clients pay to store data and to retrieve data in the Decentralized Storage Network;
- Storage Miners provide data storage to the network. Storage Miners participate in Filecoin by offering their disk space. To become Storage Miners, users must pledge their storage by depositing collateral proportional to it. Storage Miners store the client's data for a specified time. Storage Miners generate Proofs-of-Spacetime and submit them to the blockchain to prove to the network that they are storing the data through time. In case of invalid or missing proofs, Storage Miners are penalized and lose part of their collateral. Storage Miners can mine new blocks and receive the mining reward for creating a block and transaction fees for the transactions included in the block;
- Retrieval Miners provide data retrieval to the network. Retrieval Miners participate in Filecoin by serving data that users request via Get.

The Storage Market is a verifiable market which allows clients to request storage for their data and Storage Miners to offer their storage, and the Retrieval Market. The Retrieval Market enables clients to request retrieval of a specific piece and Retrieval Miners to serve it. Unlike Storage Miners, Retrieval Miners are not required to store pieces through time or generate Proofs of Storage. Any user in the network can become a Retrieval Miner by serving pieces in exchange for Filecoin tokens. Retrieval Miners can obtain pieces by receiving them directly from clients, by acquiring them from the Retrieval Market, or by storing them from being a Storage Miner. Clients and miners set the prices for the services they request or provide by submitting orders to the respective markets. The exchanges provide a way for clients and miners to see matching offers and initiate deals.

To request storage/retrieval of data in the markets and validate storage proofs, there are used the smart contracts, which enable users of Filecoin to write stateful programs that can spend tokens. Users can interact with the smart contracts by sending transactions to the ledger that trigger function calls in the contract.

4.3.4 Security Analysis

Filecoin blockchain provides *Integrity*: pieces are named after their cryptographic hash. After a request, clients only need to store this hash to retrieve the data and to verify the integrity of the content received. The *Authentication* is assured verifying that every request is signed by clients and miners. A digital signature might be required as part of Proof of Replication, to prove identity. Authentication can

help make schemes non-outsourceable since a prover would have to reveal secret identifying information (e.g., their private key) to the outsourced provider. The *Confidentiality* is not guaranteed; clients that desire for their data to be stored privately must encrypt their data before submitting them to the network. Moreover, the *Privacy* of the issuers of transactions is not guaranteed because their identities are public.

4.4 Summary

Given the discussion made so far, it is possible to summarize the main characteristics of the systems. Table 4.1 summarizes the features of the previous presented blockchain systems and Table 4.2 shows how the considered blockchain platforms implement the security properties described in Section 3.1.1.

Security property property	**Hyperledger Fabric**	**Ethereum (Public)**	**Ethereum (Private)**	**Filecoin**
Blockchain type	private	public	supports private chain and private network	public
Network nodes	Clients, Peers, Orderers	Full nodes, Light nodes, Miners	Full nodes, Light nodes, Miners	Clients, Storage miners, Retrieval miners
Consensus	PBFT	PoW	PoW, PoS, Proof of Authority	Proof of Replication, Proof of Spacetime
Native cryptocurrency	no	yes	yes	yes

Table 4.1: Main characteristics

Security property property	**Hyperledger Fabric**	**Ethereum (Public)**	**Ethereum (Private)**	**Filecoin**
Authentication	✓	✗	✗	✗
Privacy	✓	✗	✗	✗
Confidentiality	✗	✗	✓	✗
Integrity	✓	✓	✓	✓

Table 4.2: Security properties

4.5 Security risks

So far, blockchains have received great attention in different areas. However, there are some security problems and challenges. The following sections describe the main security risks of public and private blockchain such as respectively, Ethereum and Hyperledger Fabric.

A common issue of which the public blockchain system may suffer is the *Double spending* problem. It means spending the same money twice that can be possible because a digital token consists of a digital file that can be duplicated or falsified. Such double-spending leads to inflation by creating a new amount of fraudulent currency that did not previously exist [31]. Ethereum solves this issue adjusting the difficulty dynamically, in such a way that on average one block is produced by the entire network every 15 seconds [32].

Another known attack is *Sybil attack*, which is an attack against identity in which an individual entity masquerades as multiple simultaneous identities [33]. Ethereum manages this problem using the Proof of Work consensus. To create a new block on the network, a user needs a node with processing power. This attaches a significant cost to adding hundreds or thousands of pseudonymous nodes that might be able to influence the adoption of a fork or other blockchain vote.

The *Selfish mining* problem is an attack in which the selfish miner exploits the variance in block generation by partially withholding blocks. In Ethereum it could be possible to create a selfish attack, but the probability of success is very low.

There is also a problem related with the PoW consensus mechanism, used in most of the public blockchain platforms, which is the *Majority attack*, also called 51% attack. The blockchain relies on the distributed consensus mechanism to establish mutual trust. However, the consensus mechanism itself has 51% vulnerability, which can be exploited by attackers to control the entire blockchain. In PoW-based blockchains, if a single miner's hashing power accounts for more than 50% of the total hashing power of the entire blockchain, then the 51% attack may be launched.

As mentioned in Chapter 3, blockchain systems which use the PoW consensus, suffers the problem of *Versioning and Hard Forks*. Updates in the main protocols may lead to versioning or hard forks if all the blockchain community does not have the consensus of the new update.

Moreover, Ethereum avoids frivolous or malicious computational tasks, like DDoS attacks or infinite loops making users pay small transaction fees to the network.

On the other hand, private blockchains could have a security issue related to the PBFT consensus that is widely used in private blockchain systems. There is the threat of the *Control of 2/3rds of the validator set* [34]. A possible attack vector at this point for overtaking a permissioned blockchain is thieving (or brute forcing) of 2/3rds of the private keys for the validator set. The only attack vector on PBFT's safety is controlling a fraction of at least a third of all replicas. To prevent the adversary from controlling a big fraction of replicas, PBFT requires an additional protection mechanism. This mechanism can take various forms, such as an access control scheme based on a Certificate Authority, a mechanism able to identify Sybil identities, or requiring participants to dispose of important amounts of a scarce resource.

In a private environment there could also be a *Replay attack* during the endorsement of a transaction. In Hyperledger Fabric an attacker should not be able to replay blockchain transactions and affecting system state through transaction nonces. The endorsing Peers always verify that the transaction proposal has not been submitted already in the past, checking the nonce.

However, Hyperledger Fabric suffers from another threat. If a malicious user holds a valid certificate, he/she can mount a *Denial of Service* attack on the blockchain. On the other hand, Hyperledger Fabric is a permissioned blockchain platform. It means that the node of the system can be restricted to known identities. A party that mounts a DOS on the system can have their access revoked, and more importantly, since they are known identities rather than anonymous, can be held to account for their actions (or inaction, in the case they were hacked).

4.6 Performance

The blockchain system offers fault tolerance, data integrity, and authenticity. However, there are several parameters to take into account in the evaluation of the performance of a blockchain platform. The main factors to take into account are the following:

- scalability: time spent to put a transaction in the block and with the time necessary to reach a consensus. The restriction on scalability is due to the decentralized nature of the blockchain, in which every node on the network processes every transaction and maintains a copy of the entire state of the ledger;
- latency: the time required to submit a transaction;
- throughput: the number of transactions included in block per second;
- usability: different blockchains follow different programming languages for development; there is no standard available for developing the Application Programming Interface (API);
- block size: blockchain uses different block size.

4.7 Hyperledger Fabric vs Ethereum

Given the general discussion made in the previous section, it is possible to choose the blockchain platform which follows our requirements. The choice is between Hyperledger Fabric and Ethereum. Filecoin is still in the implementation phase, and Ethereum Parity is only an Ethereum Client, so even if it gives the possibility to setup private network, it shares the same issues with Ethereum. To better understand how to implement the blockchain network it is necessary to define the scenarios of our application. In the simplest scenario, there are N clients, belonging to one University, which uses the Crypto Cloud Application. While, in the extended scenario there are N clients, belonging to more than one University.

4.7.1 Ethereum blockchain network

Considering Ethereum, in the simplest scenario, there should be a client which is connected to a full node. The full node communicates with a miner node to append blocks to the blockchain. If there is more than one University, it will be necessary to run a node per provider. In Ethereum every operation has a gas price. Also for read-only operations of the state, which does not have network cost, gas price still is computed; it provides an upper bound to the complexity of the operation. Actual gas price is about 0.2 - 0.5 dollars to submit a transaction.

4.7.2 Hyperledger blockchain network

Using Hyperledger Fabric, in the simplest scenario, there will be one Organization (one University). An organization is a member who can join the network by adding its Membership Service Provider (MSP). Therefore, there must be an instance of the Membership Service Provider that defines how other members of the network may verify that signatures were generated by a valid identity of the organization. There must also be a Certificate Authority that creates X.509 certificates to feed the MSP configuration. The transaction endpoint of an Organization is a Peer. So it is necessary to have a Peer in our blockchain network. The Peer holds the chaincodes. There must also be an Ordering Service that creates the blocks and a Channel that allows communication between the entities. If there is more than one University, it will be possible to add an Organization. Therefore, if we have more than one University, there will be N Peers, N Channels, N Certificate Authorities where N is the number of Organizations.

4.7.3 Performance and Security

The Filecoin's network is still in the implementation phase [35], so we still have no information about the performance of the system. Therefore, we highlight the performance and security characteristics of Hyperledger Fabric and Ethereum, which are the candidate systems for this work. Table 4.3 shows the performance of both the system, highlighting the pros and cons and Table 4.4 shows their security issues.

4.8 Conclusions

From the analysis presented in Table 4.3, Hyperledger performs consistently better than Ethereum. Ethereum incurs large overhead in terms of memory and disk usage because of Proof of Work consensus protocol. From [19] Ethereum execution engine is also less efficient than that of Hyperledger, and the Hyperledger's data model is low level, and its flexibility enables customized optimization for analytical queries of the blockchain data. From Table 4.4 emerges that the Fabric platform suffers from fewer security issues than Ethereum mainly because of its permissioned nature. Given this analysis, of both the performance and security of the systems, it is possible to conclude that Hyperledger Fabric is the most suitable platform for this work. It is permissioned, guarantees privacy, and there is no economic cost on transactions and, generally, performs better than Ethereum.

	Ethereum	Hyperledger
Scalability	● High node scalability	● Solves performance scalability and privacy issues by permissioned mode of operation and fine-grained access control. Further, the modular architecture allows Hyperledger to be customized to a multitude of applications [36].
Throughput	● About 20 transactions per second	● About 3500 transaction per second
Latency	● 10 to 15 seconds	● Less then 1 second
Size	● To limit the size of the block it is used the gas limit	● The max size can be configured through the BatchTimeout and BatchSize [37] parameters. The default batch size is 98MB.
Energy consumption	● Ethereum is the second largest electricity consumer blockchain system (PoW consensus)	● Low waste of energy (PBFT consensus)
Usability	● Programmers must learn Solidity to write Smart Contracts	● Easy to program smart contracts (Go, Java, Node.js)

Table 4.3: Performance Pros and Cons

Ethereum	Hyperledger
● Does not provide any kind of privacy: everything is public (transactions, transactors)	● Strong authentication and authorization provided by the MSP. Privacy is preserved, providing no linkability on the transactions (Anonymous transactions).
● No confidentiality	● No confidentiality, but the data are accessible only inside the organization.
● High security based on economic incentive to mine and protect the integrity of data (PoW Consensus). Resilient to node failures.	● Nodes can be restricted to known identities. If a party mounts a DOS on the system, his/her access will be revoked.
● Forks on the Ethereum blockchain are possible	● Forks in blockchain are not possible.
● Sybil attacks are possible because authorization is only based on the possession of key pair (private, public key)	● Centralized identity management protects against Sybil attacks (MSP)
● Majority attack is possible (PoW Consensus)	● The network can be attacked if the malicious users are more than 1/3 (PBFT Consensus)
● PoW waste a lot of resources	● PBFT is less expensive than PoW from an energy point of view

Table 4.4: Security Pros and Cons

Chapter 5

Design

The goal of this thesis is to extend the Crypto Cloud Application with an adequate metadata management in order to ensure the security of sensitive operation without relying on the central server. Herein, we propose to achieve this goal with the use of a blockchain system. The blockchain technology guarantees integrity, immutability, freshness, and authenticity of the stored information, without relying on any third party. In Chapter 4, three different blockchain systems were analyzed: Hyperledger Fabric, Ethereum, and Filecoin. It discussed the main characteristics of each one, the pros and cons and why Hyperledger Fabric is the best system for this work. Its Fabric platform is modular, scalable, guarantees integrity, authentication, privacy and, in general, performs better than the other technologies. In Crypto Cloud, we have files and metadata associated with them. We could consider putting the files on the blockchain. In this case, we would need to change the whole system and removing the cloud part. However, traditional blockchain platforms are not meant for data storage but rather to store cryptographic proofs such that anyone retrieving the file can verify that the file is valid. Blockchain systems like Filecoin can provide file storage functionality, but it is still in the implementation phase. Therefore, we are not focusing on the files themselves but on the metadata. Crypto Cloud metadata are associated with users, clouds, files, and permission. Moving such metadata, we can eliminate the server but, or at least remove the assumption that it needs to be "honest but curious".

We start by presenting an overview of our proposed solution (Section 5.1). Then, we present the proposed architecture, detailing each of its components (Section 5.2). Following this, we describe the security models that are considered during the design phase (Section 5.3). Then, Section 5.4 provides the decisions about the metadata which will be on the blockchain. Section 5.5 details the new identity mechanism, Section 5.6 describes how to provide integrity to the Access Control List and Section 5.7 describes how to provide integrity to the files.

Section 5.8 discusses the advantages of passing from a database to the blockchain and what issues could be present if we open the CCDS database to simulate an "open" environment like a blockchain. We also compare the security issues which could exist in both the systems. Finally, in Section 5.9 there are the conclusions.

5.1 Overview

This section presents an overview of the proposed solution for a secure cloud system. We propose an alternative version of the Crypto Cloud application, which modifies some of the CCDS functionalities, integrating the system with the Hyperledger Fabric blockchain.

Crypto Cloud manages the identities relying on a key management server and a Public Key Infrastructure (PKI) infrastructure. In this way, it guarantees proper authentication while protecting and sharing sensitive files. Each of the Crypto Cloud's user holds a unique asymmetric key pair, which includes a private key and its corresponding public key. They protect (i.e., wrap) the symmetric keys used to secure the Crypto Cloud's files. This solution modifies the key management. We change the identity management to use the blockchain plus a control logic on the client. In the beginning, the client enrolls the user to the Fabric Certificate Authority. Then, the mapping between the username, Hyperledger identity, and Crypto Cloud identity is done to submit a transaction which contains that information. The transactions are signed with the Hyperledger users' private keys. We will call this type of transaction **Enrollment Transaction**. The critical metadata are the hash along with the version of the file and the Access Control List. Therefore, we will publish them in two types of transaction **File Metadata** transaction, which contains the metadata that guarantees the integrity of files, while **Share Metadata** transaction includes the hash of the Access Control List. In the resulting system, in every operation that deals with files, there will be the check of integrity between CCDS' metadata and the metadata stored on the blockchain. It results that, now, we do not need to trust the server anymore. Figure 5.1 represents the creation of the Enrollment transaction (1) when the user registers to the CC system, then the creation of a File Metadata transaction (2) and Share Metadata transaction (3.1) when the user uploads a new file and finally the creation of a Share Metadata transaction (3.2) when the user shares the file with other users.

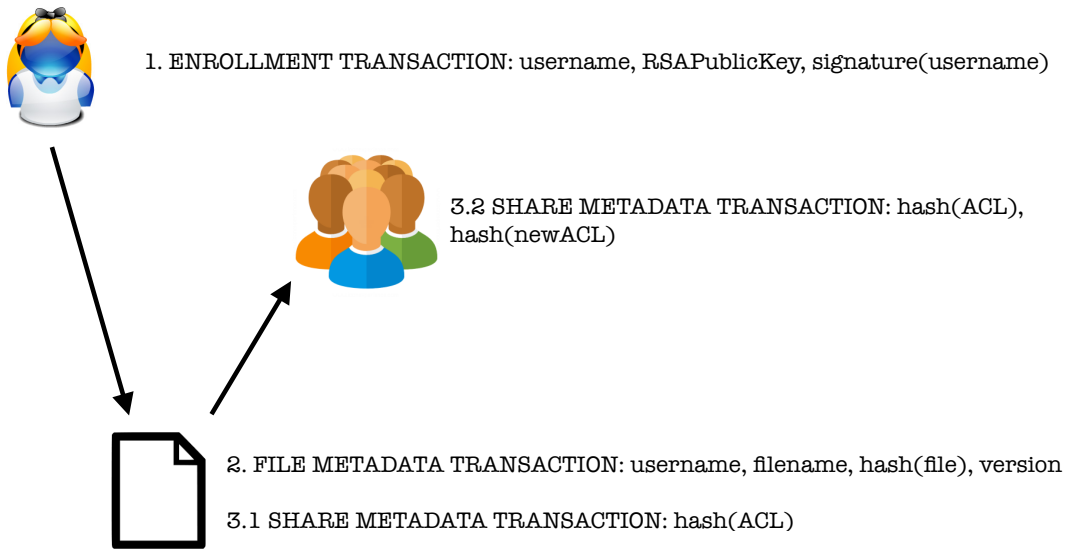


Figure 5.1: Crypto Cloud Transactions overview

Consequently, some of the algorithms are modified. In this solution, we redefine the basic algorithm, to provide integrity protection over metadata without relying on the CCDS. The hash and the version of the file are permanently memorized on the Hyperledger Fabric blockchain. In this way, it is possible to guarantee the integrity and freshness of that information. The sharing algorithm is modified with the check of the validity of the users' certificates before triggering the sharing mechanism, retrieving the identity of users from Fabric ledger. Moreover, to guarantee the validity of the ACL, we put a cryptographic proof of it, on the blockchain (a hash of the list). When the sharing mechanism is triggered, the list and its hash will be updated.

Briefly, the functionality of the server is still the same. The only difference is that we do not suppose that the server is "honest". So we must guarantee that only valid user can act on the system and that the files' metadata on the server is always valid. We can achieve these two goals with the blockchain.

5.2 Architecture

This section presents an overview of the new Crypto Cloud's architecture, illustrated in Figure 5.2. The system consists of four components: the client application, responsible for the system's main functionality. The Crypto Cloud Directory Server (CCDS), which manages the system's metadata, the Hyperledger Fabric network, and the Cloud Stores, which are responsible for providing users' storage space.

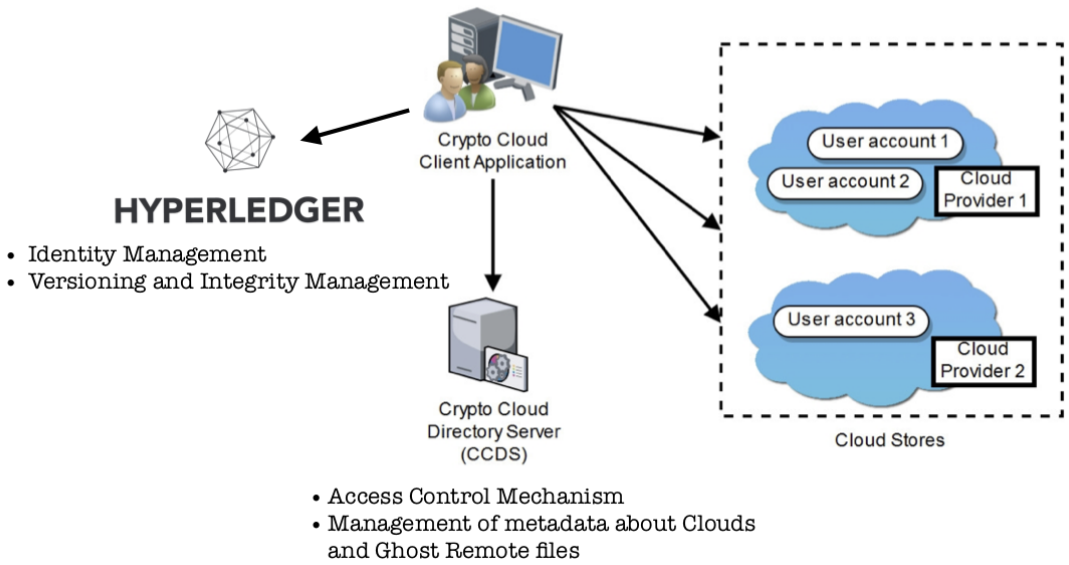


Figure 5.2: New Crypto Cloud Architecture

5.2.1 Crypto Cloud Client Application

The client application is the central component of the system. It is responsible for managing the user's files and interacting with all other Crypto Cloud's components.

The application allows authenticated users to upload and download files, as well as to manage their access permission. It is also possible to perform special operations, such as the generation of a replacement pair for the user's cryptographic key pair, or the request for registration of new users, requiring the approval from the administrators. In this new version, the client also communicates with the blockchain network. The communication is made using the Fabric API. This API is used to put and retrieve identities and files' metadata on the blockchain. Then, this information is retrieved and compared with the ones stored on the server. This mechanism is added to verify the consistency of the CCDS metadata. In this new version, the key pair is generated by the client and stored on its device. Therefore, the cryptographic operations are performed locally on the device. The other modules are equal to the previous version of the client.

5.2.2 Crypto Cloud Directory Server

The Crypto Cloud Directory Server serves the client application. Its functionalities remain the same as the previous version of Crypto Cloud. This component acts as a metadata repository, responsible for the system's metadata associated with users, files, shares, and clouds. Therefore, we can divide the metadata into five groups, based on their function.

The first group is formed by the metadata of the users, which contains:

- a unique username;
- a hash result of the user's password;
- the user's cryptographic information: the identifier of his/her private key, the user's public certificate, wrapped User Key, and User IV used for the cryptographic algorithm. The User Key is used to wrap and read-protect the access token provided by the cloud supplier from the CCDS;
- a boolean variable that represents the status of the account (enabled/disabled).

The second group includes information about the users' Cloud Stores. It is composed of:

- owner of the cloud (i.e., user);
- cloud supplier;
- workspace path for the cloud account;
- encrypted token used to access the cloud resources through the provider's API;
- info field, which contains the email associated with that cloud account.

The third group is formed by the metadata used to keep track of files that were deleted from the system but still occupy storage space in certain Cloud Stores ("ghost files"). It stores information about:

- file's remote name, which identifies a file inside the cloud space;
- cloud store where data are stored.

The fourth group includes the metadata used to implement the Access Control mechanism. The server stores:

- permission granted;
- wrapped read key, which is wrapped using the user's public key.

Finally, it possible to identify the fifth group: the metadata related to the managed files. It includes the file name, the current version, the share revision of the files, the cryptographic material, and the metadata to access files.

1. metadata to *access files*:
    - file key (KF): a symmetric key to encrypt the file, generated with AES-256 algorithm. Every " writes" on the file generates a new file key;
    - read key (KR): intermediate symmetric key, generated with AES-256 algorithm, used for encrypting the file key KF (to avoid file re-encryption). This key is then encrypted with the readers' public key and added to the ACL. Revocation of permission generates a new read key;
    - integrity Key (IK): used to calculate the Hash-based Message Authentication Code (HMAC) of the plaintext to guarantee integrity;
    - key pair (public key, private key), generate with RSA-2048;
    - HMAC: it is used to provide the integrity and authenticity of the exchanged message. The receiver performs the same MAC calculation over the received message and compares the result with the received MAC result. To calculate it, Crypto Cloud uses HMAC SHA-256 with AES-256;
    - contenthash: the hash result of the file's content. It is used to check the integrity of the data by comparing a stored hash value with the hash value of the actual data. It is generated with SHA512;
2. metadata to *address files*:
    - file's Uniform Resource Locator (URL);
    - file's remote-id;
    - file's remote name.

    These three parameters are acquired through the providers' APIs and allow the client application to read, update and delete files from Cloud Stores.

5.2.3 Hyperledger Fabric Network

Our Hyperledger network is composed of Organizations, Ordering Service, channels, Certificate Authorities, smart contracts, and Peers. The implementation details will be explained in Chapter 6. Every entity in the network refers to the Membership Service Provider, which provides cryptographic keys and certificates. The client application can submit a transaction-invocation to the endorsers, and broadcasts transaction-proposals to the ordering service. Therefore, we create an API that allows communicating with the Hyperledger Fabric network. The API provides the necessary methods to enroll the administrator and users to the Certificate Authorities and generates and submits transactions to the ledger.

5.3 Models

This section describes the models considered during the designing of our solution. We start by defining the solution's trust model (Section 5.3.1), where we state the considered assumptions. Then, we identify possible threats and vulnerabilities of our solution (Section 5.3.2).

5.3.1 Trust model

Each component described in the system's architecture runs in distinct environments: the client application executes in the user's device, the CCDS and the Hyperledger Fabric network run in the host's organization. To better understand these different environments and how these components should work, it is necessary to establish a satisfactory trust model. The trust model helps us recognize the special characteristics of the system and how the various entities are expected to behave. The following analyzes the assumptions taken into account to define the trust model of the proposed solution:

- the client's device is trustworthy: the software and hardware where the client application runs is reliable and not compromised;
- the CCDS can act maliciously: the CCDS can listen to the exchanged messages and can modify the data;
- the Cloud Providers can act maliciously: the cloud environment can have malicious insiders, so potentially the providers can act maliciously and try to read or change the content of hosted files;
- the private keys are securely stored and cannot be manipulated by agents external to the client application: these keys can only be accessed by their owners, and they cannot use them outside of the Crypto Cloud's client application;
- the cryptographic algorithms used are sound: the used cryptographic algorithms are considered secure and not broken.
- the filenames and the usernames are unique in the entire system

5.3.2 Threat model

A threat model identifies potential threats and vulnerabilities of the system. Its analysis provides a way to design better defenses against attackers and possible countermeasures, increasing the security of the system.

- Spoofing Identity:
  1. an attacker may try to replace a user's certificate with his certificate;
  2. a malicious user or application may provide fake information to persuade users to consider him as a trustworthy certification entity.
- Tampering with Data:
  1. a malicious cloud provider may modify the users' files.
- Repudiation:
  1. users can claim that they did not modify a file from the system.
- Information Disclosure:
  1. a malicious user can use the information on the blockchain to modify the files.
- Denial of Service:
  1. a malicious user can re-upload old versions of files to destroy recent updates;
  2. a malicious user can spam repeated requests to degrade the service.
- Elevation of Privilege:
  1. a malicious user can perform unauthorized operations (e.g., delete) on resources (e.g., files, keys or others) that are owned by other users.

5.4 Metadata on the blockchain

This Section provides the decision and motivation about what metadata can be stored on the blockchain, to avoid the necessity of trust the CCDS. As discussed in the previous Chapter 2, the main functions of the Crypto Cloud Directory Server are the identity management, which relies on the KMIP service plus PKI infrastructure, the metadata management, the Access Control Model based on authentication and the version control and integrity of files. In Figure 5.3 there is an overview of all the metadata. The ones in bold are the candidate metadata to be moved on the Hyperledger blockchain.

GROUP 1	GROUP 2	GROUP 3	GROUP 4	GROUP 5
Unique username	Encrypted token	File remote name	User	**Original filename**
Hash(password)	Info	File's cloud	File	**Hash of the file content**
Identifier of private key	Type		Permission	**Current version**
Public certificate	Workspace		**List of authorized users**	Share revision
Wrapped UK	Owner			URL
User IV				Remote ID
				File remote name
				Wrapped FK
				Wrapped IK
				IVs used

Figure 5.3: CCDS Metadata

We do not want to trust the server, so we should put the metadata associated with each functionality both on the blockchain and the CCDS. In addition, it is also necessary to add a control logic on the client to verify the authenticity of what the server holds.

Crypto Cloud keeps the identities of users to assure their validities when accessing the managed files. Replacing the management means eliminating the KMS server and the external PKI infrastructure. The identity in Crypto Cloud corresponds to the public key from **Group 1**. In Hyperledger Fabric the Membership Service Provider is responsible for performing the authentication of the users who participate in the system, creating an identity for each user. Fabric maintains its Certificate Authorities and PKI infrastructure. Putting the Crypto Cloud identity of users on the Fabric ledger, it is possible to modify the first CCDS functionality as will be discussed in Section 5.5.

Group 2 contains the information about the Cloud Stores so moving it on the blockchain can lead to an exposure of data. It provides data such as the email associated with the cloud account or the workspace path of the account.

Group 3 contains the metadata used to keep track of files that were deleted from the system but still occupy storage space in certain Cloud Store. We are not interested in this mechanism.

Group 4 metadata is used to manage the access control on resources. These metadata, as said before, have this structure: (resource - user - list of authorized users). It derives that, if we do not trust the server, the list of the authorized users could be modified by a malicious user. An attacker who does not have the right permission could add users to the list. Therefore, we should guarantee that the server always holds a valid version of that metadata that means to assure its integrity. We calculate the hash of the list and put it on the blockchain. When a Crypto Cloud user operates, the client verifies that the hash of the server list is the

same as the one retrieved from the blockchain. When a user executes the "share" operation, the client creates a new transaction with both the hash of the old ACL and the hash of the new ACL. This mechanism is explained in detail in Section 5.6.

Group 5 includes metadata associated with the managed files. We could think of moving that metadata on the blockchain. The file content hash and version are used to provide version control and integrity of files. Exposing them on the blockchain, we do not risk any attack because that metadata does not contain any sensitive information. When the client operates, it checks if the metadata maintained by the server is the same as the metadata on the blockchain. In this way, we guarantee their integrity without losing confidentiality. Instead, the confidentiality of the metadata used to address the file would not be preserved. This is because the metadata used to address the files, lead to files' link; in Hyperledger even though data are protected across channels, in the channel itself the ledger is public. A malicious user who holds a valid Hyperledger identity could read these metadata. If they send many read requests, it will lead to a DoS attack. So we do not put the fileId, file URL and others on the blockchain. So we have to focus on the crypto material. Section 5.7 details the resulting mechanism.

5.5 Management of identities

A key concept in the Crypto Cloud system is the identity. The identity is the mapping between a User and his/her public key. The Crypto Cloud application uses a Key Management Server and a Public Key Infrastructure (PKI). These components are responsible for implementing the Key Management Interoperability Protocol (KMIP) standard, which allows the remote access and management of user's cryptographic keys, and certification of the users' identity, guaranteeing proper authentication while protecting and sharing sensitive files. So, the KMIP service generates the RSA key pairs: a public key and a private key. The system uses the generated keys to encrypt/decrypt the other cryptographic keys. To guarantee proper certification of the users' keys and to ensure their identity when accessing the managed files, the system relies on a Public Key Infrastructure, which acts as a trusted third-party entity. There is the concept of identity also in Hyperledger Fabric; every member of the network has its own identity represented by the entity's public key. The Fabric system generates the key pair using the elliptic curve cryptography. These keys are used to sign and verify transactions. In the blockchain everything is public, signed and stored in an immutable way. Therefore, we could develop a mechanism to manage the identities taking advantage of the Fabric platform. In this way, we can avoid the use of the KMIP and PKI external service. Moreover, adding an application logic on the client, it is always possible to verify the validity of identities. It results that we can follow two different approaches:

1. The key pair used in Crypto Cloud is the same used in the Hyperledger Fabric;

2. The Crypto Cloud identity differs from the Hyperledger Fabric identity.

By choosing the first strategy, we leave the responsibility for generating cryptographic keys to the Hyperledger Fabric. Therefore, the client only updates its local state with this informations. Using the same key pair, we could explicitly publish the public key when a user enrolls in the system. In this way, the identity would be explicitly associated with every action (transaction) performed by a specific user. Another way could be based on the principle that in the blockchain the transactions are signed, so implicitly the user who sends the transaction provides information about his/her identity. The signature is done with the private key of a user and is checked with the associated public key. We can assume that the first transaction a user makes in the blockchain shows the identity. In this way, it is possible to avoid publishing the public key on the blockchain. If a user wanted to find another Fabric's identity, it would only be necessary to contact the membership service provider [23].

Using the same key pair is not the best approach for our system because, as already mentioned, the current version of Crypto Cloud uses RSA key pair to wrap symmetric keys, but Hyperledger Fabric generates the cryptographic key pair with EC cryptographic algorithm [38]. Furthermore, using the same keys for signing and encrypting is considered bad practice: it is better to avoid it.

Therefore, we have a problem of incompatibility of keys which we overcome by following the second approach: we use two different identities for each user. Using two different key pairs, we need to be sure that every file-related transaction is executed from an authentic identity. It is reasonable to publish the public key of users on the blockchain: in this way, everyone can verify that a user has that specific identity. No one can change the content of a transaction published on the blockchain, so the identity and public key relationship are permanently registered and accessible by everyone. When a user performs the first registration to Crypto Cloud, a transaction will be created that contains the binding between Crypto Cloud identity and Hyperledger identity. However, this is not enough; a malicious user could publish a transaction, which represents his identity, in which he puts the public key of another valid user. He can now retrieve and access transactions related to files without the permission to do so. To avoid this problem, we also put the signature of the transactor, done with the RSA Private Key of Crypto Cloud. Briefly, when a user enrolls to the system, the client creates an Enrollment transaction that contains {username, RSAPublicKey, signature(username)}. Every time a user reads transactions from the blockchain, the client verifies that the signature in the transaction is valid.

It is possible to summarize the user enrollment operation in the following steps:

1. the client generates the key RSA key pair locally: a private key and a public key. The public key is the Crypto Cloud identity;
2. the user submits a transaction which contains: {username, RSAPublicKey, signature(username)};
3. the user's Crypto Cloud identity is now associated with the Hyperledger Fabric identity because the transaction is signed with the Hyperledger private key.

When file-related transactions are performed, the client will verify that the identity of the transactor is valid. It is assumed that the user's identity corresponds to the

first transaction on the blockchain corresponding to a specific username. It follows that when a user wants to read his/her files, the client:

1. sends a request to Hyperledger Fabric using the filename;
2. retrieve the username of who made this transaction; then, sends a request to Fabric to obtain his/her Enrollment transaction;
3. obtains the public key of the user and the signature of the username;
4. verifies the signature using the public key;
5. verifies if the public key on the server corresponds to the public key retrieved from the blockchain;
6. verifies if the ACL of that file contains the public key.

Each transaction is then validated so even if a malicious user holds the public key of a valid user, he/she will not be able to verify the signature because he/she does not hold the private key of the valid user. Furthermore, even if the public keys are the same, the Hyperledger identities will be different.

5.5.1 Attack example

We are now going to describe a possible attack on the system and how we solve this problem. Assume to have a trusted user Alice and a malicious user Eve. Suppose Eve holds Alice's public key and the Read Key which is necessary to access the file. Alice is already enrolled in Crypto Cloud and Hyperledger with her public key. Eve registers to Hyperledger Fabric using the public key of Alice. On Hyperledger blockchain now we have two Enrollment transaction, as depicted in Figure 5.4.

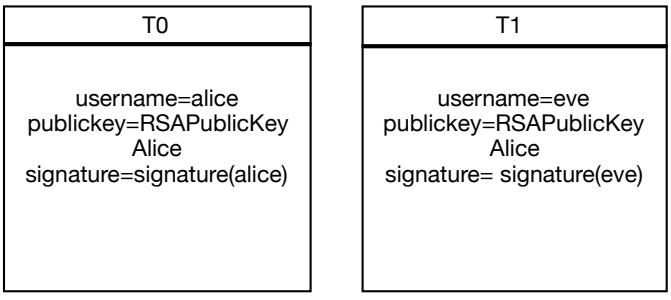


Figure 5.4: Alice and Eve identities stored on the blockchain

Alice creates a file `file.txt`. Eve has access to the file but she is not on the list of allowed users. Eve wants to upload the file, so she submits a new transaction which contains the new information of `file.txt`. Now we have two File Metadata transactions on the blockchain, represented in the Figure 5.5.

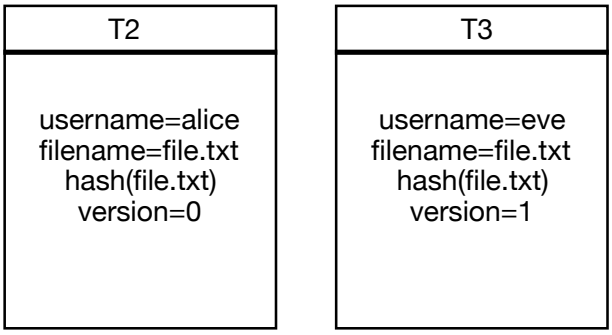


Figure 5.5: Transactions related with `file.txt`

When Alice or other users, who have access to the file, want to read it, they:

1. retrieve the last transaction referred to filename=`file.txt` from the blockchain T3;
2. find that Eve created it;
3. find the identity of Eve from the blockchain, obtaining the content of transaction T1;
4. verify the signature on transaction T1 using the correspondent public key;
5. the signature is not verified because Eve does not hold the private key corresponding to Alice's public key.

So this transaction is considered invalid, and it is ignored. The logic of the application continues finding the previous version of the transaction which is considered valid. The previous transaction about file `file.txt` is T2; then the client applies the same "algorithm" mentioned before, and it finds that this transaction is valid: the signature of Alice is verified.

5.6 Access Control List Integrity

Another function of the CCDS is Access Control. The server maintains metadata which allows doing an access control over files. For each file of each user there is a list of authorized users who can access/write/share the file: $\{KR\}_{PublicKey1}$, $\{KR\}_{PublicKey2}$,..., $\{KR\}_{PublicKeyN}$. The CCDS access control follows a simple logic: when the server receives a request, it checks if the sender is on the list and if he/she has the permission to operate. For example, if the operation is *share*, the server checks if the user has "share" authorization and if he/she is on the list of allowed users. If the check succeeds, the user will be able to update the list. The same checks are done in case of file update; a user can update a file if he/she has write permission or is the owner and if he/she is on the list of authorized users.

If we do not want to trust the server, we must guarantee the integrity of the Access Control List. This is necessary because we have to ensure that users can not update it, adding entries of malicious users, if they are not on the list. When the file is created, the list of identities who can access the file is generated. The

problem arises when a user has the Read Key (KR), so he/she can access the file, but he/she is not in the list of authorized users. Crypto Cloud assumes that the server is honest but curious. If we do not trust the server anymore, it could tamper with the Access Control List. Therefore he could add an entry corresponding to $\{KR\}_{PublicKeyServer}$. Now it can modify the list, adding users even if it does not have the permission to do that. So it is necessary to check if the list is also valid and not just the file.

To solve this problem, we calculate the hash of the access list and publish it on the blockchain. Every time a user reads a file, the client calculates the hash of the ACL retrieved from the server and check if it is equal to the one retrieved from the blockchain. In this way, integrity is always confirmed. We must update the ACL when a user performs a share operation. To do this, we put both the hash of the old version of the list and the hash of the new version. In this way, we can always verify the validity of the old version and create a correspondence with the actual one. In other words, when a user adds a new file, the client creates a Share Metadata transaction that contains: {hash(ACL)}. When a user updates an existing file, the client calls the Fabric API to create a new Share Metadata transaction that contains {hash(ACL), hash(newACL)}.

To summarize, the share operation involves the following steps:

- clients have access to the information of all the state;
- clients obtain the last version of the transaction related to the file on which the user is interested;
- the identity of the transactor is checked, retrieving the identity transaction from the blockchain;
- clients verify the integrity of the ACL retrieved from the server checking it with the one obtained from the blockchain;
- the identity of the grantee user is obtained from the blockchain;
- if the check succeeds and the client is on the list of allowed users, a new transaction is submitted with: $hash(\text{ACL})_{oldversion}$, $hash(\text{ACL})_{newversion}$.

As an example, assume that Alice wants to share `file.txt` with Bob.

1. Alice's client reads the most recent version of the blockchain. It looks to the state of the blockchain and finds the last valid transaction related to `file.txt`;
2. the client retrieves the identity of Alice from the blockchain and verifies it;
3. the client calculates the hash of $\{KR\}_{PublicKeyAlice}$ retrieved from the server and compares it with the hash(ACL) in the transaction;
4. checks if the ACL related with `file.txt` contains Alice's public key;
5. if the check succeeds, the client queries the blockchain to obtain Bob's public key;

6. updates the list. The resulting list will be: $\{KR\}_{PublicKeyAlice}$, $\{KR\}_{PublicKeyBob}$;

7. submits a file transaction to Hyperledger Fabric which contains hash($\{KR\}_{PublicKeyAlice}$), hash($\{KR\}_{PublicKeyAlice}$, $\{KR\}_{PublicKeyBob}$); send the request of sharing to the CCDS and changes the local client state.

When Bob accesses the Crypto Cloud system, the client retrieves `file.txt` and verifies the information contained in the last transaction of `file.txt`.

It is possible to optimize these operations by having clients cache of the verified identities. Every time it is necessary to check the validity of an identity, the client looks into the local state, otherwise contacts the blockchain to obtain the Enrollment transaction associated with the user.

5.7 Version control and integrity

The Crypto Cloud Directory Server (CCDS) also manages integrity and versioning of files. It guarantees the integrity of the files using three metadata values: the content hash of the file, the version, and HMAC. The file content hash represents the hash result of the content of the file in plaintext. This result is calculated by the client application and sent to the CCDS. A file version is a number managed by the server, which is incremented every time a user performs an update operation on the file. The association of both these elements ensures that a particular file's content represents a specific version. To avoid that an attacker or a malicious cloud provider can change the uploaded file's content without a user noticing it, Crypto Cloud used the HMAC using a symmetric key called Integrity Key; it is calculated over the content of the plaintext and produces a fixed-size hash result. The generated result is stored concatenated with plaintext alongside the ciphered content. When a user tries to read a file, the application performs the reverse operation. Then, the result of this operation is compared with the retrieved result from the previous one. If both results match, the file is untouched. Otherwise, it means that the content of the file was tampered and compromised.

In Crypto Cloud, there is the assumption that the CCDS is trustworthy which means that it is expected that the server acts honestly, but there is the possibility that a malicious user can tamper with the metadata in case of attack. So in the actual system using the HMAC combined with hash and version of the file, there is the guarantee of the integrity and versioning of files. In the blockchain, it is not necessary to adopt that mechanism because everything is published on the ledger is signed, and no one can change the content of transactions. Managing the HMAC will result in a more complex operation because we have to manage also the symmetric key associated with it, so it is not useful. In fact, given the discussion made in Chapter 3, one of the main characteristics of the blockchain is the guarantee of integrity. It is possible to put the content hash of the file and version on a transaction, gaining the certainty that no one can change its content once published. Therefore every time the user wants to read a file, the application checks the correspondence between CCDS' metadata and File Metadata transaction. In this way, we are sure that what is on the server is valid.

To summarize, when a user reads his/her file, the client checks:

1. if the hash of the file on the CCDS is equal to the hash of the file on the blockchain

2. if the version of the file on CCDS is equal to the version on the blockchain

3. if the hash of the ACL got from the server is equal to the hash of the ACL from the blockchain

If all the checks pass, we obtain the last version of files, with the guarantee of freshness and integrity.

5.8 From a centralized system to the blockchain

The following section discusses the advantages of moving from a central database to the blockchain. A database is a collection of data, running on a secure server in a central location, with limited user access. It is a central ledger entity that needs the trust of the administrator to manage it. On the other hand, the blockchain is a disintermediating technology, where each transaction is cryptographically signed, and always appended to an immutable ledger; it is visible to all members and distributed across the network [39]. Moreover while in a traditional database a client can perform four functions on data such as create, read, update, and delete, the blockchain is designed to be an append-only structure. A user can only add more data, in the form of additional blocks. All previous data are permanently stored and cannot be altered. Because traditional databases are centralized in nature, their maintenance is easy, and their performance is high, but this brings in the drawback of the risk of the single point of failure which is represented by the central server. Anybody with sufficient access to a centralized database could destroy or corrupt the data within it.

To fully understand the advantages of using a blockchain system instead of a traditional database, it is possible to imagine breaking the access of the database of the Crypto Cloud Directory Server, as depicted in Figure 2.3. Let us analyze the main issues which are possible if we open that database; open meaning that everyone could access and write data.

The cryptographic material, such as keys, HMAC and other, will be more exposed but, if that data is encrypted with robust algorithms, it will not be an issue. Only the user who holds the private key associated with the public one can access it. Therefore the confidentiality of these metadata will be preserved. While, if a malicious user held the file identifier (fileID), they would access the file, because the fileID leads to the recent version of the file link. So the fileID leads to the URL which could lead to a Denial of Service attack; a malicious attacker could send many read requests, making the file inaccessible to the valid users. Generally, a DoS attack will happen when a malicious user can spam repeated requests to degrade the service.

If the database is accessible to everyone, the system will lose the Access Control mechanism. If the write operation was available for all the users of the system, a malicious user could perform unauthorized actions on resources (files, keys or others) that are owned by other users. If a malicious user held the fileID, he could access it, performs updates and sharing operations because there is no access control

on the activities. Therefore we have the problem of tampering with data in which an attacker could modify users' files. There is also the problem about versioning, such as a malicious user can re-upload old versions of records to destroy recent updates which can lead to a DoS attack. An attacker could also change the link from the correct file to another one. There are mechanisms to protect the files integrity, such as content hash, version, and HMAC. However, a malicious user could change the link to the valid file, also changing the hash and the version, and upload a new version of HMAC with a new IK, and no one will be able to notice it.

With a private blockchain, it is possible to avoid the loss of authentication. In Hyperledger Fabric there is the Membership Service Provider which is responsible for performing the authentication of the users who participate in the system, and using the mechanism explained in Section 5.5 we guarantee the validity of the identities. Verifying the identity of who made the transactions, we can check if the user is not allowed to make that requests. If his/her public key is associated with an invalid identity, he/she cannot perform such operations because he/she is not able to verify the signature with the correspondent public key. The integrity and freshness of files are provided with the mechanism described in Section 5.7. DoS attacks are possible when users with valid certificates send many read requests, but this is always possible, so we have to live with that. Finally, adopting the blockchain system, we can still have the Access Control Model mechanism as discussed previously.

After the discussion, it is possible to summarise the security threats of traditional database and blockchain in Table 5.1.

Security issue	**Open database**	**Private Blockchain**
DoS attack	Yes, possible	Yes, limited
Access Control Model	No	Yes
Replay attack	No	Yes
Authentication	No	Yes
Tampering message	No	Yes
Confidentiality of metadata to address the file	No	No
Confidentiality of metadata to read the file	Yes	Yes
Privacy	No	Yes

Table 5.1: Security issues in open database vs private blockchain

5.9 Summary

In this chapter we introduced a new version of the Crypto Cloud system, which integrates Crypto Cloud [1] with the Hyperledger Fabric blockchain system.

In the proposed solution the assumption that the server is honest but curious is overcome, so we do not need to trust the server anymore. We introduced the Fabric network in the CC architecture, and use it to provide metadata integrity. In

this way, we are sure that even in case of an attack, the server must provide valid metadata. Otherwise, the attack is detected.

The new Crypto Cloud system provides users' identity management and assures the validity of metadata without relying on the CCDS. The Enrollment transactions represent the identity of the users. File Metadata transactions contain the files' metadata. Finally, Share Metadata transactions store the hash of ACLs on the blockchain.

Enrollment transactions contain the user's public key and its signature. The first time a user registers to the system, an Enrollment transaction is created. Every time the user issues an operation, the client checks the signature, verifies if the user's public key on the server corresponds to the public key on the transaction and checks if that public key is also in the ACL. In this way, the validity of the users' metadata held by the server is guaranteed.

File Metadata transactions contain the metadata that represents the "integrity", namely: the file: the file content hash and its version. Every time a user wants to read his/her files, the client verifies, for each file, the correspondence between the server's metadata and the blockchain metadata. In doing so, the client will always be capable of detecting if the metadata was tampered with.

Share Metadata transactions represents the integrity of the Access Control List. Every time a user creates a file, a new transaction is submitted, containing the hash of the ACL. Moreover, when users issue a "share" operation, clients submit a new Share Metadata transaction that includes the hash of both the old version of the ACL and the updated one. When clients read files, they verify the integrity of the server's ACL by referring to the Share Metadata transactions.

Chapter 6

Implementation

This chapter describes the implementation of the proposed Crypto Cloud system with Hyperledger Fabric. In the the implementation we consider a simplified version of the design, not considering the Access Control List integrity because of time constraints. The focus of this work is not to implement all the functionalities explained in the previous Chapter 5 but to create a Proof of Concept used to demonstrate the feasibility and practical potential of the proposed approach.

In Section 6.1, we present an overview of the developed implementation. Section 6.2 describes the structure of the Hyperledger Fabric network, necessary to create the blockchain infrastructure. Section 6.3 details the application interface developed to communicate with the Fabric network. Section 6.4 details the changes in the protocols responsible for the most relevant Crypto Cloud's operations. Finally, Section 6.5 concludes this chapter, summarizing all aspects of the implementation.

6.1 Overview

This section presents an overview view of what was implemented. We start by defining the scenarios in which the application can be executed. In the simplest scenario, there are N clients, belonging to one University, which uses the Crypto Cloud Application and communicate with the Crypto Cloud Directory Server. The University represents our Hyperledger Fabric Organization, so we need a ledger that contains all the transactions related with the users' metadata. While, in the extended scenario there are N clients, belonging to more than one University, which use the Crypto Cloud Application. Adding Universities means adding Organizations, so we will have more than one Peer and more than one ledger. The Anchor Peers enable the communication between different Organizations and they hold the copy of the ledgers of the connected Organizations. It is possible to suppose for simplicity that we are in the first scenario.

The first implementation step is the creation of the Hyperledger Fabric network. A Fabric permissioned blockchain network is a technical infrastructure that provides ledger services to application consumers and administrators. A network consists of ledgers, one per channel comprised of the blockchain and the state database, smart contracts, peer nodes, ordering services, channels and Fabric Certificate Authorities. In the simplest scenario we build an infrastructure which allows creating a complete blockchain environment and test its potentiality. Therefore, we create a Peer, an

Ordering Node and a Certificate Authority. If another University uses the Crypto Cloud Application, it will be necessary to add a Peer per University. To implement the functionalities described in Chapter 5 two chaincodes are necessary. They represent identities and files' metadata and provide functionality to access and write them on the Fabric ledger.

Once the network is created, it is necessary to develop a program which allows interacting with the Fabric entities. We implement a basic Fabric API, which communicates with the Fabric infrastructure. It is written in Java, as the Crypto Cloud Prototype, and makes available all the operations which are possible in the Fabric blockchain. Afterwards, it is possible to modify the Crypto Cloud Client in order to communicate with the Fabric API. Figure 6.1 depcits the resulting Hyperledger Fabric infrastructure.

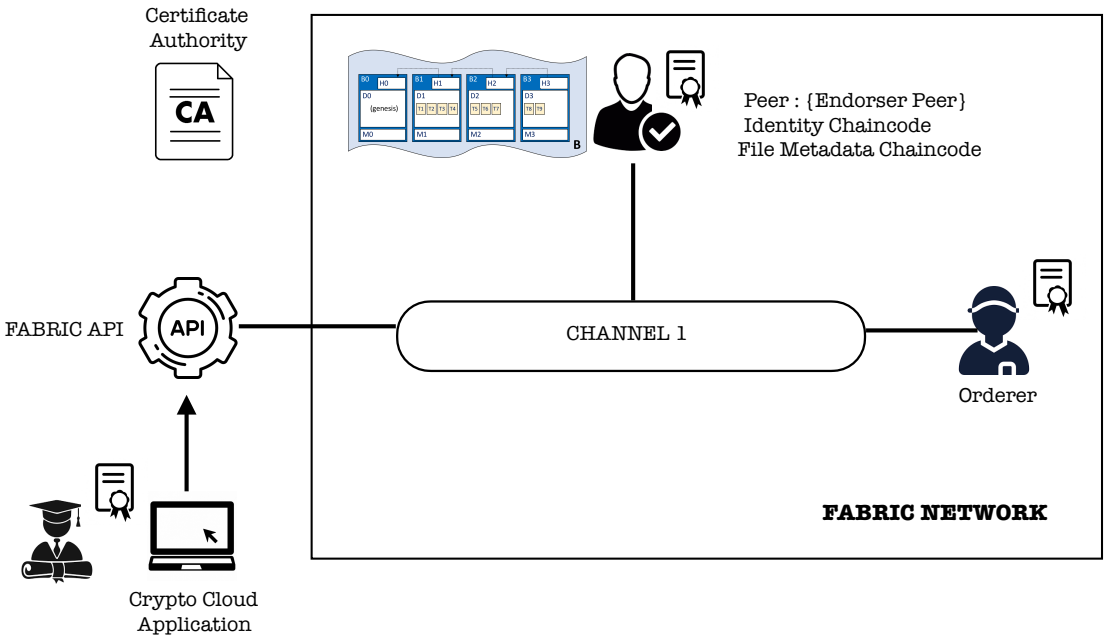


Figure 6.1: Crypto Cloud Fabric Network

6.2 Create the Hyperledger Network

This section describes the environment necessary to implement a Fabric blockchain system. To create the network, it is necessary to define validating peers, the ordering service, organizations and generate certificates and keys. The basic pattern provides a Hyperledger Fabric network which consists of one organization (in this work our University), one Peer Node, a Certificate Authority, and a solo Ordering Service. The Peer node is responsible for performing validation of the transaction and submits them to the Orderer node. The Orderer sorts the transaction and creates blocks. Then, it sends the blocks to the Peer which will add them to the ledger. Every entity in the Fabric network must be enrolled to the Membership Service Provider which provides the certificates and key necessary to be identified in the network. Only valid entities can act into the network where valid means

identities with a certificate issued by a Certificate Authority of the Fabric PKI. Fabric provides the `cryptogen` tool which generates the cryptographic material such as X-509 certificates and signing keys which define identity of the users. Every Organization has one root certificate which allows the binding between Peers, Orderers, and Organizations. The transactions are signed with the Private Key of the user and they are verified with the corresponding public key. It also provides the `configtxgen` tool which creates the orderer genesis block, the channel configuration transaction and the anchor peer transactions, one for each peer organization. It consumes a file which contains the definition of the network.

Every entity can be run either natively on the operative system or can be created through Docker containers. In Fabric, Docker consumes the docker-compose.yaml file which defines the blockchain network topology. We deploy the network with Docker which facilitates the management of the different entities needed to create the network and because there is a detailed explanation on the official Hyperledger Documentation [23]. Once the network is created, it is possible to create and initialize the channel, create the chaincode, register and enroll the users, invoke and query to test the network. We create one channel, which allows the communication between the Peer, the Orderer, the Certification Authority and the client app.

6.2.1 Chaincode

To obtain the desired functionality, two smart contracts must be implemented to encode the rules of the system. The chaincode is used to specify assets, as well as the logic that manages them. We want to represent two assets: identities and files' metadata. Towards this, we decided to develop the chaincode using the Go language, referring to the examples on the documentation [23]. Every chaincode program must implement the "Chaincode" interface [24] whose methods are called as a response to the received transactions. In particular, the `init` method is called when a chaincode receives an instantiate or upgrade transaction so that it may perform any necessary initialization, including initialization of application state. The `invoke` method is called in response to receiving an invoke transaction to process transaction proposals. The other interface implemented in the chaincode is the ChaincodeStubInterface [24] which is used to access and modify the ledger, and to make invocations between chaincodes. We developed two chaincodes:

1. Identity Chaincode: put/get identities;

2. File Metadata Chaincode: put/get metadata of the managed files.

In the first, we defined the structure *Identity* which contains the username, the RSA Crypto Cloud public key and the signature of the username, done with the RSA private key of the CC user. It provides two functions: create identity function and query identity function which allows submitting transactions. When creating a new identity, it is necessary to put an entry, an identity, on the World State database. Then, it can be regenerated from the chain at any time. The state database will automatically get recovered (or generated if needed) upon peer startup before transactions are accepted. For the key of the state database, it was decided to use the username to identify the transactions uniquely. To retrieve an identity,

the Fabric API provides a function that allows obtaining the value of the specified asset key (the key used is the username).

In the second chaincode the *FileMetadata* structure is defined, which is formed by the file name, the content hash, the version of the file and the username of the transactor. To submit transactions the chaincode provides the `invoke` method which is called as a result of an application request to run the smart contract. When creating a new identity, it is necessary to put an entry on the state database corresponding to the file's metadata. We use as the key of the state database the filename, in order to uniquelyidentify the transaction related with the file, since the filename is unique in the whole system (see the assumptions on Section 5.3.1). The query function allows retrieving the value of the specified filename.

After the smart contracts have been developed, an administrator in the Organization must install it onto the Peer node. After it has occurred, the Peer has full knowledge of the chaincode. However, the other components connected to the channel are unaware of it, so it must be instantiated on the channel. After instantiation, every component on the channel is aware of the existence of the chaincode, meaning that the client application can now invoke it.

6.2.2 Dimensions analysis

As discussed, two types of transaction can be sumbitted on the blockchain: Enrollment transactions and File Metadata transactions. It is possible to analyze the dimension of these two transactions to give an idea about the dimension of what we put onto the blockchain. The File Metadata transaction is represented in Table 6.1 and, regardless of the uploaded size, contains about 2KB. The Enrollment transaction metadata, represented in Table 6.2, takes less than 1KB per transaction.

Metadata	**Dimension**
Username	size in bytes of String
Filename	size in bytes of String
Content Hash	64 bytes
Version	1 byte
TOTAL	<1KB

Table 6.1: File Metadata Transaction

Metadata	**Dimension**
Username	size in bytes of String
User's public key	2048 bytes
Signature of user public key	64 bytes
TOTAL	about 2KB

Table 6.2: Enrollment Transaction

6.3 Fabric API

Once the Hyperledger network is created, we need to develop a client node which can communicate with it. The Hyperledger Fabric makes available a Fabric SDK Java to facilitate Java applications to manage the lifecycle of Hyperledger channels and user chaincode. The SDK provides a means to execute user chaincode, query blocks and transactions on the channel, and monitor events. Our Fabric API allows interacting with a Certificate Authority and generating enrollment certificates. These identities are used to sign and verify transactions. When we launched our network, an admin user was registered with our Certificate Authority. We then use this admin object to subsequently register and enroll new users. Then, the program will invoke a certificate signing request and ultimately output an Enrollment Certificate (eCert) and key material. Our application will then look to this location when they need to create or load the identity objects for our various users. These identities will be used when a user wants to query and update the ledger.

As said before, we can issue Enrollment transaction and File Metadata transaction. So the Fabric API must provide the methods which allows to create an identity and create a new file metadata transaction.

Finally, the Fabric API provides queries to retrieve data from the ledger. The data are stored as a series of key-value pairs, and it is possible to query for the value of a single key, multiple keys, or, if the ledger is written in a rich data storage format like JSON, perform complex searches against it (looking for all assets that contain certain keywords, for example). Every query is signed with Hyperledger identity which submits the request.

6.4 Modified protocols

This section describes the implementation details of the protocols responsible for performing the Crypto Cloud's operations which interact with the blockchain.

In the implementation of the Crypto Cloud application, the communication with the KMIP client is made on the client side. We want to substitute the management of the identities, so it makes sense to add the communication with the Hyperledger Fabric network on the client. Moreover, when all the features mentioned in the previous chapter will be moved on the blockchain, it will be possible to shutdown the server. The client module provided an API capable of interacting with a service endpoint using the KMIP protocol which is used to manage the cryptographic keys. So the first class changed was the implementation of the KeyManager. The RSA key pair generation is now done locally rather than using the KMIP mechanism.

The system must provide a mechanism to validate the public keys and associated identities of users. When a new user joins the system, it is necessary to generate the keys which represents his/her identity. As discussed in the previous chapter, we cannot use the identity generated by Hyperledger without changing the whole key management; so we use two different identities. In our solution, when a user wants to subscribe to the system, the client registers the user to the CCDS and enrolls him/her to Hyperledger Fabric Membership Service Provider. The subscription to the Certificate Authority returns the Hyperledger Fabric identity of the user. Then, the application creates a transaction which contains the Crypto Cloud identity

calling the corresponding function of the Fabric API; it contains the username, the Crypto Cloud RSA public key of the user and a signature of the username made with the RSA private key. This guarantees the binding between Hyperledger identity and Crypto Cloud identity. Thanks to the assumption on Section 5.3.1, we can say that the identity of a specific user U is stored on the first occurrence of his username in the transactions. If another user subscribes with the same username, it will not be considered.

The read operation consists of updating a local file with the most recent version available from the cloud stores. During this operation, the Client Application obtains the file's metadata from the CCDS using the file's identifier. Moreover, it retrieves the hash and the version querying the File Metadata Chaincode through the Fabric API. Upon receiving the file's metadata both from the server and the blockchain, the Client Application checks if the file exists in the local workspace or if it is obsolete. If it is obsolete, the application accesses the file's URL to retrieve its content. The client receives the file and unwraps the file's keys using the user's private key and deciphers its content. The client must check if the information on the server corresponds to the ones on the retrieved transaction. Therefore, it checks the identity of the user who creates/update the file: it invokes the query identity function of the Identity Chaincode, passing the username of who made the transaction related with the file. Then, it retrieves the public key of the user and verifies the signature contained in the transaction with the associated public key. The client compares the hash of the file and the version obtained from the server with the one retrieved from Hyperledger. If the controls succeeded, the client verifies if the public key of the user is equal to the one on the server and if it is on the list of the authorized users. This means that it can read the file.

The write protocol consists of uploading a local copy of a file to a registered cloud store. To create a new file in the Crypto Cloud system, the Client Application generates the file's keys and IVs and performs the cryptographic operations, guaranteeing the confidentiality and integrity of the file. After that, the protected file is uploaded to the cloud store using the Cloud Provider's API. After uploading, the Client Application sends a request to the CCDS with the new file's metadata and a request to the blockchain using our Fabric API. We invoke the create file function, passing as arguments the username, the file name, the public key, the hash and the version of the file. The CCDS verifies if the file is unique and does not conflict with other files in the user's workspace. After registering the new file, the CCDS answers the client with the new file's id, version number, and share revision. The Client Application stores the received metadata in the user's session.

To perform a file update, the Client Application requests the CCDS for the file's metadata using the file's id and checks if the user maintains the most recent updated version of the file. It also requests the last transaction related to the file calling the File Metadata chaincode. Then, it verifies its validity checking the corresponding identity retrieved from the blockchain. Now the client verifies if the hash of the retrieved file is equal to the hash of the file on the blockchain, if the public key is the same as the one on the server and if the user is on the access list. Finally, the Client Application renews the file's keys and submits a new transaction which contains the update information of the file.

The share operation allows a user to share a specific file with another user, called grantee user. When performing this operation, the user queries the blockchain,

calling the Fabric API, and retrieves the last transaction related with the desired file; if the identity of the transactor is already verified it is not necessary to redo the verifications, otherwise the identity is retrieved from Hyperledger calling query identity function; if the signature is verifiable with the corresponding public key the operation continues. Now the client verifies if the hash of the retrieved file is equal to the hash of the file on the blockchain and if the public key of the user is on the access list. Then, the client finds the grantee user's public key from the blockchain calling the query identity function of Fabric API. Then, it adds the user to the access list, adding $\{KR\}_{PublicKeyUser}$ and send a request to the CCDS with the updated information. In this operation the control of the ACL integrity is missing; it should be a mechanism which checks and updates the list of authorized users every time a sharing operation is issued. This mechanism is described in Section 5.6.

6.4.1 Example of basic algorithm

This Section explains the basic algorithm implemented by the system. The functional process involves the management and protection of users' files. We consider only read and write operations, and not the sharing operation. Alice registers into the system, so the client enrolls Alice to the CCDS (1) and to Hyperledger blockchain through the Fabric API (3).

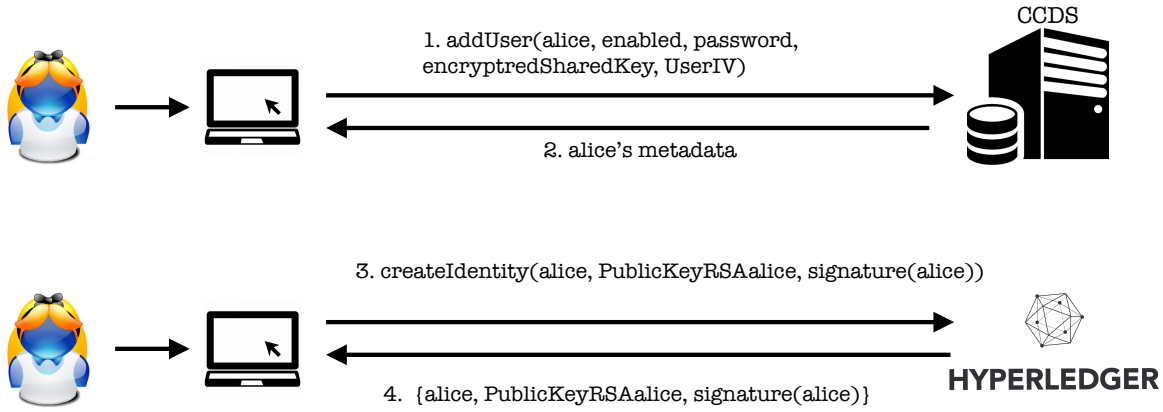


Figure 6.2: Alice's registration

Alice upload a file `file.txt`. Alice's client uploads the new file's metadata both on the server (5) and on Hyperledger Fabric (7).

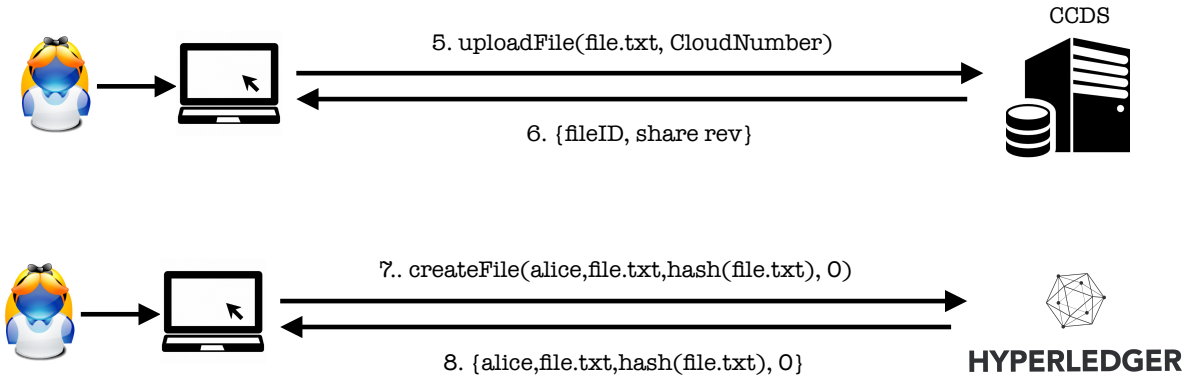
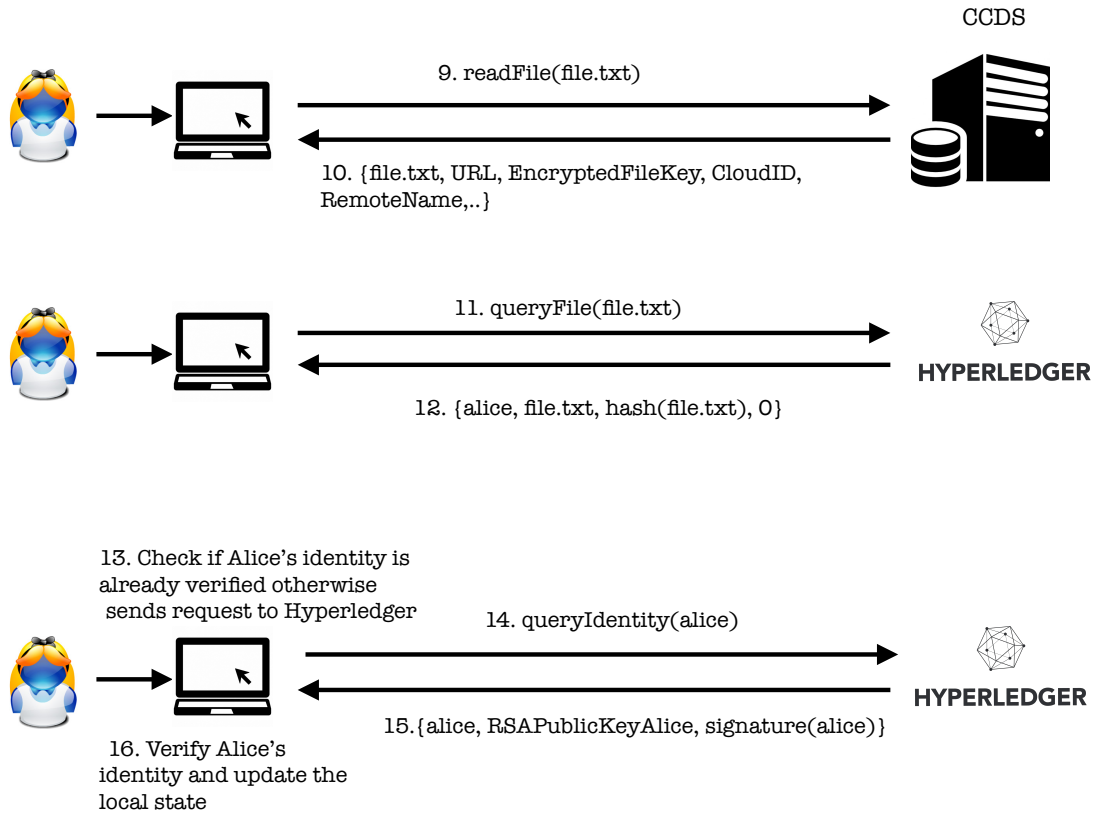


Figure 6.3: Uploading of the file

Alice retrieves her files (9). For each file, Alice's client retrieve the File Metadata transaction associated with the file (11) and retrieve the identity of the transactor using the username contained in (12). The client verifies that the files are uploaded by valid users. This is done verifying the signature of the identity retrieved from Hyperledger (16), checking if the user public key on the CCDS is the same as the identity transaction (19) and if it is in the ACL (22). In this way authenticity of transactions is guaranteed. Therefore, Alice reads the files (in this case there is only `file.txt`). The client checks the integrity of the metadata retrieved from the CCDS (23, 24).



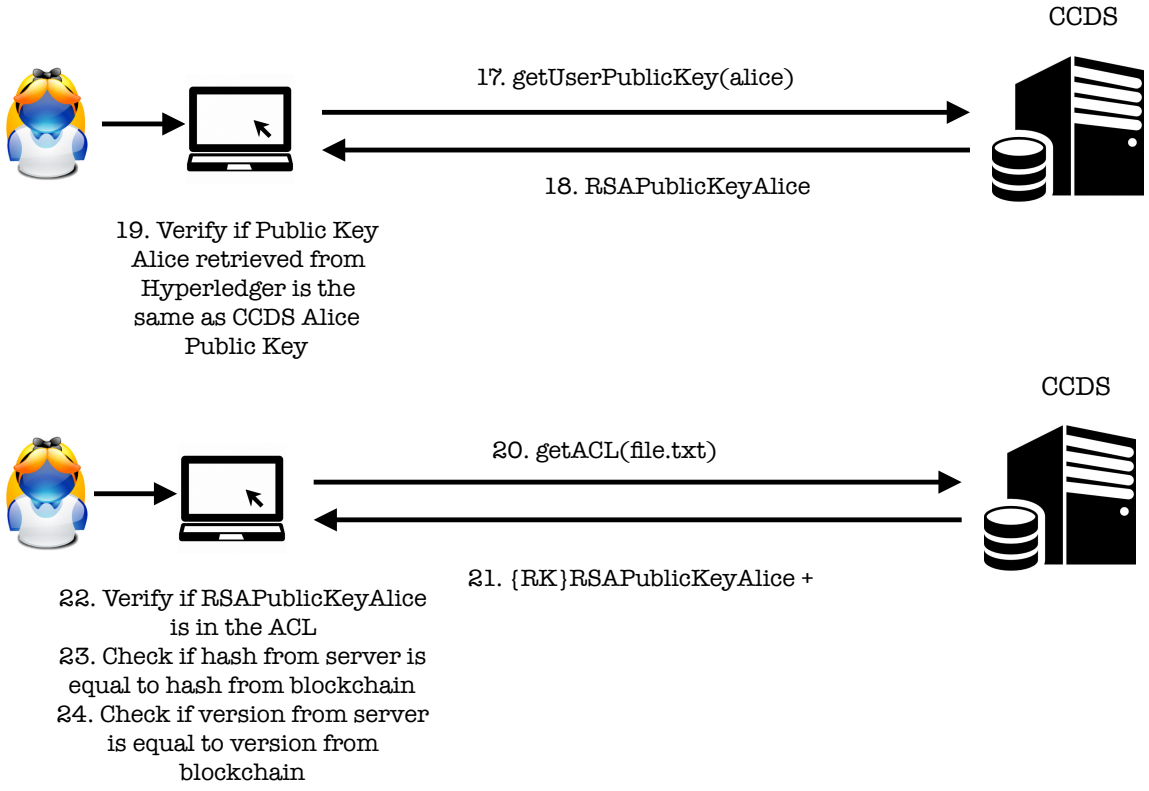


Figure 6.4: Alice's client reads files

6.4.2 Example of sharing operation

This Section explains an example of share operation implemented by the system. The share operation allows a user to share a certain file with another user, called grantee user. Assuming Alice is already enrolled to the system. Bob signs up to the system (1), (3).

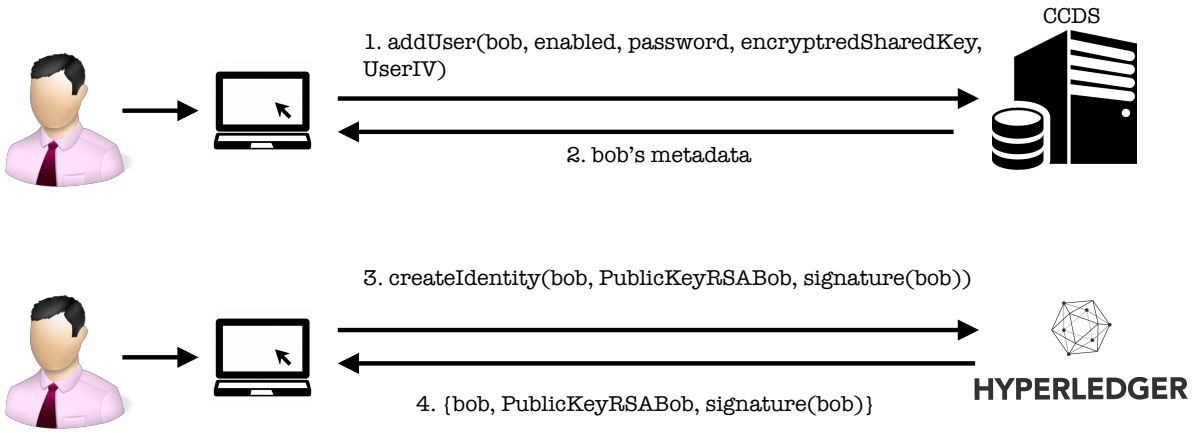


Figure 6.5: Bob's registration

Alice shares `file.txt` with Bob (5). The share operation controls the integrity, freshness and authenticity of files following the same operation as depicted in Figure 6.4. In addition, Alice's client retrieves Bob's identity from the ledger (8) and uses it to update the ACL (11).

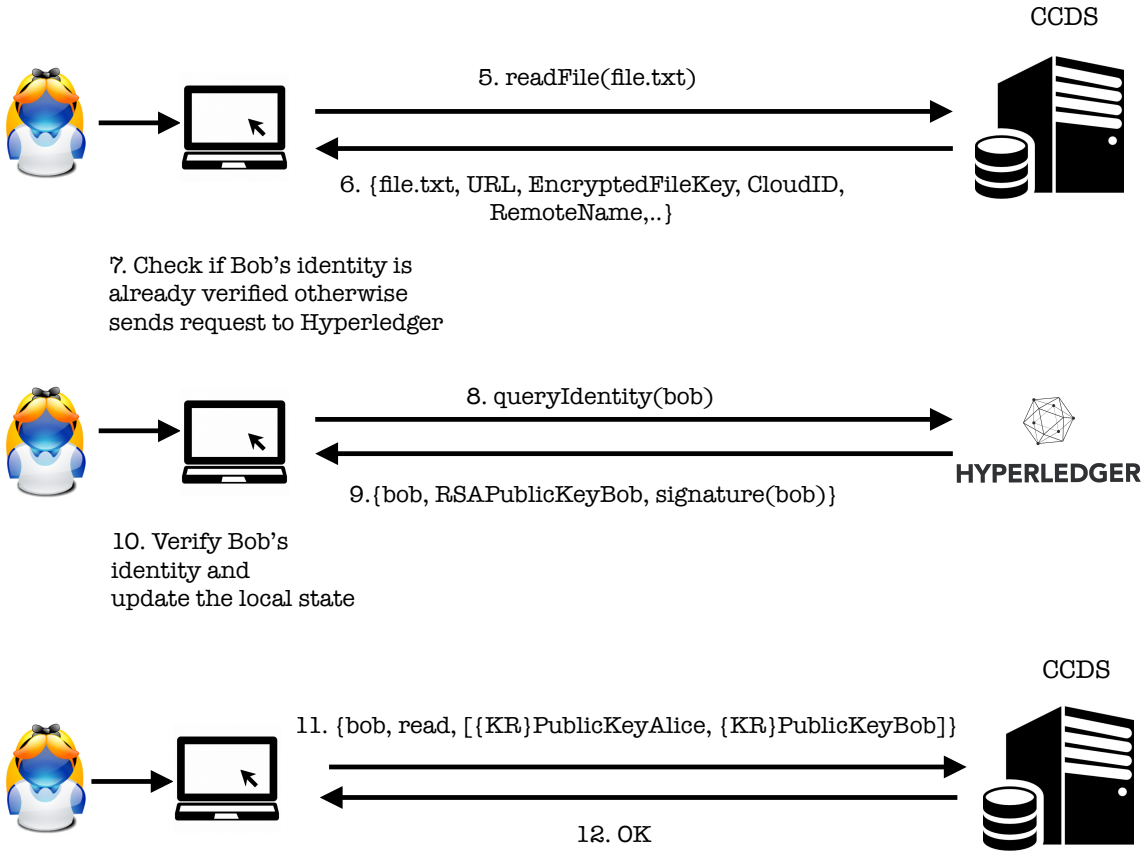
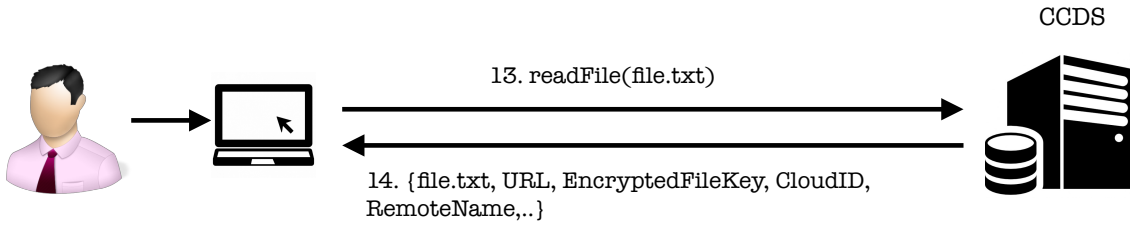


Figure 6.6: Sharing between Alice and Bob

Finally, Bob retrieves his files finding `file.txt` (13). The client application retrieves the last transaction related to the file from the blockchain (15). Then the client finds that it was made by Alice and checks the authenticity of the identity (20, 23, 26). Therefore, the client checks the integrity of the files (27, 28).



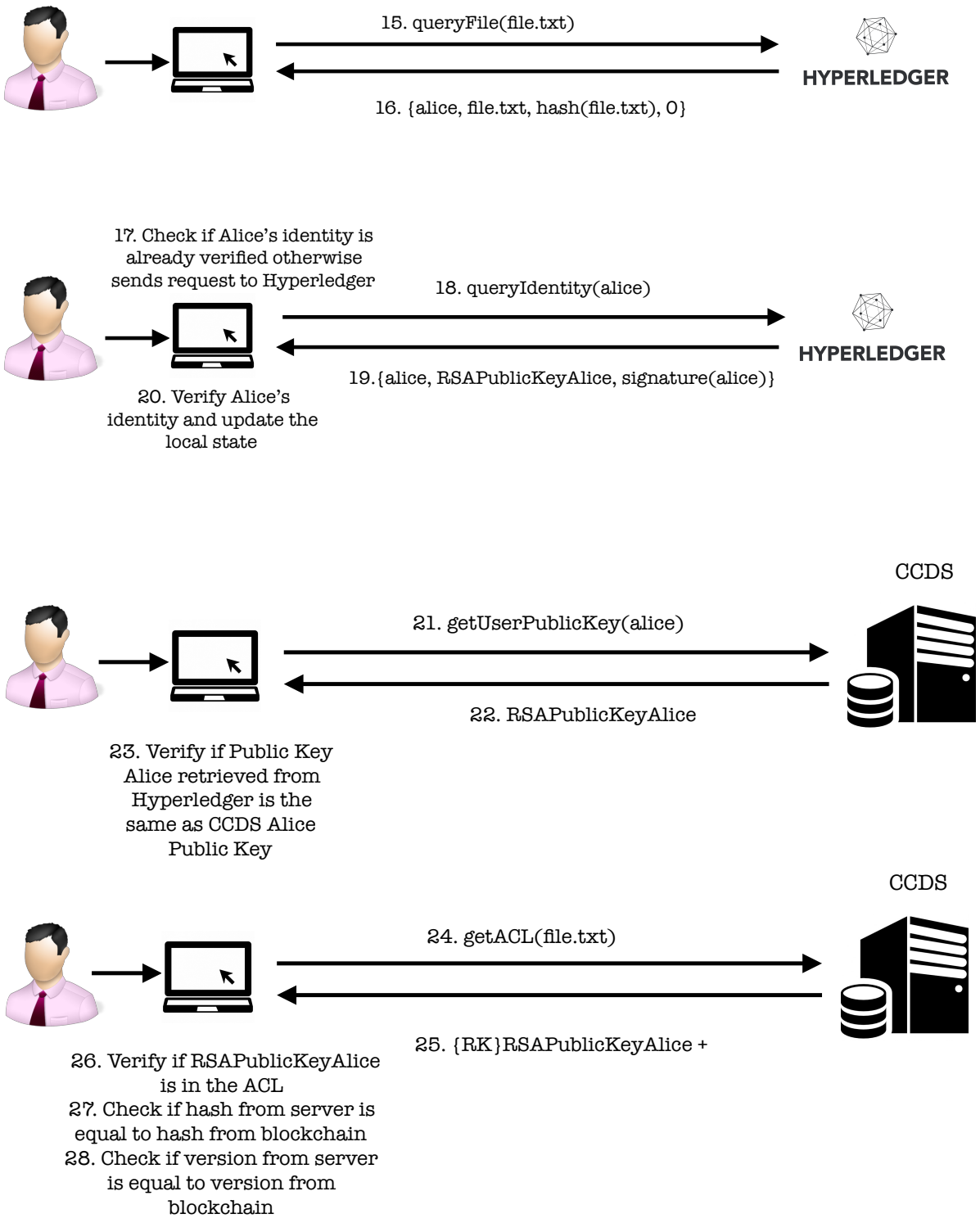


Figure 6.7: Bob's client retrieves files

6.5 Summary

In this chapter, we described the selected technologies and the implementation details for the proposed solution. The proposed Crypto Cloud modifications includes the management of the users' identities and the control of the integrity of files' metadata.

Regarding the management of the identities we proposed an implementation which combines the blockchain and the client application; we create an Enrollment Transaction that maps the user with his/her public key and we submit it on the Hyperledger blockchain. Then, every time a user performs a read/share operation, the client get the File Metadata Transactions related with files and verifies its authenticity. It means that the identity of the last transaction associated with the file in which the client is interested in must be done by a valid identity. To check the identity the client retrieves the Enrollment Transaction of the transactor and check if the signature is verifiable with the correspondent public key. Moreover we check the correspondence CCDS RSA public key - Hyperledger RSA public key and we verify if that key is on the Access Control List. All these mechanisms guarantee the authenticity and validity of the operations.

The other implemented functionality is the integrity mechanism. First of all we developed a control logic on the client side that submits the File Metadata transaction when creating files. Every time the application retrieve files, in addition of get files' metadata from the CCDS, it submits a query to the Fabric API getting the last File Metadata Transaction related with the file. Then, the content hash (which guarantees integrity) and the version retrieved from the blockchain are compared with the ones retrieved from the Client. If all checks pass, we can be sure about the validity of the metadata on the server.

The mechanism which guarantees the integrity of the Access Control List was not implemented because of time constraints.

Chapter 7

Evaluation

This chapter presents the evaluation of the proposed Crypto Cloud Proof of Concept. We start this evaluation by presenting the overall performance of Crypto Cloud with Hyperledger (CC-HL), describing the followed methodology and comparing the obtained results with the Crypto Cloud previous version (CC) (Section 7.2). In Section 7.3, we provide a security analysis of our solution, describing the countermeasures applied to mitigate the previously identified threats. Lastly, in Section 7.4, we summarize the main aspects of the evaluation of the proposed solution.

7.1 Performance Evaluation

To evaluate the performance of our solution, several benchmark tests were carried out. The latencies measured from benchmarks were obtained using a profiler software, called JProfiler v10.0 [40], which performs the instrumentation of the running code on a Java Virtual Machine (JVM) and traces its information. This technique has a relatively low overhead associated [40]. The experiments have been performed over an Intel(R) Core(TM) i7 3230M CPU running at 2,5 GHz with TurboBoost technology enabled, with 16GB of DDR3 memory running at 2133MHz, and 256GB of SSD. The OS used was macOs Mojave 10.14.2 (x64) running standard services. For all experiments, the Client Application, the CCDS, the Fabric Network and the PostgreSQL databases were deployed in the same machine as the benchmarks. The Fabric instance consists of one Peer, one Ordering Node, a Certification Authority, a Couch database and the two chaincodes discussed in Section 6.2.1. Every instance of these entities runs in a Docker container. The benchmark tests consist of the Client Application performing several operation requests to the system. These operations includes: reading a file from the cloud, writing a new file to the cloud, update an existing file and sharing a file with a user. Additionally, we also ran the same benchmarks using the existing Crypto Cloud prototype [1], in order to compare the obtained results with the original Crypto Cloud solution. We modified the previous version of the Crypto Cloud application in order to generate and consume local RSA key pairs; so we do not use the KMIP protocol.

All performed benchmarks measured the latency times of each operation on our system. These operations were executed individually 30 times, with approximately 10 seconds of interval between them, and its mean time and standard deviation

were analyzed. The experiments took place on January, 2019.

7.2 Crypto Cloud Performance

In order to evaluate the performance of the Crypto Cloud system, we obtained latency measures for each of the Crypto Cloud operations using different file sizes: 100KB, 1MB and 10MB. During these benchmarks, we did not considered the latency times resulting from the Cloud upload and download operations, since these operations are performed outside of our controlled environment and we cannot guarantee that all cloud operations are performed under the same conditions (e.g. the packets follow same network routes and use same provider's nodes). The values depicted below represent the latency measurements obtained from the performed operations.

The first operation of our interest is the addition of a user. As shown in Figure 7.1 in CC-HL the operation takes 25% more time than the same operation executed in CC. This is because when a user registers the first time to the system, he/she is both enrolled to the Hyperledger Membership Service Provider and to the CCDS. Moreover, the application puts the identity of the user on the ledger, submitting a transaction to the blockchain; this is done calling our Fabric API. The identity contains also the signature of the username of the transactor, so there is also an additional time due to the calculation of the signature.

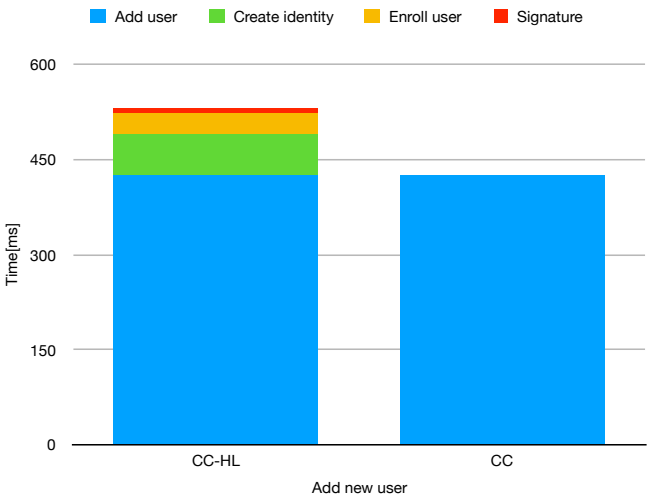


Figure 7.1: Crypto Cloud's mean latency of add user

As stated in Section 6.4, the read operation consists of getting the file's content and metadata both from the CCDS and Hyperledger blockchain; then, the application unwraps the file's keys, decipher the content and checks its integrity. From the obtained results, depicted in Figure 7.2, we can observe that the computation time of this operation increases with the rise of the file size parameter. This is true for both the system and can be explained by the increase in the time during the decipher process. We can observe that the CC-HL read of files took 20% to 40% more time than the same operation in CC. Analyzing the time spent in each read call, we found that the query blockchain function consumes a high percentage on the total time spent. Moreover, the application contacts the blockchain also to verify the

authenticity of the transactions (retrieves the identity and checks the signature) if the identity was not already checked. In addition, in the new version, we retrieve the ACL and the public key of the user who made the transaction from the CCDS. These operations are necessary to check that the consistency of the metadata held by server matches the ones on the blockchain, resulting in the increase of time of the read operation. Finally the client checks the integrity of the content hash and the version both from the server and Hyperledger.

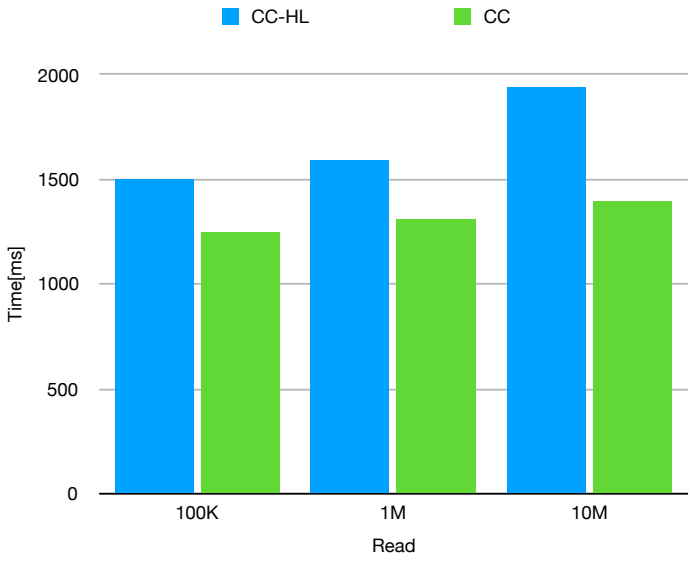


Figure 7.2: Crypto Cloud's mean latency of read files

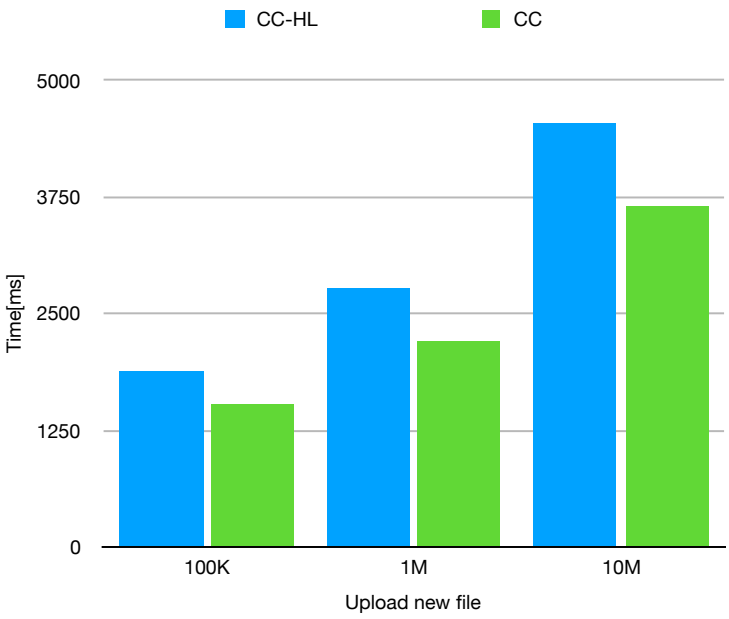


Figure 7.3: Crypto Cloud's mean latency of upload new files

Similar to the read operation, the write operation also depends on the file size parameter. As depicted in Figure 7.3, when performing the write of new files,

CC-HL took 20% to 25% more than the Crypto Cloud's time. The time is higher because every time a file upload operation is issued, the application submits a transaction on the blockchain containing the content hash, the version of the file, the filename and the username of the transactor.

Figure 7.4 depicts the result of uploading of existing files (the file is modified every time). In the case of upload of an existing file, our Proof of Concept took 10% to 20% more than the CC time. This operation is different from the upload of a new file because the application also finds the last valid transaction on the blockchain. This is necessary to check the integrity of the metadata held by the server. Moreover, the application verifies the identity of the transactor, which must be valid. To retrieve the public key and the ACL from the server, additional calls were done. All these operations contribute to the increase in the overall time.

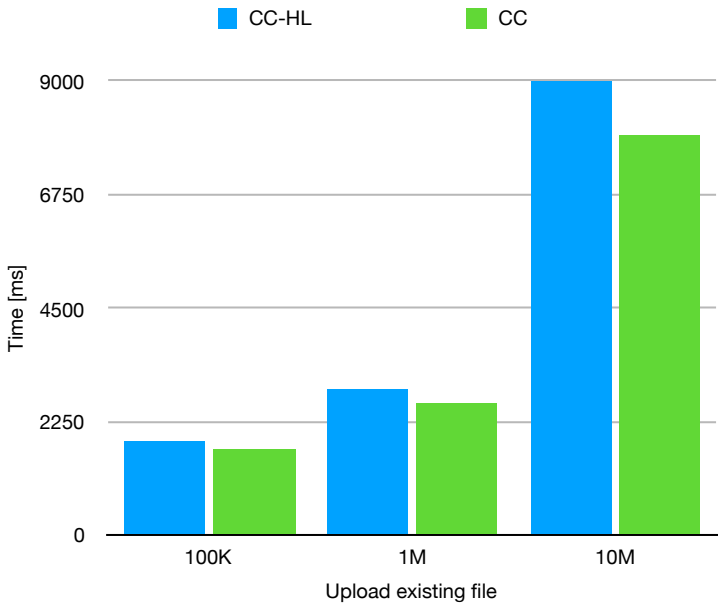


Figure 7.4: Crypto Cloud's mean latency of upload existing files

The share operation consists of wrapping the file's Read Key (RK) with another user's public key. This operation only involves the file's metadata and does not deal with the file content, which results in similar latency times for different file sizes. When comparing the obtained results with Crypto Cloud, we can observe a high increase in the operation time. The results differs in an order of magnitude: in the previous version of Crypto Cloud, the time is expressed in microseconds while in the version with Crypto Cloud it is measured in milliseconds. Because of this significant difference, it does not make sense to create a graph of the results.

In the share of the new CC, in addition to the operations done in old CC, we call three times the Fabric API: one to retrieve the file metadata, one to retrieve the identity of transactor, and one to retrieve the identity of grantee user.

7.3 Security Analysis

The gain of using the Hyperledger Fabric blockchain can be summarized in terms of integrity, authenticity and authentication and not trusting the server. The implementation of Crypto Cloud follows different mechanisms in order to mitigate the previously identified threats, in Section 5.3.2.

The Spoofing Identity threats consist of the violation of the system's authentication properties, where a malicious entity poses as an authorized one. Crypto Cloud verifies the identity of the users every time an operation is issued; only the users that can demonstrate to hold a specific identity could do operations. The verification is done relying on the blockchain and checking the signature contained in the transactions. This guarantees the authenticity of the data on the blockchain.

Regarding threats that involve Tampering with Data, violating integrity properties of the system, an attacker (e.g. malicious cloud) may try to modify users' files. Our solution deals with this by implementing an integrity mechanism that relies on the blockchain; every time a file is read, the application checks the content hash and the version retrieved from the blockchain with the ones obtained from the server. In this way, integrity and freshness of files is always guaranteed. This prevents users from denying that they performed modifications to the files.

In order to mitigate Repudiation threats, where a user denies performing an action over a file, our implementation counters this threats by recording the metadata related with creation and upload of files on the blockchain. This prevents users from denying that they performed modifications to the files.

Regarding Information Disclosure on the blockchain, the confidentiality of the metadata stored on the blockchain is guaranteed because the hash and the version of the files do not expose sensible information. Also the public key and signature do not expose sensitive information of the users.

The Denial of Service threats are still possible, as in the previous version of Crypto Cloud.

Regarding the Elevation of Privilege threats, our solution deals with this problem by implementing strict access control mechanisms based on ACLs that verify user's permission over the files on every request. But in the implementation we do not check the integrity of the ACL and this would expose the system a potential tampering of ACL. This represents a weakness of this work.

7.4 Summary

In this chapter, we evaluated the solution based on performance and security. In terms of performance, we evaluated the modified operations and compared the obtained results with the results of the previous version of Crypto Cloud. All the operations add an overhead due to the communication with blockchain.

Regarding security, our solution implemented different mechanisms to mitigate the previously identified threats. Our solution guarantees the confidentiality and integrity of stored files without trusting the Crypto Cloud Directory Server by relying instead on the blockchain. Even if the attacker compromises clouds' infrastructures or the server's metadata, the users' files remain secure. Crypto Cloud also implements authentication and authorization mechanisms to prevent illicit access. The

users' identities, the integrity and validity of files are verified using the blockchain environment.

With the proposed solution we increased the global consumption on the client side but with the gain of having not to trust the server anymore. Through the assessment of the Crypto Cloud solution and based on its design and implementation details, we can conclude that our Proof of Concept solution meets all the previously established requirements except the integrity of the ACL. The remaining mechanisms described in the design can be easily implemented in future work.

Chapter 8

Conclusion

The Crypto Cloud system allows using multiple cloud providers without renouncing privacy, guaranteeing the confidentiality and integrity of managed files. We extended this system using blockchain technology, since it delivers trust, transparency, neutrality, security, and immutability without having to trust the Crypto Cloud central server. By using the blockchain, we prevent the loss of integrity of the metadata held by the server; so we created a tamper-proof system.

We started by analyzing different blockchain implementations, such as Hyperledger Fabric, Ethereum, and Filecoin; we compared them and illustrated the benefits/disadvantages of each technology. We chose Hyperledger Fabric because it is a permissioned blockchain which guarantees the authentication, integrity, and privacy of the stored data.

In this work, the integration of Crypto Cloud with Hyperledger Fabric blockchain was successfully implemented, except delivering the integrity of the Access Control List due to time constraints. The new system provides a mechanism to validate the public keys and associated identities of the users, using the blockchain to provide authenticity of the public keys and using the client to verify it. A new component was added to the CC architecture which is the Hyperledger Fabric network. We created two smart contracts which represent, respectively, users' identities and files' metadata. To interact with such smart contracts, we implemented an API using Hyperledger Fabric Java SDK [41]. In addition, the protection of the stored files is enhanced by using the blockchain. We added the metadata which guarantees the integrity of files on the blockchain; every time a user wants to read his/her files, the client application verifies the consistency between the metadata on the server and the ones on the blockchain. In this way, if the server contains tampered metadata, it will be detected and discarded. Regarding performance, the new Crypto Cloud adds an overhead to the creation and reading of files, taking about 20% to 40% more time than the previous version. Nevertheless, we gain the advantage of not having to trust the server anymore, always guaranteeing that the metadata held by the server is not compromised.

The major implementation difficulties were in the implementation of the blockchain. This occurs because, even if the Hyperledger technology is well documented, there are some aspects that are not so clear, due to its "young" nature; in particular, there are very few examples (and very few really works) about how to use the Hyperledger Fabric Java SDK. This slowed the implementation significantly.

Finally, it is possible to affirm we have seen a great increase in the interest

about the blockchain technology in the last years. The blockchain has gained much attention because it ensures security and immutability of the data stored. What can be deduced from this research is that with the blockchain the point of view of how an application is developed completely changes. Not relying anymore on a single entity or a third party is a real revolution both regarding security and availability. This simplifies many types of application that require the involvement of several third parties. Breaking down the burden of trusting external entities is a step forward to innovation. In our case, it is translated into having no longer to worry about the server, that can manipulate sensitive informations. With our Proof of Concept, we gained security control without trusting neither the central server nor the cloud providers. However, the panorama in this area is huge, and this is just a small drop in the ocean of the future.

8.1 Future Work

The solution herein proposed was properly implemented and achieves its proposed goals except the implementation of the Access Control List integrity. Therefore, there are few improvements that can be taken into account in future versions of this work, such as:

- **Integrity of the Access Control List**: the hash of the ACL must be stored on the blockchain to prevent that malicious users could modify it without the right permissions. The mechanism is explained in detail on Section 5.6;
- **Removal of the server**: all the most important metadata were moved on the blockchain, so it will be possible to eliminate the server. The remaining metadata could be maintained on the client side.
- **Create a distributed storage, based on blockchain technology**: there are technologies like Filecoin, which allows users to sell and buy remote storage space for backing up their data. Adopting a technology as Filecoin, it will be possible to substitute the cloud providers with the Filecoin Decentralized Storage Network (see Section 4.3). In addition, it is possible to integrate the functionalities discussed in Chapter 5 guaranteeing authenticity, integrity, and freshness of the files.

Bibliography

[1] F. D. B. Custodio, "Crypto cloud", master thesis in information systems and computer engineering, Instituto Superior Tecnico, 2017

[2] S. Pereira, A. Alves, N. Santos, and R. Chaves, "Storekeeper: A security-enhanced cloud storage aggregation service", Reliable Distributed Systems (SRDS), 2016 IEEE 35th Symposium on, 2016

[3] OASIS, "Key management interoperability protocol technical committee." Available: `https://www.oasis-open.org/committees/kmip/`

[4] S. Goldfeder, H. Kalodner, D. Reisman, and A. Narayanan, "When the cookie meets the blockchain: Privacy risks of web payments via cryptocurrencies", Proceedings on Privacy Enhancing Technologies, 2018

[5] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, "An overview of blockchain technology: Architecture, consensus, and future trends", Big Data (BigData Congress), 2017 IEEE International Congress on, 2017

[6] Z. Zheng, S. Xie, H.-N. Dai, X. Chen, and H. Wang, "Blockchain challenges and opportunities: A survey", International Journal of Web and Grid Services, 2018

[7] Deloitte, "When (and if) income is realized from bitcoin chain-splits", 2017

[8] Techopedia, "Smart contract." Available: `https://www.techopedia.com/definition/32499/smart-contract`

[9] E. T. from the arXiv, "Bitcoin transactions are not as anonymous as everyone hoped." Available: `https://www.technologyreview.com/s/608716/bitcoin-transactions-arent-as-anonymous-as-everyone-hoped`

[10] Plusalsight, "Blockchain architecture." Available: `https://www.pluralsight.com/guides/blockchain-architecture`

[11] A. Lewis, "A gentle introduction to blockchain technology", Bits on Blocks, 2015

[12] G. W. Peters and E. Panayi, "Understanding modern banking ledgers through blockchain technologies: Future of transaction processing and smart contracts on the internet of money", Banking Beyond Banks and Money, 2016

[13] P. M. L. Costa, "Supply chain management with blockchain technologies", 2018

[14] Z. Zheng, S. Xie, H.-N. Dai, and H. Wang, "Blockchain challenges and opportunities: A survey", Work Pap.–2016, 2016

[15] A. Tar, "Proof-of-work." Available: https://it.cointelegraph.com/explained/proof-of-work-explained

[16] Z. Zheng, S. Xie, H.-N. Dai, X. Chen, and H. Wang, "Blockchain challenges and opportunities: A survey", International Journal of Web and Grid Services, 2018

[17] M. Castro, B. Liskov, *et al.*, "Practical byzantine fault tolerance", OSDI, 1999
[18] "What is practical byzantine fault tolerance?." Available: https://crushcrypto.com/what-is-practical-byzantine-fault-tolerance/
[19] T. T. A. Dinh, J. Wang, G. Chen, R. Liu, B. C. Ooi, and K.-L. Tan, "Blockbench: A framework for analyzing private blockchains", Proceedings of the 2017 ACM International Conference on Management of Data, 2017
[20] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, *et al.*, "Hyperledger fabric: a distributed operating system for permissioned blockchains", Proceedings of the Thirteenth EuroSys Conference, 2018
[21] P. Yee, "Updates to the internet x. 509 public key infrastructure certificate and certificate revocation list (crl) profile", 2013
[22] IBM, "Identity, hyperledger fabric docs." Available: https://hyperledger-fabric.readthedocs.io/en/release-1.3/identity/identity.html?highlight=digital%20identity
[23] Hyperledger Fabric documentation, Available: https://hyperledger-fabric.readthedocs.io/en/release-1.3/
[24] IBM, "Chaincode, hyperledger fabric docs." Available: https://hyperledger-fabric.readthedocs.io/en/release-1.3/smartcontract.html?highlight=chaincode
[25] IBM, "Identity mixer." Available: https://www.zurich.ibm.com/identity_mixer/
[26] H. D. Initiative *et al.*, "What is ethereum?", tech. rep. Available: http://ethdocs.org/en/latest/introduction/what-is-ethereum.html
[27] G. Wood, "Ethereum: a secure decentralised generalised transaction ledger. ethereum project yellow paper 151 (2014)", 2014
[28] Ethereum, "Parity ethereum documentation." Available: https://wiki.parity.io/Parity-Ethereum
[29] T. Tiwari, D. Starobinski, and A. Trachtenberg, "Distributed web mining of ethereum", International Symposium on Cyber Security Cryptography and Machine Learning, 2018
[30] P. Labs, "Filecoin: A decentralized storage network", 2017
[31] U. Chohan, "The double spending problem and cryptocurrencies", 2017
[32] V. B. on the Future of Ethereum, Available: https://hackernoon.com/vitalik-buterin-on-the-future-of-ethereum-d3577317b0cf
[33] B. N. Levine, C. Shields, and N. B. Margolin, "A survey of solutions to the sybil attack", University of Massachusetts Amherst, Amherst, MA, 2006
[34] X. L. Amanda Davenport, Sachin Shetty, "Attack surface analysis of permissioned blockchain platforms for smart cities", 2017
[35] J. B. P. Labs, "Filecoin research roadmap for 2017", 2017, Available: https://filecoin.io/research-roadmap-2017.pdf
[36] M. Valenta and P. Sandner, "Comparison of ethereum, hyperledger fabric and corda", tech. rep., FSBC Working Paper, 2017
[37] IBM, "Hyperledger fabric." Available: https://github.com/hyperledger/fabric/blob/13447bf5ead693f07285ce63a1903c5d0d25f096/common/tools/configtxgen/localconfig/config.go
[38] D. Johnson, A. Menezes, and S. Vanstone, "The elliptic curve digital signature algorithm (ecdsa)", International journal of information security, 2001

[39] J. S. Perry, "Blockchain:why can not this be done with a database?." Available: https://developer.ibm.com/code/2018/04/16/blockchain-vs-or-database-better/

[40] E. Technologies, "Java profiler - jprofiler." Available: https://www.ej-technologies.com/products/jprofiler/

[41] "Java sdk for hyperledger fabric 2.0." Available: https://github.com/hyperledger/fabric-sdk-java

List of Abbreviations

CC Crypto Cloud

CCDS Crypto Cloud Directory Server

ACL Access Control List

MSP Membership Service Provider

RK Read Key

SDS Storekeeper Directory Server

API Application Programing Interface

KMIP Key Management Interoperability Protocol

URL Uniform Resource Locator

PKI Public Key Infrastructure

HMAC Hash-based Message Authentication Code

PoW Proof of Work

PBFT Practical Byzantine Fault Tolerance