Hypercloud: A blockchain-based secret management in multi-cloud storage platforms

Marco António Vaz Coelho

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisors: Prof. Miguel Ângelo Marques de Matos
Prof. Ricardo Jorge Fernandes Chaves

Examination Committee
Chairperson: Prof. Luís Manuel Antunes Veiga
Supervisor: Prof. Miguel Ângelo Marques de Matos
Member of the Committee: Prof. Miguel Nuno Dias Alves Pupo Correia

November 2020
To my family
Acknowledgments

I would like to thank my supervisors Miguel Matos and Ricardo Chaves for all the guidance throughout this thesis.

I extend my sincere gratitude to my family for their constant support during this thesis.
Resumo

Os serviços de armazenamento na cloud permitem aos utilizadores armazenar ficheiros, aceder a essa informação em qualquer lugar com acesso à Internet, e partilhá-los com outros utilizadores. Os utilizadores podem ter mais do que uma conta tornando a gestão mais difícil visto que os ficheiros estão espalhados por diversas contas. Os cloud storage aggregators permitem aos utilizadores terem uma visão global dos seus ficheiros, simplificando a gestão dos mesmos. Os sistemas atuais não apresentam soluções satisfatórias para a partilha de ficheiros entre utilizadores de cloud providers diferentes ou garantias de confidencialidade dos ficheiros visto que necessitam de confiar em entidades intermediárias, como um servidor central ou serviços de e-mail. Nesta dissertação, apresentamos o Hypercloud, um cloud storage aggregator que garante confidencialidade dos ficheiros do utilizador, permite a verificação da integridade dos mesmos e permite a partilha de ficheiros entre utilizadores sem necessitar de confiar em entidades intermédias. Hypercloud é mais lento na execução das funcionalidades quando comparado com outros cloud storage aggregators, mas oferece um aumento das garantias de segurança que não são possíveis de obter com outros sistemas do mesmo tipo.

Palavras-chave: Blockchain, Armazenamento Cloud, Confidencialidade, Integridade, Autenticação
Abstract

Cloud storage services allow users to store data, access the information from everywhere with an Internet connection and share it with other users. Users can have more than one cloud account making file management harder since the files are scattered. Cloud Storage Aggregator systems allow the users to have a global view of their data of all cloud accounts, simplifying file management. These present systems do not present a satisfactory solution to allow file sharing between users of different cloud providers or guarantee the confidentiality of the files because they need to trust in intermediary entities, like a central server or e-mail services. In this thesis, we present Hypercloud, a cloud storage aggregator that guarantees confidentiality and allows the verification of the integrity of the user’s files, allows file sharing between users of the same and different cloud providers, without the need to trust in intermediary entities. Hypercloud presents a slower performance when compared to other cloud storage aggregators, but increased guarantees of security that are not possible in other systems.

Keywords: Blockchain, Cloud Storage, Confidentiality, Integrity, Authentication
# Contents

Acknowledgments ....................................................... v
Resumo ............................................................... vii
Abstract ............................................................... ix
List of Tables .......................................................... xv
List of Figures .......................................................... xv
Nomenclature .......................................................... xvii
Glossary ............................................................... 1

1 Introduction .......................................................... 1
   1.1 Motivation ....................................................... 2
   1.2 Objectives ....................................................... 2
   1.3 Thesis Outline .................................................. 3

2 Background .......................................................... 5
   2.1 Cryptography .................................................... 5
       2.1.1 Symmetric Cryptography ................................ 5
       2.1.2 Asymmetric Cryptography .............................. 5
       2.1.3 Hashing .................................................... 6
       2.1.4 Message Authentication Codes ......................... 6
       2.1.5 Digital Certificates ...................................... 6
   2.2 KMIP ............................................................. 7
   2.3 OAuth ........................................................... 7
   2.4 Blockchain ...................................................... 8
       2.4.1 Smart Contracts ........................................... 8
       2.4.2 Types of blockchains ................................... 9
       2.4.3 Consensus .................................................. 9
   2.5 Examples of Blockchains ..................................... 11
       2.5.1 Ethereum .................................................. 12
# Implementation

5.1 Architecture .................................................. 39
5.2 Data Structures ............................................... 41
5.3 Functionalities ................................................ 43
   5.3.1 Register user ............................................ 44
   5.3.2 Login user ............................................... 45
   5.3.3 Update LToken .......................................... 46
   5.3.4 Update LFiles .......................................... 47
   5.3.5 Download File ......................................... 49
   5.3.6 Update File ............................................ 51
   5.3.7 Update list of permissions of the ACL ............. 52
   5.3.8 Objectives and Solutions .............................. 54
   5.3.9 Summary ............................................... 55

# Evaluation

6.1 Performance Evaluation ...................................... 57
6.2 Hypercloud performance .................................... 58
   6.2.1 Performance of the Functionalities ................. 58
   6.2.2 Time per operation .................................... 61
6.3 Performance comparison ..................................... 65
6.4 Security Analysis ............................................ 67
   6.4.1 Spoofing ................................................. 68
   6.4.2 Tampering ............................................... 68
   6.4.3 Repudiation ............................................. 69
   6.4.4 Information Disclosure ................................. 69
   6.4.5 Denial of Service ...................................... 69
   6.4.6 Elevation of privilege ................................ 69
6.5 Summary ..................................................... 70

# Conclusions

7.1 Future Work .................................................. 71

Bibliography ....................................................... 73
List of Figures

3.1 Architecture of DepSky [31] .................................................. 18
3.2 The combination of symmetric encryption, secret sharing, and erasure codes in
   DEPSKY-CA [31] ................................................................. 19
3.3 SCFS Architecture [32] .......................................................... 20
3.4 Encoded file with parity blocks added for both the server and dispersal codes [33] 22
3.5 Architecture of Storekeeper [29] ............................................. 23
3.6 Architecture of Cryptocloud [34] ............................................. 24
3.7 Architecture of Cryptocloud with Hyperledger Fabric [37]. ...................... 27
4.1 Hypercloud’s architecture ....................................................... 33
5.1 Client application modules .................................................... 40
5.2 Register User ................................................................. 44
5.3 Login ................................................................. 45
5.4 Add Cloud Account ........................................................... 46
5.5 Upload New File ............................................................. 48
5.6 Synchronize Files ............................................................. 49
5.7 Download File ............................................................. 50
5.8 Update File ............................................................. 51
5.9 User A shares a file F with the user B ....................................... 53
5.10 User A removes the access of the user B to the file F ......................... 54
6.1 Time to upload a new file with different sizes ................................. 58
6.2 Time to download a new file for files files with different sizes .......... 59
6.3 Time to assign different permissions to a file ................................ 60
6.4 Revoke Share for files with 100KB, 1MB and 10MB ........................ 60
6.5 Revoke Share for files with 100KB, 1MB and 10MB ........................ 61
6.6 Cloud operation for files with 100KB, 1MB and 10MB ..................... 61
6.7 Time to execute specific operations of the functionalities of Hypercloud. The
colours have the different meanings in each column. .......................... 62
6.8 Performance comparison for a 100KB file ................................. 65
6.9 Performance comparison for a 1MB file .................................... 66
6.10 Performance comparison for a 10MB file ................................. 66
Nomenclature

ACL  Access Control List
API  Application Programing Interface
FK  File Key
K   AES Key
KMIP Key Management Interoperability Protocol
KMS Key Management Service
PK  RSA Private Key
PK_BC ECDSA Private Key
PU  RSA Public Key
PU_BC ECDSA Public Key
RK  Read Key
URL Uniform Resource Locator
Chapter 1

Introduction

Over the years, the presence of cloud storage services, like Dropbox [1] and Google Drive [2], in today’s society has been increasing. Cloud storage services are a convenient and reliable way to save videos, images, and every other type of file. These services are a good choice to store files because they are built to be highly available, allowing the user to access their files anytime they have access to the internet. These services also present solutions to allow file sharing between users, for example, Dropbox supports the creation of shareable links that allow every person with access to the link to read and download the file.

Some users may have multiple cloud accounts. With multiple accounts, the user will have his files scattered through multiple accounts and the management of his data becomes harder. Cloud storage aggregators solve this problem. These systems present a global view of the user’s files from all of his accounts, making the file management easier. Some cloud storage aggregators allow users to share files with other users even if they are using cloud accounts of different providers. An example of a sharing solution is the generation of links that are shared and allow the users to access the file. These links may be shared through e-mail or using a central server that manages file access permissions.

The present solutions for cloud storage aggregators facilitate the management of the files of the user but do not address important usability and security problems. There are important security problems that need to be taken into account:

- The need to use different means of communication to share files, like e-mail;
- The assumption that the cloud providers can be completely trusted; do not offer a way to verify the file’s integrity;
- The need to trust in intermediate entities like a central server.
Until the time that this dissertation was written, and to the best of our knowledge, no solution takes into account all these problems.

In this dissertation, we present Hypercloud, a cloud storage aggregator solution that aims to improve the security of this type of system, solving the enunciated problems. This solution is achieved using the blockchain technology with traditional cryptography. Hypercloud also uses TLS channels, for the communication between the various entities of the system, and OAuth, for the user’s authentication and operations in their clouds. These protocols are well known and widely accepted.

1.1 Motivation

Cloud storage is a useful technology that presents a way for users to save and access their files, guaranteeing high availability of the system. Cloud storage aggregators are solutions that facilitate the file management of users with multiple cloud accounts. They present a global view of the user’s files. To this date there was no system that offers all the following characteristics:

- **Possibility to share files without the necessity of external services**: file sharing is a useful functionality that allows the users to share the files between each other, independently of the cloud providers being used. The use of external services to share the files makes the system dependent on their correct behavior and availability.

- **Assume the cloud provider may try to read and modify the user’s files**: We cannot control how the cloud providers behave. If the cloud providers read or are able to change the file’s content, the confidentiality and integrity of the user’s files are not guaranteed. It is possible to know if they are managing the files correctly. This verification would guarantee the users would not have access to a file that is not valid.

- **Trust in intermediate entities like a central server**: the addition of entities in the system’s architecture makes the system more vulnerable to failures since it has to trust in more parts for it to work properly. This addition of entities also increases the number of possibilities for a system to fail.

1.2 Objectives

The objective of this dissertation is to present a cloud storage aggregator solution:

- That guarantees the confidentiality of the managed files from outsiders, such as the cloud providers and other users;
• That allows the verification of the file’s integrity;

• That supports file sharing between users of the system;

• That is not dependent of intermediate entities like a central server or e-mail services.

The solution that we will present explores the blockchain technology with the addition of traditional cryptography to achieve these objectives.

1.3 Thesis Outline

In this chapter we introduced the topic, the problems and the objectives for this solution. In Chapter 2, we present the required background to understand the problem and the proposed solution. In Chapter 3, we present the related work to this topic. In Chapter 4, we present the proposed solution, explaining the architecture, the assumptions and the functionalities of the system. In Chapter 5, we explain how the solution was implemented. In Chapter 6, we present the performance and security evaluation. In Chapter 7, we conclude this document and present improvements for future work.
Chapter 2

Background

This chapter introduces the required background. In Section 2.1 we present the required concepts about cryptography. In Section 2.2 we present the KMIP protocol. In Section 2.3 we present the OAuth protocol. In Section 2.4 we introduce the concept of blockchain. In Section 2.5 we present some examples of blockchains. Finally, in Section 2.6 we summarize this chapter.

2.1 Cryptography

Cryptography has three important building blocks: hashing, symmetric, and asymmetric cryptography. These cryptographic mechanisms can be used to provide data confidentiality, authentication, data integrity, and non-repudiation.

2.1.1 Symmetric Cryptography

Symmetric cryptography [3] uses a secret key to cipher and decipher data. This method transforms the input data (e.g. plaintext input) into encrypted data (e.g. ciphertext) and vice-versa. The secret key can be shared between two or more entities, guaranteeing the confidentiality of the ciphered data. The entities that hold a copy of the secret key must maintain it secure because the security of symmetric encryption depends on the secrecy of the key, not the secrecy of the algorithm.

2.1.2 Asymmetric Cryptography

Asymmetric cryptography [3] uses two separate keys: a public key and a private key. In this approach, each user generates its key pair and places the public key in a public register. The private key is kept secret. If a user B wishes to send a secret message to a user A, B encrypts the message using A’s public key. When A receives the message, A will decipher the message
using its private key.

This approach can also be used to sign a message and exchange keys. To sign a message, since the private key is only accessible by its owner, the sender applies the cryptographic algorithm to the data or to a transformation of it, using his private key as the key to cipher the data. Then, the receiver can verify that the message was sent by the correct entity, using the same algorithm to decipher the message with the sender’s public key.

2.1.3 Hashing

A hash function [3] is used to produce a “fingerprint” of a message. It has the following properties:

- The function can be applied to a variably sized block of data; the function produces a fixed-length output;
- The same data X will always produce the same output H(X); similar data produce a very different hash value;
- It is computationally infeasible to obtain the original data from the hash;
- It is infeasible to find two equal outputs using different inputs.

Those functions can be used to check the integrity of the data comparing the received hash with the calculated hash of the actual data.

2.1.4 Message Authentication Codes

A Message Authentication Code (MAC) [3] is an authentication technique that involves the use of a secret key to generate a small block of data that is appended to the message. This technique assumes that the sender and the receiver, say A and B, share the same secret key. When A has a message to send to B, A calculates the message’s MAC as a function of the message and the shared key. A appends the calculated MAC to the message and sends it to B. B performs the same calculations on the received message, using the same secret key, generating a new MAC. The received code is compared with the calculated one. By doing this verification, the receiver can detect changes in the message.

An HMAC is a type of MAC that uses a hash function and cryptographic keys.

2.1.5 Digital Certificates

The combination of a public key that has been signed by a certification authority (CA), the accompanying digital signature and certain additional parameters are called a certificate [4]. A
Digital Certificate is used to cipher data that only the owner of the corresponding private key can decipher. Can also be used to verify if the data was signed by the user with the private key that corresponds to the certificate.

The X.509 standard has become the most relevant one. This standard specifies the structure of the certificates. An X.509 certificate has an issuer name, a subject name, the subject’s public key, a period for which the certificate is valid, the issuer’s signature, and other fields.

2.2 KMIP

KMIP [5, 6] is a protocol that simplifies the way companies manage cryptographic keys, eliminating the need for redundant, incompatible key management processes. The KMIP standard enables key lifecycle management that includes the generation, submission, retrieval, and deletion of cryptographic keys. For user authentication, a KMIP client and server need to use TLS to negotiate a mutually-authenticated connection except for the Query operation. The query operation do not require the client to assure its authenticity. This standard also defines the message encoding format, specifies the expected results for certain use cases and describes proper error handling that may occur during the interaction.

2.3 OAuth

OAuth [7] is an authorization protocol. It introduces an authorization layer and separates the client’s role from the resource owner’s role, protecting the end user’s credentials. The client, instead of using the credentials of the resource owner, receives an access token to access the protected resources. The access token is a string denoting a specific scope, lifetime, and other access attributes. Access tokens are issued to third-party clients by an authorization server with the approval of the resource owner. This protocol is used, for example, by Dropbox, to allow applications to interact with the data the user has in his cloud account.

OAuth standard defines four roles: the resource owner, resource server, client, and authorization server. The resource owner is the entity (person or application) that owns the resource. The resource server is the server that stores the protected resources. The client is an application making protected resource requests on behalf of the resource owner and with its authorization. The authorization server is the server issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorization.

The protocol defines four modes of authorization called grant types: authorization code, implicit grant, resource owned password credentials, and client credentials.
The **Authorization Code** allows the user to authenticate himself towards the authentication server. The authentication server provides an authorization code to the user. The user introduces the authorization code in the client application. The client application sends a request to the authentication server asking for an access token. The client application uses the provided access token to access the user’s resources stored at the resources server.

With the **Implicit grant**, instead of issuing the client an authorization code, the client is issued an access token directly. The user inserts the access token directly in the client application and the client application uses that access token to access the user’s resources.

With **Resource owned password credentials**, the user gives direct access to the client to its credentials. This method should only be used when there is a high degree of trust between the resource owner and the client. The client uses the credentials to request a valid access token to the authorization server. The client uses the valid access token to access the protected resources. With **Client credentials** the client is also the access owner. The client application authenticates itself with its own credentials towards the authorization server. The client app receives a valid access token. The client application uses the provided access token to access the resources.

### 2.4 Blockchain

The blockchain technology [8] allows the creation of a decentralized system where there is no need for a third party to control the transactions. Blockchain is a distributed database solution that maintains a continuously growing list of data records that are confirmed by the nodes participating in it. The data is recorded in a ledger, that includes the information of every transaction ever completed and it is available to all nodes. The ledger is verifiable and the information that was already written cannot be erased [9]. The blockchain is organized in a sequence of blocks, which holds a complete list of transaction records like a conventional public ledger. Each block points to the immediately previous block via a reference. A reference is the hash value of the previous block. Those references allow users to verify and trace previous records. For example, in the Bitcoin blockchain, each transaction could be traced to previous transactions iteratively. This improves the transparency of the data stored in the blockchain [10].

#### 2.4.1 Smart Contracts

A smart contract is a computer program that allows to express business logic in code. A smart contract is triggered when a transaction addresses it. It is executed independently and
automatically in a prescribed manner on every node in the network, according to the data that
was included in the transaction that it triggered. This allows interactions between mutually
distrustful entities safely because the transacting entities can inspect the code and identify its
outcomes before deciding to engage with the contract. Both entities have certainty that the
code will execute since it is already deployed on a network that neither of them fully controls;
the entities have verifiability over the process since all the interactions are digitally signed [11].

2.4.2 Types of blockchains

Blockchains can be classified by the permissions of the nodes and by the organizations that con-
trol the network. The first distinction is between permissionless and permissioned blockchains.

Permissionless blockchains are open and decentralized. Any peer can join and leave the
network as a reader and as a writer at any time. There is no central entity that manages the
membership or which could ban illegitimate readers or writers. This means that the written
content is readable by any peer. With the use of cryptographic primitives, it is technically
possible to design a permissionless blockchain that hides privacy relevant information [12].

Permissioned blockchains only authorize a limited set of readers and writers. Here, a central
entity decides and attributes the right to individual peers to participate in the write or read
operations of the blockchain [12].

The second distinction is between public, private and consortium blockchains.

In public blockchain, all records are visible to the public and everyone could take part in
the consensus process [13]. A public blockchain is also called a permissionless blockchain (e.g.
Ethereum [14]).

In a private blockchain, only those nodes that come from one specific organization are allowed
to join the consensus process. A private blockchain is regarded as a centralized network since
it is fully controlled by one organization [13] and can only be permissioned (e.g. HyperLedger
Fabric [15]).

In a consortium blockchain, only a group of pre-selected nodes would participate in the
consensus process. This type of blockchain is constructed by several organizations. Since only a
small portion of nodes would be selected to determine the consensus it is partially decentralized
[13].

2.4.3 Consensus

In the blockchain, there is no central node that ensures that the ledgers on the distributed
nodes are all the same. To ensure that the ledgers in different nodes are consistent a consensus
protocol is used [10]. We present four consensus protocols: Proof of Work [10], Proof of Stake [10], Practical Byzantine Fault Tolerance [16] and Raft [17].

In **Proof of Work** (PoW) [10], each node of the network is calculating a hash value of the block header. The nodes that calculate the hashes are called miners and the PoW procedure is called mining. The consensus requires that the calculated value has certain characteristics that depend on the rules of the system, e.g. start with three zeroes. This value depends on the content of the block. To obtain a valid result, the nodes change a nonce that is added to the hash calculation, until the targeted value is reached. When one node obtains a correct value, all other nodes must mutually confirm that the value follows the set of rules to be valid. After the confirmation, the transactions in the new block are validated and the miner creates a new block in the blockchain.

In a decentralized network, valid blocks might be generated simultaneously when multiple nodes find a suitable nonce nearly at the same time. As a result, branches (also called forks) may be generated. In PoW, the longer chain is considered the authentic one. After a certain number of new blocks are appended to the blockchain, it is nearly impossible to reverse the blockchain to tamper with the transactions.

In PoW, miners do a lot of computer calculations, wasting a substantial amount of energy [10]. This algorithm tolerates an adversary with up to 1/2 of the computing power of the network. Since the nodes are not expected to be known to the whole network, this algorithm is suitable for public blockchains [13].

**Proof of Stake** (PoS) is an energy-saving alternative to PoW. This algorithm requires the clients to prove the ownership of a certain amount of currency. It is believed that clients with more stake would be less likely to attack the network. The selection based on the account balance is quite unfair because the single richest client is bound to be dominant in the network. Comparing with PoW, PoS saves more energy [10]. A user would need to be the owner of more than 1/2 of the stake of the network to be able to control it. With this algorithm, like PoW, the nodes are not expected to be known to the whole network, so it is also suitable for public blockchains [13].

**Practical Byzantine Fault Tolerance** (PBFT) [16] is a consensus protocol that can be used for state machine replication. The service is modeled as a state machine that is replicated across different nodes in a distributed system. Each state machine replica maintains the service state and implements the service’s operations. The replicas move through a succession of configurations called views. In a view, one replica is the primary and the others are backups. View changes are carried out when the primary fails. The replicas must be deterministic and
they must start in the same state. The algorithm starts with a client sending a request to invoke a service operation to the primary; the primary multicasts the request to the backups; the replicas execute the request and send a reply to the client; being $f$ the number of failures, the client waits for $f+1$ replies from different replicas with the same result; this is the result of the operation. Applying this algorithm to the blockchain, a new block is determined in a round. In each round, a primary would be selected according to some rules and it is responsible for ordering the transaction. The whole process could be divided into three phases: pre-prepared, prepared and commit. In each phase, a node would enter the next phase if it has received votes from over $2/3$ of all nodes. PBFT requires that every node is known to the network [10].

Raft [17] is a crash fault-tolerant (CFT) ordering service based on an implementation of the Raft protocol. Raft follows a “leader and follower” model, in which a leader is dynamically elected among the ordering nodes in a channel, and that leader replicates messages to the follower nodes. The system can sustain the loss of nodes, including leader nodes, as long as there is a majority of ordering nodes remaining. Raft nodes are always in one of three states: follower, candidate, or leader. All nodes initially start out as a follower. In this state, they can accept log entries from a leader or cast votes for a leader. If no log entries or heartbeats are received for a set amount of time, nodes self-promote to the candidate state. In the candidate state, nodes request votes from other nodes. If a candidate receives a quorum of votes, then it is promoted to a leader. Raft implements consensus by first electing a distinguished leader, then giving the leader complete responsibility for managing the replicated log. The leader accepts log entries from clients, replicates them on other servers, and tells servers when it is safe to apply log entries to their state machines.

PBFT is the most economical in terms of energy consumption if we compare to PoW and PoS and can handle up to $1/3$ malicious byzantine replicas. With this algorithm, the nodes’ identities are expected to be known to the whole network, so it is preferred for consortium and private blockchains. Raft, as a crash fault-tolerant ordering service, is not designed to support byzantine faults, but, according to Hyperledger Fabric, this protocol is the first step for the development of a byzantine fault ordering service [17].

2.5 Examples of Blockchains

This section presents examples of blockchain platforms. We will introduce Ethereum, FileCoin and HyperLedger Fabric. Hyperledger Fabric will receive a bigger focus since it is the one used in Hypercloud.
2.5.1 Ethereum

Ethereum [14] is a blockchain platform used to develop decentralized applications and it was built on a public and permissionless blockchain. It has a native currency called Ether and every computation performed in Ethereum is subjected to fees. Those fees are applied to prevent attacks and abuses on the network and are expressed in gas units. Ether can be exchanged for gas. Every operation has a specific amount of gas associated with it. A miner will choose the minimum gas price for which he will execute transactions and the sender of a transaction will pay the fee in gas units to him. In Ethereum, a transaction is seen as a single cryptographically-signed instruction constructed by an external actor to Ethereum (e.g. a human).

Ethereum uses the PoW consensus. It works as a method of securing that the blockchains state will remain correct and, since mining a new block comes with an attached reward to the miner, as a wealth distribution mechanism [14].

2.5.2 Filecoin

Filecoin is a decentralized storage network (DSN) built on a blockchain and with a native token. Clients spend tokens to store and retrieve data and miners earn tokens by storing and serving data [18].

In the Filecoin protocol, storage providers must convince their clients that they stored the data they were paid to store. To achieve this, storage providers will generate Proofs-of-Storage that the blockchain network or the clients will verify. A Proof-of-Storage allows a user to repeatedly check if the server is storing its data. This allows the user to verify the integrity of the data outsourced to a server in a very efficient way, even more efficiently than downloading the data. A Proof-of-Replication is a type of Proof-of-Storage which allows servers (prover P) to convince a user that some data D has been replicated to its own uniquely dedicated physical storage. P commits to store n distinct replicas (physically independent copies) of some data D, and then convinces the verifier V, that P is indeed storing each of the replicas via a challenge/response protocol. A Proof-of-Spacetime scheme allows storage providers to prove efficiently that they stored some data for some period of time [18].

2.5.3 HyperLedger Fabric

Hyperledger Fabric [15] is an open-source permissioned distributed ledger established under the Linux Foundation. It was designed for use in business contexts. HyperLedger Fabric supports smart contracts written in general-purpose programming languages like Java, Go, and Node.js. 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, removing completely the possibility to perform attacks aiming at the cryptocurrency properties of the system. The absence of cryptographic mining operations means that the platform can be deployed with roughly the same operational cost as any other distributed system since there is no need for the expensive mining process [19].

A Fabric blockchain consists of a set of nodes that form a network. All of those nodes have an identity provided by a modular membership service provider (MSP). Nodes in a Fabric network can have one of three roles [15]:

- Client: those nodes sign and send transaction proposals to the peers specified by the endorsement policy and broadcast the transactions to the ordering service.

- Peers: Those nodes execute transaction proposals and validate transactions. All peers maintain the blockchain ledger. Only a subset of the peers, the endorsers, execute all transactions. The peers can be:
  - Endorsing peers [20]: that receive the transaction proposal. They execute the chain-code and generate a transaction proposal response that is signed by the peer;
  - Committing peers [21]: that validates blocks of ordered transactions and appends them to its local copy of the ledger;
  - Anchor peers [22]: those peers enable communication between peers of different organizations;
  - Leading peers [22]: This is a node that communicates with the network ordering service on behalf of the organization.

- Ordering Service Nodes: OSN are the nodes that collectively form the ordering service. The ordering service establishes the total order of all transactions in Fabric. Orderers do not participate in the execution or in the validation of transactions.

The blockchain network can include various actors like clients and orderers. Each of these actors has a digital identity [23] encapsulated in an X.509 digital certificate. These identities determine the exact permissions of the actor over the resources and information in the blockchain. For an identity to be verifiable, it comes from a trusted authority, the Membership Service Provider (MSP).

The Membership Service Provider identifies which CAs are trusted to define the members of a trusted domain. The MSP does it by listing the identities of their members, by
identifying which CAs are authorized to issue valid identities for their members or a combination of both. An MSP, in addition to listing who is a network participant, identifies specific roles an actor might play in the scope of the organization and represents and define its access privileges in the network. The MSP also allows the identification of a list of identities that have been revoked [24].

A smart contract defines the rules between different organizations in executable code and is defined within a **chaincode**. Every chaincode is associated with an endorsement policy that is applied to every smart contract within it. An endorsement policy identifies which organization must approve transactions generated by a given smart contract for that transaction to be declared valid [25].

**Transaction Flow**

The transaction flow can be divided into three parts: Execution, Ordering, and Validation.

In the Execution phase [15], the clients sign and send the transaction proposal to one or more endorsers for execution. A proposal contains the identity of the submitting client, the transaction payload in the form of an operation to execute, parameters, and the identifier of the chaincode. The chaincode's identifier is a nonce to be used only once by each client and a transaction identifier derived from the client identifier and the nonce. The endorsers simulate the proposal, by executing the operation on the specified chaincode, which has been installed on the blockchain. As a result of the simulation, each endorser produces a value writeset, that is the state updates produced by simulation, as well as a readset, that represents the version dependencies of the proposal simulation. The client collects endorsements until they satisfy the endorsement policy of the chaincode. Then, the client proceeds to create the transaction and passes it to the ordering service when it has collected enough endorsements on the proposal. The transaction contains the transaction payload, transaction metadata, and a set of endorsements.

The Ordering phase establishes a total order on all submitted transactions per channel. The ordering service batches multiple transactions into blocks and outputs a hash-chained sequence of blocks containing transactions. This helps in improving the throughput of the broadcast protocol.

The Validation phase consists of three sequential phases:

- **Endorsement Policy Evaluation** [15]: the evaluation validates the endorsement concerning the endorsement policy configured in the chaincode;

- **Read-Write Conflict Check** [15]: for each transaction, the peers compare the version of the keys in the readset field to those in the current state of the ledger to ensure they are still
the same;

- Ledger Update Phase [15]: in this step, the block is appended to the locally stored ledger and the blockchain’s state is updated.

The Ledger

In Hyperledger Fabric, a ledger [26] consists of a world state and a blockchain.

A world state is a database that holds the current values of a set of ledger states. The world state makes it easy for a program to directly access the current value of a state without the need to calculate it by analyzing the entire transaction log. The world state can change frequently, as states can be created, updated, and deleted.

A blockchain is a transaction log that records all the changes that have resulted in the current world state. Transactions are collected inside blocks that are appended to the blockchain. The blockchain data structure is very different from the world state because once written, it cannot be modified.

Blocks

A block [27] is composed of three sections: block header, block data, and block metadata.

The block header has three fields that are a derivation of the hash of the block: a block number, a current block hash, and a previous block header hash. The block number is an integer that starts at zero and increases by one for every new block appended to the blockchain. The current block hash is the hash of every transaction in the current block. The previous block header hash is the hash from the previous block header.

The Block Data contains a list of transactions.

The Block Metadata contains the certificate and signature of the block creator, a valid/invalid indicator for every transaction in a bitmap, and the hash of the cumulative state updates up until and including that block. The certificate and the signature are used to verify the block and the hash is used to detect a state fork.

Transactions

A transaction [28] is composed by:

- A Header that has some essential metadata about the transaction like the version and name of the chaincode used.
• A Signature that contains a cryptographic signature created by the client application. This field is used to check that the transaction details have not been tampered with.

• A Proposal that encodes the input parameters supplied by an application to the smart contract which creates the proposed ledger update.

• A Response that is the before and after values of the world state. It’s the output of a smart contract, and if the transaction is successfully validated, it will be applied to the ledger to update the world state.

• The Endorsements that is a list of signed transaction responses from each required organization sufficient to satisfy the endorsement policy.

2.6 Summary

In this section, we introduced the blockchain technology, a distributed ledger that contains every single transaction made in the system and that information cannot be erased. We presented the concept of consensus and four protocols: PoW, PoS, PBFT, and RAFT. There are different types of blockchain and some examples of platforms are Ethereum, Filecoin, and HyperLedger Fabric.

What gives the blockchain its data integrity and security characteristics is the existence of a distributed and public ledger that cannot be modified or deleted after the data has been approved by all nodes. This characteristic eliminates the necessity of trusting a central entity since the performed operations are signed by the authors and can be logged into a ledger that cannot be tampered with. The ledger is organized in blocks and each block has a reference to the previous block. That improves the transparency of the stored data. The blockchain is mostly known as the technology running the Bitcoin cryptocurrency but it can also be applied to other types of users. For example, it allows the creation of an environment for digital contracts and peer-to-peer data sharing in a cloud service [8]. These properties are what allow the construction of a solution that does not need to trust in intermediary entities for the correct execution of the protocols.
Chapter 3

Related Work

In this chapter, we present the related work. We begin with an overview of cloud storage aggregators. In Sections 3.2, 3.3 and 3.4, we present the DepSky system, the SCFS system and HAIL. In Section 3.5 we present Storekeeper system. In Section 3.6, we present the first version of Crypto Cloud, a system based on Storekeeper, a cloud storage aggregator. In Section 3.7, we present the second version of Crypto Cloud, which already integrates the blockchain.

3.1 Cloud Storage Aggregators

Cloud Storage Aggregators aggregate all popular public clouds like Google Drive, Dropbox, and OneDrive. They all require read and write permissions over the user’s files [29].

Odrive [30] is an example of a cloud aggregator. It provides a web application and a desktop application. The desktop application will create an Odrive folder that stays automatically synchronized with the user’s Odrive account. When linking a cloud storage account to Odrive, the user gives read and write permissions on all user’s files stored in the cloud, however, Odrive does not store or see any user credentials (authentication is done against the cloud’s API), the cloud authorization token is stored locally [29]. Odrive, by design, supports a Progressive Sync model where the user syncs down what he needs from the cloud, as he needs it. Odrive uses web links as a sharing mechanism. The user generates a weblink to a file or directory and shares it to anyone, even users that are not using the system. Anyone who receives the web link will be able to view and download the files. Since web links are by default public, Odrive enables the use of passwords to access the shared content. Odrive allows the use of AES-256 keys to cipher files on the client’s side. The ciphered files are saved in the cloud account and only the user can decipher them [30].
3.2 DepSky

DepSky [31] is a Cloud of Clouds Storage System that tries to solve problems of loss of availability, loss and corruption of data, loss of privacy, and vendor lock-in. DepSky deals with the loss of availability by storing the data on several clouds, allowing access to the data as long as a subset of them are reachable. Loss and corruption of data are dealt with by using Byzantine fault-tolerant replication to store data on several cloud services, allowing data to be retrieved correctly even if some of the clouds corrupt or lose it. To solve the loss of privacy, DepSky employs a secret sharing scheme and erasure codes to avoid storing clear data in the clouds and to improve storage efficiency. DepSky addresses the vendor lock-in problem in two ways. Since DepSky does not depend on a single cloud provider, but on a few, data access can be balanced among the providers. Second, DepSky uses erasure codes to store only a fraction of the total amount of data in each cloud. In case the need for exchanging one provider with another arises, the cost of migrating the data will be at most a fraction of what it would be otherwise.

The clouds are used to store data without code execution on the user side. DepSky’s algorithms are implemented as a library on the user side that allows read and write operations on the cloud.

The DepSky’s protocols require a set of $n \geq 3f + 1$ storage clouds where, at most, $f$ of which can be faulty.

Every writer of a data unit, $du$, share a private key used to sign the data wrote in that $du$. The readers of the $du$ have access to the corresponding public key. This public key is used to verify the writers’ signatures.

![Figure 3.1: Architecture of DepSky [31]](image)

DepSky allows data sharing. Every user has to share the same cloud account. To provide
data confidentiality in the cloud, this data is ciphered with a symmetric key. To share the symmetric key, a dealer (writer) distributes the secret key to n players (clouds), where \( n \geq 2f + 1 \). Each player receives only a fraction of the key, so no cloud has access to the complete key and to obtain the secret at least \( f + 1 \) different shares of the key are needed. DepSky’s mechanism uses the cloud provider’s access control. If the client has permission to access the data stored by the cloud, then he will have access to at least \( f + 1 \) different clouds and be able to obtain the complete key to decipher data.

In DepSky’s write algorithm, the writer writes to a quorum of clouds and waits for \( n - f \) acknowledgments, where \( f \) is the number of faulty clouds. When the writer receives enough acks, he writes the metadata and waits for \( n - f \) acks. In the read algorithm, the reader asks every cloud the metadata of a file \( F \). The reader waits for \( n - f \) responses and chooses the response with the higher version number. The reader search in the clouds for the file that hash the same version and digest of the received metadata.

![Figure 3.2: The combination of symmetric encryption, secret sharing, and erasure codes in DEPSKY-CA [31]](image)

DepSky increases data availability by using multiple cloud providers to save the data provides data confidentiality and deals with corrupted data. This system also allows sharing data but the users need to share the same cloud account. Also, for this system to be effective, the user is obligated to have multiple cloud accounts in multiple cloud providers.

### 3.3 SCFS

SCFS [32] is a cloud-backed file system. It solves the following problems: reliability problems like a single point of failure, lack of trust in the cloud provider, and inefficient file sharing. The SCFS goals are:

- Users being able to share files while guaranteeing the files integrity, confidentiality, and
availability;

- Increase data durability;

- Offer an API of a file system with strong consistency;

- Leverage the clouds’ services scalability, supporting large numbers of users and files as well as large data volumes.

SCFS is made of three components:

- Backend cloud storage used for maintaining the data of the file;

- Coordination service used for managing the metadata and to support synchronization (access control and lock control);

- SCFS agent that implements most of the SCFS functionality, and corresponds to the file system the client mounted at the user machine.

SCFS is able to guarantee data integrity, confidentiality, and availability using DepSky’s [31] secret share scheme to distribute secret keys and protect files.

![SCFS Architecture](image)

**Figure 3.3: SCFS Architecture [32]**

SCFS guarantees strong consistency using consistency anchors (CA). The client calculates the objects hash and stores it with the object’s id. Those values are then stored in the CA.

SCFS can function with three modes: blocking, where the service sends a file for the cloud provider immediately after the “close” instruction; non-blocking, where the service uploads the
file asynchronously after the “close” instruction; non-sharing where the sharing mechanisms of the system are deactivated.

SCFS does not trust the cloud provider guaranteeing data confidentiality, integrity, and availability like DepSky. SCFS is a file system that uses cloud providers, so it is not expected to have the same objectives as a cloud storage aggregator.

### 3.4 HAIL

HAIL [33] is a distributed cryptographic system that allows a set of servers to prove to a client that a stored file is intact and retrievable. HAIL manages file integrity and availability across a collection of servers or independent storage services. It makes use of Proofs of Retrievability (POR) as building blocks by which storage resources can be tested and reallocated when failures are detected. HAIL exploits both within-server redundancy and cross-server redundancy. HAIL relies on a single trusted verifier, which is a client or a service acting on behalf of a client, that interacts with servers to verify the integrity of stored files.

Let $L$ be the number of primary servers and $n$ the total number of servers. The client generates the following keys: dispersal-code keys, server-code key, and challenge keys. Dispersal-code keys are assigned to non-primary servers and are created $n - L$ keys. Non-primary servers allow data redundancy. $n$ server-code keys are created and each server is assigned to one key. Challenge keys are used to generate challenges.

To encode a file $F$ in HAIL, the client divides $F$ into $L$ segments, $F_1 \ldots F_L$. The client distributes the segments by the $L$ primary servers, $S_1 \ldots S_L$, respectively. Each segment is encoded with the server code, implementing adversarial codes. Adversarial codes are keyed codes resistant to a large fraction of adversarial corruptions against a computationally bounded adversary. The parity blocks are created using an integrity-protected error-correcting code that allows the verification of integrity blocks and correct errors. These blocks are stored in the secondary servers. Lastly, a cryptographic MAC of the file is computed and stored with the file. The result is illustrated in Figure 3.4.

In the decoding process in HAIL, it is possible to correct $(n - L - 1)/2$ errors in each row. The decode is done firstly by rows then by columns. Each row of the matrix is decoded and the corresponding message is checked for integrity using the MACs embedded in the parity blocks. If the number of corruptions in the row exceeds the error correction capability of the dispersal code, or if none of the MACs in the parity blocks of a row verifies, then all blocks in that row are marked as erasures. In the second step, the server code implemented with an erasure code is used to recover the row erasures from the first step.
The Challenge-Response Protocol is used to verify if a subset of random blocks is correct. The client sends a challenge to all servers. The servers answer with a set line of the subset. The client verifies the validity of the server responses using an MVerECC [33] algorithm. If the client detects more than $\epsilon$ errors, it calls the redistribution algorithm. To redistribute the shares, the client downloads the file shares from all servers and applies the decoding algorithm. When the original file is completely decoded, the client may reconstruct the shares of the corrupted file. After the corruption has been removed from the servers, the client sends them the new shares.

HAIL is a system that allows a set of servers to prove to a client that a stored file is intact and retrievable. The objective of this system is not to allow data sharing between clients.

3.5 Storekeeper

Storekeeper [29] is a cloud storage aggregator that guarantees data confidentiality. This system allows file sharing between users of different cloud providers. This is possible because Storekeeper presents a single storage workspace where all files are localized, independently of its cloud provider. This system does not guarantee integrity nor availability and, in terms of consistency semantics, it allows file versioning.

Storekeeper is composed of the client application and the Storekeeper Directory Server (SDS). The application is installed on the user’s device and it gives an interface to the system. The application maintains a local cache of the user’s files. The SDS generates the meta-data associated
3.5 Identification

When a user signs up for the first time, using the username assigned by the SDS administrator, he defines a password. The Storekeeper client, after reading the username and password, calculates the Login Key (KL) based on those credentials. The client also generates a cryptographic RSA key pair. The client ciphers the username and the user’s private key using the KL. The resulting ciphertext is sent to the SDS along with the user’s public key.

3.5.2 Permissions

Each file has an owner that has complete access to it. A user can have 3 access types:

- Read: can only read the file
- Write: can read and write the file
- Share: can read, write and share the file with other users

Storekeeper allows confidentiality by ciphering the file on the client’s side with a Key File (KF). This key is ciphered with the file owner’s public key. That way the owner is the only user that can access the KF. When the file is shared, the owner generates a symmetric key (Read Key) and uses it to encrypt the KF. The Read Key is encrypted with the public key of the user that the file is being shared. In this way, this new user can have access to the Read Key and he will be able to get the KF.
When a user A modifies a file, he creates a new KF that is used to cipher the file. User A ciphers KF with a Read Key and sends the wrapped KF to the SDS. The RK usage allows the revocation of files in a more efficient way. When a revocation occurs, a new RK is created, ciphered with the reader’s public keys and stored in the ACL. That way, it is not necessary to create a new KF.

3.5.3 Discussion

Storekeeper does not need to have access to the cloud store, those permissions are on the user side, improving the user’s data confidentiality. This system also allows the share of files between users of different cloud providers keeping the confidentiality of the files and provides a more efficient way to revoke permissions with the use of a Read Key to cipher the files. Although this solution solves the problem of data sharing and data confidentiality, it needs to assume that the SDS is well behaved.

3.6 Crypto Cloud

Crypto Cloud [34] is a cloud-based storage system based on Storekeeper. This system implements a new key management system, a new authentication method and access control. It adds integrity to the data stored in Storekeeper, using an Integrity Key to calculate an HMAC of the file.

3.6.1 Architecture

Figure 3.6: Architecture of Cryptocloud [34]
Crypto Cloud can be divided into Crypto Cloud Client Application (CCCA), Crypto Cloud Directory Server (CCDS), Public Key Infrastructure (PKI) and Key Management Server.

CCDS is a revised version of Storekeeper’s SDS. In this new version, the authentication and access control mechanisms were improved. It is used OAuth protocol [35] that is implemented on the CCDS.

The Key Management Server (KMS) implements the KMIP protocol [36]. This is responsible for the access, storage and management of the user’s cryptographic key. This component allows the client application to obtain the user’s key and perform the cryptographic operations locally or send a KMIP request to perform the cryptographic operation remotely.

The PKI works as a third party component. It is responsible for the user’s key certification and identity association of a user to its public key.

**Crypto Cloud Client Application**

The Crypto Cloud Client Application [34] is the central component of the system. This component manages the user’s files and interacts with every other component. The application allows the upload and download of files, the management of its permissions, the generation of a cryptographic key to replace the older one and the creation of a registry request of a new user (the last one requires the admin acceptance). The application is divided between the interface layer and the service layer. The interface layer receives the user’s input, validates it and sends it to the service layer. The service layer is responsible for executing the operations the user inserts in the system. To process the request, the service layer is divided into modules:

- **Session Manager**: preserves the local state of the application, after the successful login in the system.

- **Communication Manager**: interacts with the CCDS. The user authentication is done in this module.

- **Key Manager**: manages the cryptographic key pairs. Works with KMIP’s client API to have access to the cryptographic keys and execute the necessary operations.

- **File Manager**: maintains the workspace that contains the local copies of the user’s files. It is possible to generate, read or write files. It also manages a directory of temporary files and of files that generate conflicts.
Crypto Cloud Directory Server

The Crypto Cloud Directory Server (CCDS) [34] serves the client application. Its main function is to maintain the files’ metadata used to manage them. This allows keeping track of those files in the cloud stores. The CCDS also stores the ACL along with the file’s cryptographic keys. The keys are protected using the user’s keys.

After a successful authentication, the CCDS grants an authorization token that allows access to its resources.

Key Management Server

The Key Management Server [34] is responsible for accessing and protecting the user’s private keys. This component frees the users and the CCDS from the responsibility of managing the user’s cryptographic keys and implements the KMIP protocol. It allows users to generate, obtain, use, and destroy their cryptographic keys. It was implemented a dedicated key storage hardware to generate, access and protect managed cryptographic keys.

Public Key Infrastructure

The Public Key Infrastructure (PKI) [34] component acts as a trusted third-party entity and is responsible for the digital certification and validation of the user’s public keys. This component emits X.509 certificates, signed by the infrastructure, binding the certificate’s public key to the user’s identity. It maintains a repository of revoked certificates (CRL) that allows clients to check if its certificate was revoked before using it.

3.6.2 Cloud Storage

Cloud stores [34] represent the cloud accounts registered on the system by the users. These clouds store files without executing any application logic, that means they are passive stores. The Client App authenticates the user to the cloud system. Then the user receives an access token for future accesses.

3.6.3 File Version

Crypto Cloud implements a version control mechanism [34] in the CCDS. This mechanism relies on the file content hash, which is the digest of the file’s content before encryption, and file version number, which is a number managed by the CCDS. It is incremented every time a users updates the file. The association of both of these elements ensures that a certain file’s content represents a specific file version.
3.6.4 Discussion

Crypto Cloud improves the key management of Storekeeper. The private keys are now stored on a trusted server, the KMS, and the public keys are stored on a trusted third party entity, the PKI. This method removes the responsibility to manage keys from the central server, now called CCDS. While maintaining the features of Storekeeper, this solution ensures the file’s integrity, using an Integrity Key to calculate an HMAC of the file, and improves the key management, removing that responsibility from the central server. The system is still dependent on the correct behavior of the CCDS. This solution also makes the system dependent on two new external entities, the KMS and the PKI.

3.7 Crypto Cloud with HyperLedger

The Crypto Cloud system [37] was extended to remove the need to trust the central server to do sensitive operations. This newer version of the Crypto Cloud uses HyperLedger Fabric to achieve that goal. Metadata on the Crypto Cloud system is associated with the users, clouds, files and permissions. The blockchain will ensure the data on the CCDS has not been tampered with. This new solution shows a new way of key management and modifies the protocols to interact with the files stored in the cloud providers, adding the blockchain.

The system can be divided into Crypto Cloud Client Application, Crypto Cloud Directory Server and HyperLedger Fabric Network.

Figure 3.7: Architecture of Cryptocloud with Hyperledger Fabric [37].
3.7.1 Crypto Cloud Client Application

The Crypto Cloud Client Application [37] is the central component of the system. This component manages the user’s files and interacts with every other component. The application allows the upload and download of files, the management of its permissions, the generation of a cryptographic key to replace the older one and the creation of a registry request of a new user (this requires the administrator approval). The application allows the client to communicate 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. The retrieved information is then compared with the stored on the server. In this version of the Crypto Cloud, the key pair is generated by the client and stored on its device. With this modification, the cryptographic operations are performed on the user’s device. The other modules are equal to the previous version.

3.7.2 Crypto Cloud Directory Server

The Crypto Cloud Directory Server [37] 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. We can divide the metadata into five groups, based on their function:

- The first group is formed by the metadata of the users;
- The second group includes information about the users’ Cloud Stores;
- 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”);
- The fourth group includes the metadata used to implement the Access Control mechanism;
- The fifth group includes the metadata related to the managed files.

3.7.3 Management of identities

On the Crypto Cloud system, the identity is the mapping between the user and its public key. In Hyperledger Fabric, the concept of identity also exists. The Fabric system generates a key pair that is associated with a user. This key is used to verify and sign transactions.

Each user has two different identities. The user’s public key is published on the blockchain, so everyone can verify that a user has that specific identity. Since no one can change the content of a transaction published on the blockchain, the identity and public key relationship is
permanently registered and accessible by every user. When a user performs the first registration to the system, a transaction will be created that contains the binding between the Crypto Cloud identity and Hyperledger identity. That transaction is signed with the user’s RSA Private Key of Crypto Cloud. When a file related transaction is performed, the client will verify if the identity of the transactor is valid. It is assumed that the user’s identity corresponds to the first transaction on the blockchain to a specific username [37].

3.7.4 Access Control List Integrity

In this version, the server no longer needs to be trusted, so it may tamper with the Access Control List. Because of this, it is necessary to check if the list is valid.

To solve the problem of an untrustworthy server, the client who updates the file calculates the hash of the access list and publishes 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. When a user performs a share operation, the ACL must be updated. To do this, the user who will update the list calculates the hash of the old version of the ACL and the hash of the new version [37].

3.7.5 Version Control and Integrity

In this version, and since blockchain guarantees integrity, it is possible to put the hash of the file and version on a transaction. Therefore every time the user wants to read a file, the application checks the correspondence between CCDS’ metadata and File Metadata transaction [37].

3.7.6 Discussion

Crypto Cloud with Hyperledger presents a way to verify if the CCDS is behaving correctly using the Hyperledger Fabric’s technology. To verify the data from the CCDS, the client compares the information received from the server and compares it with the one received from the blockchain. This solution can detect if the server is not behaving correctly but still has it as a part of the system’s architecture and its correct behavior is needed for the functionalities to be successful.

3.8 Discussion

In this chapter, we presented the Related Work. These systems present solutions to solve problems of data sharing, data confidentiality, data integrity, and data availability, but all present some limitations. DepSky and SCFS are only applicable if the users have multiple accounts in
multiple cloud providers and data sharing is only possible between users that share the same account. HAIL is a system that aims to provide integrity of the data, not focusing on the problem of data sharing. Storekeeper is dependent on the correct behavior of the central server. The first version of Crypto Cloud proposes an architecture that removes the key management responsibility from the central server and gives it to a KMS with an HSM. This solution keeps the cryptographic material from the users safe in a trusted entity, but the protocols still depend on the correct behavior of the CCDS. If it starts to have an incorrect behavior, the system has no way to verify if the CCDS’s data is correct. The second version of Crypto Cloud proposes a solution that allows the client application to verify if the data that the central server is returning is correct and did not tamper using the blockchain technology. This solution solves the problem from the last version since it is possible to verify the data that the CCDS returns with the stored in the blockchain, but the system still fails to complete the tasks if the central server starts to behave incorrectly.
Chapter 4

Hypercloud

In this chapter, we will present Hypercloud, our proposed solution for the problem introduced in chapter one. In Section 1, we describe the assumptions that we are considered during the design of this solution. In Section 2, we present the proposed architecture and detail each component. In Section 3, we present the main protocols. In Section 4, we summarize the chapter.

The solutions that were presented in chapter three do not solve our problem in its totality. The most complete solutions to fulfill our goal are the two versions of Crypto Cloud, thus the proposed solution is based on these two systems.

4.1 Assumptions and System Model

This section presents the assumptions used to design Hypercloud. Our main objective is to design a system that allows the users to have a global view of their files stored in multiple cloud accounts and that does not need to trust in intermediate entities, like the central server in Crypto Cloud. A user should be able to share the files with other Hypercloud users. The user should not be obligated to give access to his account to any of the system’s entities and the data must remain confidential. The user must be able to verify the integrity of his files. The Hypercloud does not focus on the availability of the files because they are saved in the cloud providers which we do not control. The assumptions are the following:

- The machine of the user and the KMS are trustworthy. The data stored in these entities is available, will not be tampered with by malicious users or malfunctions of the machines, and will remain confidential.

- The central server may have a byzantine behavior. We assume that an entity is having a byzantine behavior when it is not following the protocols. The central server may read and alter the data it is keeping, it may fail, and it may not execute the commands correctly.
• The cloud providers are not trustworthy. The cloud providers may read and modify the user’s files, but we assume the cloud providers are always available and will return a response.

• The communication channels are not secure. The channels, by default, do not guarantee confidentiality, integrity nor authenticity of the exchanged data. They are vulnerable to threats like replay attacks or man-in-the-middle.

• The cryptographic algorithms are sound.

• The blockchain will keep the data unaltered and will be always available. This is a reasonable assumption because, as explained in Section 2.4 the main advantage of blockchain technology is to guarantee the integrity of the managed data if the conditions for consensus can be met.

• The blockchain’s network may have malicious nodes. That means it may try to read the managed data, may plot with byzantine clients, or be in the machine of a client. The number of malicious nodes will not be high enough to fail the conditions of the consensus, as stated in the previous point.

• A client that received permission to write or share a file of another client will not act maliciously by tampering with the file content or its metadata. We assume that the owner of the file will only give access to users he trusts so every action the new users do to that specific file we see as expected ones.

• There are no concurrent operations. Only a client is performing an operation at a time.

• There are no Denial of Service attacks.

4.2 Architecture

This section presents the architecture of the Hypercloud, a blockchain-based secret management in multi-cloud storage platforms. This solution aims to guarantee data confidentiality and the possibility to verify the integrity of the data retrieved from the cloud, all without using a central entity. This system is composed by the user, the client application, the KMS, the cloud providers, and the fabric network.

The user is the person using the system. He or she interacts with other entities using the client application. The client application is installed in the user’s machine and runs the needed operations to complete the functionality the user wants. This entity communicates with the
KMS, the fabric network, and the cloud providers. The KMS manages the user’s keys and in the environment considered in our approach, it is running in the user’s machine. This approach allows the design of a solution where the users do not need to trust in any machine besides their own. The cloud providers are the providers themselves, e.g. Dropbox. The fabric network’s entity is the Hyperledger Fabric’s blockchain. It is a network composed by organizations, peers, orderers, and certificate authorities. The communications between the entities are done using TLS channels. This protocol is widely used and guarantees data confidentiality, integrity, and authenticity.

Each entity is responsible for the management of a set of data:

- The user is responsible to keep his username and password to use the Hypercloud system. The password needs to remain confidential.

- The client application is responsible to keep a local copy of the keys of the user. Each user has an ECDSA key pair, an RSA key pair, and an AES key. The client also manages the LToken file and the LFiles file and keeps a local version of those files. The LToken keeps the tokens of the cloud accounts of the user and the LFiles maps the filename to the remote URL in the user’s cloud account and the local version of the file.

- The KMS generates the RSA key pair and stores the user’s keys. In the environment of this solution, the KMS is running in the user’s machine. This approach allows the design of a solution where the users do not need to trust in any machine besides their own.

- The cloud providers are responsible to keep the users’ files.

- The fabric network is responsible to manage the users’ LFiles and LToken and the files’ ACL. The ACL is composed of:
  - A filename;
– A version number ACLver
– An URL to the file
– A file version
– A read keys RK
– A File key FK
– A hash of the file and its version
– A list of permissions to access the file

The list of permissions is a list that maps the username to the type of permission he has to the specific file. The fabric network is also responsible for that guarantee the information remains correct and enforce the access control.

**Keys**

Hypercloud uses 7 different keys.

– The RK is an AES key used to cipher FK. It is ciphered by a user with share permission over a file with the RSA public key of all the users with access to that file;

– The FK is an AES key used to cipher the file and its metadata;

– The ECDSA key pair PU_{BC} and PK_{BC} is used by the user to sign the transactions he sends to the fabric network;

– The RSA key pair PU and PK is used by the user to decipher the RK of the files he has access to or by the other user (a user B) who want to give access to a file to a user A, ciphering RK with the RSA public key of the user A;

– The key K is an AES key used by the owner of the key to cipher his LFiles and LToken.

### 4.3 Protocols

In this section, we will present Hypercloud’s protocols and what is their function in the global view of the system’s.

Hypercloud has three main protocols: write to a file, read a file from the cloud, and update permissions. These three protocols allow the users to interact directly with the files and we believe that these are the most important in this system, because they will be the ones that are used the most by the users.
There are other functionalities that work as auxiliaries to these main protocols:

- The Register User and Login are used to generate a new identity for a user and to authenticate the user to the system, respectively. These functionalities are needed because the users need to have an identity to assign permissions and this identity can only be used by one user.

- The Synchronize Files is the solution we found to maintain the user’s metadata about the permissions he has to the files as consistent as possible without the need to trust other entities. With this approach, the user may not have the latest information about the permissions he has to the files, but he will eventually have once he runs the file synchronization functionality. The blockchain property that guarantees the data once inserted in the ledger is not deleted allows us to guarantee that once the user finds more recent data, he will not receive older information later. This functionality also helps the permission assignment, because it lets the user know that he gained or lost permissions, but it also helps the write and read operations because the user needs to know he has permission to a file to download it and to update it.

- The Add Cloud Account and Remove Cloud Account are used to give or remove the access to a cloud account to the Hypercloud application. These functionalities are essential so the user can start to upload and download files.

To initialize the system, we need to register a new user, log in to the system, and add a cloud account. After this, we can perform the rest of the functionalities that we will describe in the following sections.

4.3.1 Read operation

The read operation is used every time a user wishes to download a file from the clouds. The main steps performed by this operation are:

- Get the ACL from the fabric network: the client application sends a read request to obtain a specific ACL stored in the world state;

- Verify the version: the client accesses the ACL to obtain the most recent version number of the file and compares it with the one saved in his LFiles. If the local file has a lower version, then continues the operation, otherwise, the client has the most recent version;

- Download the file: the client accesses the URL saved in the ACL and downloads the file from the cloud;
4.3.2 Write Operation

A write operation that sends new files or updates an existing file in the cloud. This operation generates or updates an ACL in the fabric network. The main steps performed by this operation are:

- Encrypt the file: Generate an FK and cipher the file that will be sent to the cloud.
- Upload file: The client sends the ciphered file to the specified cloud account. If the file is new, the cloud provider will generate a shareable link. If not, the client just overwrites the file.
- Upload ACL: The client adds all the metadata to the ACL and sends a transaction to the fabric network. The fabric network verifies if all the information in the ACL is valid and if the user has permission to perform the operation.
- Update LFiles: If the file is new, the client adds the entry corresponding to the file, otherwise, he only updates the local version. He ciphers LFiles and creates a transaction to register it in the blockchain.

4.3.3 Permission Update Operation

These operations are the ones that interact with the permissions of a file. This interaction is done by the user but needs to be accepted by the fabric network. These operations may be to share a file, revoke the share of a file, or just modify the type of permission of a user to a file. Assuming user A is the owner of a file F and wants to edit the permission of user B, the main steps performed by this operation are:

- Cipher RK: To have access to a file, a user must have access to the RK. To give access, user A ciphers RK with the PU of user B that he obtained from the fabric network. To completely remove the access, user A generates a new RK that is not ciphered with user B’s PU;
- Update permission list: the client of A updates the permission list in the ACL, adding B with the new permission if gained access, or removing him from the list if he lost access.
• Update ACL: The client of A updates the ACL sending a transaction to the fabric network.
  The fabric network verifies if A has permission to perform the operation and if the ACL’s
  values are valid.

4.4 Summary

In this section, we presented the assumptions and system model of Hypercloud. The new
architecture, comparing to the Crypto Cloud’s, removes the central server, the KMIP server
and PKI, and only adds the fabric network that will ensure the managed data is modified by
valid operations. Finally, we presented the three main protocols: read a file, that allows the
user to have access to content of the file; write a file, that allows the user to create or update a
file; and edit permissions of a file that allows the users to share a file or remove the access to a
user. In the next chapter we discuss Hypercloud’s implementation.
Chapter 5

Implementation

In this section, we will explain how we implemented Hypercloud. In Section 1, we will present the implemented architecture. In Section 2, we will explain the data structures. In Section 3, we explain how the functionalities work and how this solution achieves the objectives. In Section 4, we summarize this chapter.

5.1 Architecture

In this section, we will explain how the prototype for the Hypercloud solution was built and which technologies were used.

Most of the prototype of Hypercloud was implemented using Java 11 [38] because it is a widely used language with good documentation available. Some components of the prototype are written with javascript [39], YAML [40], and bash [41]. Javascript is used to write the chaincodes and we chose this language since it is an already familiar one from the set of languages the peers support. YAML is used to configure the components of the fabric network. Bash is used to perform operations in the components, like starting or stopping the network.

The prototype of the Hypercloud is composed by the cloud provider with the accounts of the users, the fabric Network, a KMS, and the client application, both running locally in the user’s device.

The KMS is an HTTPS server that implements a REST endpoint. It uses the library com.sun.net.httpserver that allows us to create easily a REST service. This service allows the client applications to interact with the data managed by the KMS only if a secure channel was created between both entities. In this channel, the client can guarantee that he is communicating with the correct KMS and the KMS can execute the operations of the client after he is registered in the system, and authenticates himself successfully with his username and password. The KMS
manages a MySQL database where it stores the following data about a user:

- Username,
- Password
- RSA key pair
- AES key
- ECDSA key pair

This database is accessed using JDBC [42]. JDBC is a Java API that allows a Java application to send SQL instructions to the database.

The fabric network is composed of 2 organizations, Org1 and Org2. Each organization is composed of a certificate authority and 2 peers. Each peer uses CouchDB to maintain the world state. The fabric network has an orderer. Each peer has 4 chaincodes installed that are invoked by the client application when sending a transaction to the network.

![Figure 5.1: Client application modules](image)

Figure 5.1: Client application modules

Figure 5.1 shows the composition of the client application. The client application is composed by the console, the service module, the communication module, and a server to obtain the access token from the cloud provider. The console is used by the user to interact with the client application. It is in this component where the user chooses which functionality to execute, inserts the arguments, and receives the results. The service module is responsible to execute the various operations of the functionalities and to interact with the communication module. The communication module is responsible to exchange messages with the other entities and it can be divided into three smaller modules: the fabric communication, the KMS communication, and the communication with the cloud. In the fabric communication module, the client
sends transactions, invokes functions of a specific chaincode, and searches for information in the blockchain. The KMS communication is used to exchange requests and responses with the KMS. The Cloud communication module is used to communicate with the cloud providers through the respective API. The client application interacts with the KMS through REST endpoints. The exchanged requests and responses are encoded in JSON and sent through a secure TLS channel.

We use Dropbox’s APIv2 [43] to interact with the Dropbox. The cryptographic operations are done using the native Java libraries java.security and javax.crypto. The client application uses symmetric and asymmetric cryptography. To perform symmetric cryptography, the client uses an AES key with 128 bits, and the cipher is calculated with the CBC mode with PKCS5 padding. To perform asymmetric cryptography, the client uses a pair of 512 bits RSA keys, and the cipher is calculated through ECB with PKCS1 padding. To calculate the hashes, this prototype uses the SHA-256 algorithm. The client application uses the ECDSA key pair to sign the transactions that are sent to the fabric network. The communication and the cryptographic operations between the client and the fabric network are done using version 1.4 of the Hyperledger Fabric’s Java API. This API allows the client application to send transactions and invoke specific methods of a chaincode installed in the peers.

5.2 Data Structures

In this section, we present the data structures used in Hypercloud.

For the system to work correctly, it is necessary to maintain the state of the files and the user’s data. The user interacts with the system through the client application. The client application entity is not reliable to keep the user’s data in a permanent manner, because this data is only guaranteed to be correct while the session is active. A session begins when a user logs in and ends when the application are stopped or other user logs in with different credentials. The client application needs to communicate with the fabric network to obtain the present state of the system, that is the final result of all operations performed by the clients. In Hyperledger Fabric the present state of the system is called the world state. In Hypercloud, the world state is composed by the files’ ACLs, the LFiles and LTokens of the users, and the users’ RSA public keys, totalizing 4 tables in the CouchDB database. This data is stored in a CouchDB database that can be accessed and managed efficiently. The ACL is composed of a JSON file that is kept by the peers. The chaincode acl-contract verifies if the transactions to modify the ACL are valid, guaranteeing that the data stored can be trusted. The ACL is composed by:

- Filename;
• The version of the ACL;
• The URL of the file in the cloud ciphered with FK;
• The version of the file ciphered with FK;
• A list of RKs that maps the username to the RK ciphered with the user’s RSA public key;
• The FK ciphered with RK;
• The hash of the file after it is ciphered with FK with the version. The hash is ciphered with the FK;
• A field that has the name of the last operation done to the ACL. This field is useful when searching for the blocks of a specific operation. E.g. synchronize files searches for blocks that correspond to the operations Share File and Revoke Share.

In the blockchain, nothing is deleted so it is possible to get all the past transactions and verify which operations modified the ACL. The possible operations to perform over an ACL are written in the chaincode and are the following:

• The creation of a new ACL;
• Update an existing ACL;
• Read an ACL;
• Share a file;
• Remove the permission of a user to a file.

The LFiles is a JSON file that maps the name of the user’s files to the local version and its shareable link. The LFiles are managed by the chaincode `lfiles-contract`. This chaincode is only used to save the new data that corresponds to the LFiles in the blockchain. The LFiles is saved in the blockchain ciphered with the user’s AES key.

The LToken is a JSON file that maps the name of the cloud account to its access token. The LToken are managed by the chaincode `ltoken-contract` that has a similar function to `lfiles-contract` but to the LToken. The LToken is saved in the blockchain ciphered with the user’s AES key.

The users table is composed by the JSON files that map the username and the user’s RSA public key. These files are managed by the chaincode `users-contract` that allows the registration of a new user and to get the public keys of other users registered in the system.
The `user_info` table is in the KMS’s database. This table contains the username, password, RSA key pair, ECDSA key pair, and AES key.

To interact with the JSON files, this solution uses the Gson library [44]. This library allows us to easily create and interact with JSON objects. To exchange messages between the KMS and the client, we use the Spring and Apache APIs. The Spring API facilitates the creation of REST requests and the Apache API allows the generation of a secure channel between both entities. To interact with the fabric network this solution uses the version 1.4 of the Hyperledger Fabric’s API. To apply the cryptographic operations, Hypercloud uses the native libraries of Java called `java.security` and `javax.crypto`. To download files from the cloud the client uses Apache Commons IO Library v1.3 [45]. This is an API that offers methods to manage files in Java.

### 5.3 Functionalities

In this section, we will present the functionalities of Hypercloud in detail. The functionalities of this system are:

- Register User;
- Login User;
- Add Cloud Account;
- Delete Cloud Account;
- Upload New File;
- Update File;
- Download File;
- Share File;
- Revoke Share;
- Synchronize Files

Some functionalities are similar but we chose to divide into two distinct ones because it seems easier from the user perspective. We can consider that the main functionalities are: Register User, Login User, Update LToken, Update LFiles, Download File, Update File, and Update ACL’s list of permissions. For each main functionality, we will describe their goal at a high level. Then we will present a diagram for the functionality and to finish we will explain how
the functionality works, present the possible approaches, and the rationale to choose the current solution.

5.3.1 Register user

This functionality is used to add a new user to the system, creating a PU, PK, K, PU_BC, PK_BC, LFiles, and LToken as described below. The user must know a username U that is not already being used and a password pass.

1. The user inserts U and pass in the client application. The client sends a request to add the new user to the KMS. The KMS generates an RSA key pair, PU and PK. Both the KMS and the client store the new keys.

2. The client application generates and saves an AES key K, ciphers K with PU, and stores it in the KMS.

3. The client application generates an LFiles and LToken and ciphers them with K. After the client registers the new user in the KMS, he will register the user in the fabric network. The client application sends a transaction to generate a new user in the fabric network with username U and that is the owner of the public key PU. The fabric network maps

Figure 5.2: Register User

1. The user inserts U and pass in the client application. The client sends a request to add the new user to the KMS. The KMS generates an RSA key pair, PU and PK. Both the KMS and the client store the new keys.

2. The client application generates and saves an AES key K, ciphers K with PU, and stores it in the KMS.

3. The client application generates an LFiles and LToken and ciphers them with K. After the client registers the new user in the KMS, he will register the user in the fabric network. The client application sends a transaction to generate a new user in the fabric network with username U and that is the owner of the public key PU. The fabric network maps
the received information and generates an ECDSA key pair for the client, \( PU_{BC} \) and \( PK_{BC} \).

4. The client saves the keys in his wallet and sends a request to the KMS to save his ECDSA keys.

We use the AES key to cipher and decipher the LFiles and LToken because symmetric encryption is faster than asymmetric encryption and since those files are not shared with anyone else, there is no need to use the key pairs to sign the files.

After the execution of this functionality, the user is able to login into the system. The KMS has the AES, RSA key pair and ECDSA key pair of the user. The fabric network has the user’s RSA public key registered in the world state and the LToken and LFiles in the ledger.

5.3.2 Login user

This functionality is used to verify if the client application has access to all keys of a user that is already registered in the system. If any key is missing, the client will retrieve it from the KMS. Also, this functionality is used to obtain \([LFiles]_K\) and \([LToken]_K\) from the Fabric Network.

1. The user inserts his username and password in the client application. The client application verifies if any key is missing and, if it is, requests the missing keys from the KMS. The KMS verifies the username and password of the user and, if correct, returns the requested keys. The client sends a transaction to verify the ledger, looking for the most recent update on the user’s \([LFiles]_K\) and \([LToken]_K\) and returns them. In the end, the client application executes the synchronize files functionality. This functionality verifies if there is a modification in the user’s permissions to the files managed by the system.
2. The client application sends a request to obtain the latest version of his LFiles and LToken from the blockchain. The peers receive the request and search in every block for the last transaction made by U to change LFiles and LToken. The transactions are signed with the PK_BC of U, so it is possible to verify who made the transaction.

After the execution of this functionality, the user is able to manage his cloud accounts and the saved files. All the other entities maintain the same state.

### 5.3.3 Update LToken

The operations that update the LToken are the addition and removal of cloud accounts.

![Figure 5.4: Add Cloud Account](image)

To add a new cloud account:

1. The user inserts the name of the cloud account in the client application. To guarantee that the user’s credentials remain confidential, the user authenticates himself through the cloud provider itself, no entity will have access to the account’s password. After a successful authentication, the user must give access to the Hypercloud application to access his files. After this process, the cloud provider returns an authorization code to the client application.

2. The client application sends the authorization token to the cloud provider to obtain the reusable access token `Token`.

3. The client adds the token to LToken.
To remove a cloud account, the user inserts the name of the cloud account to remove in the client application and it removes the respective token from LToken.

To update the LToken in the Fabric network, the client application ciphers the updated LToken with the user’s key K. The client application sends a transaction to the Fabric network to update \([\text{LToken}]\_K\). The Fabric network verifies if the transaction is correct, was not executed in the past and the signature is valid. If this verification is successful, the Fabric network updates \([\text{LToken}]\_K\).

Another option to store LToken in the Fabric network could be the client application sending a transaction to update \([\text{LToken}]\_K\) and every peer would validate the transaction, verifying if the user sending the update is the user whose the LToken belongs to. We did not choose this option because the LToken is only accessed by the owner and will never be shared. Because of this, only the owner of the list needs to validate the data and there is no need for other elements of the network to participate in the process, slowing it down.

After the execution of this functionalities, if the user added a new token, he is able to save files to the respective cloud account. If the user removed a token, he loses the access to the cloud account. The fabric network saves a new version of the LToken in the ledger.

5.3.4 Update LFiles

The operations that update the LFiles are the upload of a new file and synchronization of files.

To upload a new file:

1. The user chooses the file to upload and the client application generates an RK and an FK.
   The client application ciphers the file with FK;
2. The client application sends the ciphered file to the cloud provider
3. the client application generates an ACL inserting:
   - The filename;
   - The ACLver initialized on one;
   - The URL to download the file ciphered with FK;
   - The file version initialized on one ciphered with FK;
   - The hash of the file with the version ciphered with FK;
   - RK ciphered with the user’s PU;
   - FK ciphered with RK;
   - A list with the permissions of the users the file is shared with.
4. The client application updates the LFiles, adding the file to the list and ciphers it with K.

5. The client sends a transaction to add the ACL. The peers verify if the transaction is correct and, in the chaincode, verify if the ACL is for a new file. If it is, the peers save the ACL.

6. Finally, the client application sends another transaction to update the user’s LFiles. The peers verify if the transaction is correct and update the $[\text{LFiles}]_K$.

After the execution of this functionality, the user is able to manage the content and the access permissions of the file. The fabric network saves a new ACL to the new file in the worldstate and saves a new version of the LFiles in the ledger. The cloud provider saved the new file.

The synchronization of files is used by the client application to verify if there is any new shared file with the user or if he lost access to it. To perform this functionality:

1. The user sends a transaction to the fabric network with the number of the most recent block verified by the user when he ran this functionality the last time. The peers, in the
Fabric Network, search for the most recent block, obtain the hash of the previous block, and run the blockchain until the current block id corresponds to the number of the block received from the client. In each block, the peers search for operations to share a file or revoke the share, in these operations, the list of permissions of the ACL is changed.

2. For every file that suffered a modification in its permissions list, the client requests the ACL to the Fabric network. The client verifies if the modifications in the list of permissions affect the user, verifying if the file was already in the user’s LFiles. The client application updates LFiles adding the filenames of the files it received new permissions, updating the existing ones that were changed, and removing the ones that it lost permission to access.

3. Finally, the client sends a transaction to the Fabric network to update $[\text{LFiles}]_K$.

After the execution of this functionality, the user has the most recent information about the files he manages. The user may discover he gained new access to a file or lost it. The fabric network saves a new version of the LFiles in the ledger.

### 5.3.5 Download File

This functionality is used to download the current version of a file from the cloud provider.

To download the current version of a file:
1. The user inserts the name of the file to download in the client application. The client application sends a query to the Fabric network to obtain the ACL of the file. The client application receives the ACL from the fabric network.

2. The client application starts by deciphering RK with his PU, then deciphers FK with RK and then the file’s version from the ACL with FK. He compares the local version with the retrieved one. If the local version is inferior to the current one, the client deciphers the URL from the ACL and downloads the file from the cloud provider. The client application calculates the hash of the downloaded file and its version and compares it with the one stored in the ACL. If the hashes are equal, then the client application updates the local file. If not, the client application warns the user some error occurred.

Another option for the download of files could be the continuous update of the files so the user would have the most recent version of its files all the time. This solution could be achieved by two methods:

- Constantly verifying the version of the files managed by the user in the fabric network: this method would decrease the performance of the fabric network because it requires more transactions, and the client application, because it would have, at least, one more process to be constantly verifying the local files and communicate with the Fabric network;

- Every time a user updates a file A, he would also update a shared file B that corresponds to the updated file A: this method would add one more file to manage by the fabric network. The verification of the permissions of this file would decrease the performance of the functionality Update file and increase the complexity of the system.

Since we do not see an advantage that would justify the disadvantages of these last two methods, we applied the method where the user downloads the files when needed.
After this functionality, the user has, in his machine, a version of the downloaded file and is able to read and edit it locally.

### 5.3.6 Update File

This functionality is used to update an existing file $F$ in the cloud.

![Diagram of Update File process](image)

**Figure 5.8: Update File**

To update a file:

1. The user inserts the name of the file $F$ that will be updated in the client application. The client application sends a request to the Fabric network to obtain the ACL of $F$.

2. The client application deciphers the $F$’s version from the ACL and verifies if they are both equal. If they are equal, the user generates a new $FK$, ciphers $F$ with $FK$, and continues to step 3. If not, the client warns the user that he has the most recent version.

3. The client application sends the encrypted file with the new $FK$ to the cloud. The cloud returns a confirmation that the update was successful.

4. The client application sends a request to the Fabric network to update the ACL with the new version information. The Fabric network confirms the update was successful.

51
3. The client application sends the file to the cloud provider. After storing the file in the cloud, the client application updates the ACL, changing the:

- ACLver;
- URL ciphered with FK;
- File version ciphered with FK;
- FK ciphered with RK;
- Hash of the file and its version ciphered with FK.

4. The client application sends a transaction to the fabric network to update the ACL of F. The peers verify if the transaction is correct, the new ACL is more recent than the stored one, the user logged in the client application has Write or Share permission over F and if the list of permissions and list of RKs are the same as the ones in the stored ACL.

There are two alternative methods to verify if the update of the file is legitimate:

- The clients verify the transactions. This option would remove the necessity of using the chaincode and increase the efficiency of updating a file. The disadvantages are that for an operation to be considered valid, at least, a certain number of clients would need to validate the transactions and that would require that some clients to be available when an update file is done for the method to be effective. The other problem is that we assume the clients do not trust each other, so the validation needs to be done by a trusted entity.

- Each client verifies the ledger. When a user needs to access a file, he asks the Fabric network to analyze the ledger verifying if all the transactions are correct. This option would increase the efficiency of the operation because there would not be the need to run the chaincode. The disadvantage would be that if a user wanted to download the file, the validation of every transaction would be costly in terms of efficiency and it would no be scalable with the increase of the number of transactions per interval of time.

After this functionality, the cloud provider receives an edited version of an existing file. The user updates his LFiles and the ACL of the file in the fabric network.

5.3.7 Update list of permissions of the ACL

The functionalities of Share File and Revoke Share are operations that update the list of permissions of a file. This list shows which users have access to a specific file and with which type of permission (Read, Write, or Share). This list is stored in the ACL of the file. Share File and
Revoke Share are used when a user A wants to give some type of access, to a file F he owns or has share permission, to a user B. It works as follows:

1. The client application of the user A sends a request to the fabric network asking for the PU of B and the ACL of F. The fabric network returns the information. The client application of user A deciphers RK with his PU, ciphers with the PU of user B, and adds it to the ACL of F. The client application of user A adds the user B and his permission to the list of permissions in the F’s ACL;

2. The client application of user A updates the ACLver. It sends a transaction to the Fabric network to update the ACL. The peers verify if the transaction is correct, if the new ACL is more recent than the stored one, if the user A has Share permission over F and if the filename, URL, file version, FK, and hash of the file with its version are equal to the values stored in the old ACL.

The process for a user A to revoke a share to the user B is similar.

1. The client application of user A sends a transaction to the Fabric network to get the ACL of F and the PUs of every user that has access to F. The client application of user A removes \([RK]_{PU_{of B}}\) from the ACL and the username of user B from the list of permissions to F. Client application A generates a new RK and FK, ciphers FK with RK, ciphers RK with the PU of each user that has access to F and updates de ACLver.

2. The client application of user A sends a transaction to the Fabric network to update the ACL of F. The peers verify if the transaction is correct, if the new ACL is more recent than the stored one, if the user A has Share permission over F, and if the filename, URL, file version, and hash of the file with its version are equal to the values stored in the old ACL.
3. The client application of A makes a change in the file F and runs the update file functionality to cipher with the new FK.

![Diagram](image)

Figure 5.10: User A removes the access of the user B to the file F

The change made to F, in the step three of the revoke share, is needed so the user B does not discover the new FK, since he knows the last value of F, and allow the other users to have access to the latest version of the file since the FK that is on the ACL does not correspond the FK used to cipher F before the revoke share operation. The rationale to update the list of permissions is the same as the used to update a file.

After these functionalities, a user B may have gained or lost permission to a file. The fabric network saves a new version of the ACL of the file F.

5.3.8 Objectives and Solutions

In this section, we will explain how the objectives of Hypercloud are achieved with the proposed solution.

- **Guarantee the confidentiality of the managed files from outsiders**: Through the use of traditional cryptography, the users can cipher the files with FK. FK is only accessible to the users with permission to access the file since it is ciphered with RK that is ciphered with the users’ PU.

- **Allow the verification of the integrity of the files**: The fabric network can guarantee that the managed data did not tamper with. This property allows us to save the hash of
the ciphered file in the respective ACL. When a user downloads a file, he can compare the hash of the downloaded file with the one saved in the file’s ACL, verifying the integrity.

- **Support file sharing between users of the system:** With the use of the property of the immutable ledger of the blockchain, the user’s with share permission can modify the ACL and give permission to another user B to the file. User B, to acknowledge the share, verifies the past transactions saved in the ledger to check if he received any new permission to a file and updates his LFiles, being able to download the file. This share solution does not need to trust in any other service, e.g. e-mail. It is only dependent on the Hypercloud itself.

- **Remove dependability of intermediate entities:** this is achieved with the use of the Hyperledger Fabric technology. Hyperledger is composed of peers that run a chaincode to guarantee that the transactions sent by the users are valid, maintaining a correct state of the system.

### 5.3.9 Summary

We built Hypercloud with various technologies and libraries, like Java and the fabric’s API for Java. Then we presented the used data structures and how they were built like the ACL that is a JSON file with various information about a file. Finally, we presented the implementation of the functionalities in the Hypercloud and this solution achieves the objectives.
Chapter 6

Evaluation

In this chapter, we present the evaluation of Hypercloud. In Section 6.1 we present the performance evaluation of Hypercloud. In Section 6.2, we present a performance comparison with the Crypto Cloud. In Section 6.3, we present a security evaluation of Hypercloud.

6.1 Performance Evaluation

To evaluate the performance of Hypercloud, we performed several tests to obtain the time to perform the functionalities and operations. Those times were measured by calculating the difference between the start and end time. The timestamps were obtained with a Java’s native library. This method allows for a detailed measurement of the various operations performed by a functionality with low overhead associated. The experiments were performed with an Intel(R) Core(TM) i5-4210M CPU running at 2.60GHz, with 8,00GB of DDR3L RAM running at 1600MHz, and 1Tb of HDD running at 5400 rpm. The OS used was Ubuntu 18.04 (x64). For the experiments, all the entities, except for the cloud provider, were running in the local machine so the environment was not the most realistic, but it is good enough to test the performance of the interactions of the various entities of the system and how does it compare with the previous version of Crypto Cloud. Every instance of the fabric network components run in a Docker container. The tests consist of the client application running a functionality several times. These functionalities include read, write, share, and revoke operations. The tests are the same that were performed in the first version of the Crypto Cloud prototype, which will be compared with Hypercloud. The performed tests measured the latency of each functionality or operation ran by the functionality in the described environment. We analyze the average and standard deviation. The experiments took place in September 2020.
6.2 Hypercloud performance

In this section we will present the results of the tests performed to the Hypercloud. We chose files with 100KB, 1MB, and 10MB to perform the tests because we wanted to analyze what is the system’s behavior for a small, medium, and big file.

6.2.1 Performance of the Functionalities

In this section we present the results that show the time to run the functionality for different occasions.

Upload New File

This test consists of the upload of files with 100KB, 1MB, and 10MB, 50 times each and the results are in Figure 6.1. We removed the first 5 and the last 5, to reduce the impact of Java’s startup time. From this test, we obtain the time to run the complete functionality.

![Write Operation - Upload New File](image)

Figure 6.1: Time to upload a new file with different sizes

In Figure 6.1 we show the time to upload a file of 100KB in blue, the time to upload a 1MB file in orange, and the time to upload a 10MB file in grey. The time to upload a file increase as the size of the file increases. This is expected because the time spent ciphering the file, calculating the hash, and uploading it to the cloud depends on the size of the file. We also note that the files are increasing by ten times from the previous size, but the time to perform the operations increases at a much slower rate. It makes sense because not all of the operations of the functionality do depend on the file’s size, so an increase in the file’s size does not mean an increase in all operations performed during the functionality.
Download File

This test consists of the download of files with 100KB, 1MB, and 10MB, 50 times each. The results are in Figure 6.2. We removed the first 5 and the last 5, to reduce the impact of Java’s startup time. From this test, we obtain the time to run the complete functionality.

![Read Operation - Download File](image)

Figure 6.2: Time to download a new file for files with different sizes

With the data presented in Figure 6.2, we can verify that the time increases as the size of the files increases. This is the expected behavior since the download of the file and its validation depend on the size of the file. We also observed that the files are increasing by ten times from the previous size, but the time to perform the operations increases at a much slower rate.

Share File

The results in Figure 6.3 consist of the measurement of the time to give each of the three types of permission to a user. Each test was run 50 times for each permission. For each 50 runs, we removed the first 5 and the last 5, to reduce the impact of Java’s startup time.

As we can verify, the time to give each permission is the same. This is an expected result because the operations are the same independently of the permission to give. This operation has no variables, like the file size or number of users with access to the file, that would change the time to perform the operation.

Revoke Share

The first test measures the time to revoke a file while a different number of users have access to it. We tested this with a 100KB file and with four, three, and two users having access to the file. For this first test, we did not count the time to update the file, since the time to perform
these operations depends on the size of the file, and that is not a variable in this test. For each 50 runs, we removed the first 5 and the last 5, to reduce the impact of Java’s startup time.

As we can observe in Figure 6.4, the time to revoke a file increases with the number of users with access to the file. This is the expected behavior because the user that ran the revoke operation has to generate a new RK, ask for the PU of all the users that will retain access to the file, and cipher the RK with PU of every user. With the set of results we have, we observe that the time increases roughly 15 seconds with the addition of each user.

The second test measures the time to revoke a file with different sizes and with 2 users having permission over. The size of the files that were tested are 100KB, 1MB, and 10MB. The results are in Figure 6.5.
Figure 6.5: Revoke Share for files with 100KB, 1MB and 10MB

The results in Figure 6.5 show that the time to revoke a file increases as the size of the files increase. This is expected since the revoke operation does an update of the file that was revoked to save the result of the encryption with the new FK that is saved in the ACL. This operation depends on the size of the file, because, as the size increases, more time is needed to cipher the file and upload it at the Dropbox cloud account, as shown in Figure 6.6.

<table>
<thead>
<tr>
<th>Size</th>
<th>Operation</th>
<th>Download</th>
<th>Upload</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small</td>
<td></td>
<td>1,8</td>
<td>1,2</td>
</tr>
<tr>
<td>Medium</td>
<td></td>
<td>1,9</td>
<td>5,0</td>
</tr>
<tr>
<td>Big</td>
<td></td>
<td>3,8</td>
<td>38,1</td>
</tr>
</tbody>
</table>

Figure 6.6: Cloud operation for files with 100KB, 1MB and 10MB

6.2.2 Time per operation

In this section we show how is the time distributed through the functionalities.

Add Cloud Account

This test consists of running the Add Cloud Account functionality 100 times to add 50 Dropbox cloud accounts to the same user. The first 50 runs measured the time to obtain the access token and to update LToken. The last 50 runs measured the total time to execute the functionality Add Cloud Account. For every 50 runs, we removed the first 5 and the last 5, to reduce the impact of Java’s startup time.
In figure 6.7, the first column corresponds to the time for the Add Cloud Account functionality to execute. The portion in orange corresponds to the time to get the access token after receiving an access code from the cloud. This measurement depends on the bandwidth and the cloud provider and these are things we cannot control. The portion in yellow corresponds to the time to update LFiles locally and in the blockchain. In this time interval, the client performs a write in the LToken, ciphers it with the user’s AES key, and sends it to the fabric network. The fabric network runs the chaincode to update the LToken and updates the world state. The portion in green shows the time of the other operations of the functionality. Those are the verification if the user is logged in, the start of the local server to receive the access token, open the web browser for the user to authenticate himself, and give authorization for the app to interact with the cloud account, and stop the local server. The sum of these three portions corresponds to the time to run the complete functionality for a file with 100KB, which is about 20.2 seconds with an error of 238 ms.

We can verify that the biggest part of the time is spent while updating the LToken. This operation of updating the LToken interacts with the fabric network and this interaction is slow since the client waits for a satisfactory response from the fabric network to proceed.

This distribution of the time in the functionality was expected since we expect the communication with the blockchain to be the slowest part.

**Upload New File**

This test consists of uploading a new file with 100KB 50 times, which the results are in the second column of Figure 6.7. The objective was to measure:
- The time spent to calculate the information to generate an ACL and to save it in the ledger in orange;

- The time to apply the cipher a file, update it in a Dropbox account and generate a shareable link in yellow;

- The time to write the shareable link, filename, and version in the LFiles, cipher with it and update the blockchain in green;

- The time for the other operations, that is the verification if the user is logged in and the print of the result of the functionality in brown.

We removed the first 5 results and the last 5, to reduce the impact of Java’s startup time. The sum of every portion is the time to execute the Upload New File operation for a file of 100KB, which is 34.6 seconds with an error of 0.5 seconds.

As we can verify in the second column of Figure 6.7, in this functionality, the biggest part of the time is spent in the interaction with the blockchain to upload a new ACL and to update the value of LFiles. Since the accesses to the blockchain are costly when comparing with the other performed operations, and this functionality makes two transactions that invoke a chaincode to update the world state, those values were expected.

Download File

We performed a second test to this functionality, which the results are shown in the third column of figure 6.7. We downloaded a 100KB file 50 times to measure the time the functionality takes to:

- Obtain the ACL of the files from the fabric network and decipher the metadata from the list in orange; download the file from the Dropbox account in yellow;

- Validate the downloaded file, calculating the hash and comparing with the saved one and decipher the file in green;

- Update the local version, and update the LFiles in brown.

We removed the first 5 results and the last 5, to reduce the impact of Java’s startup time. Analyzing the third column of figure 6.7, we can verify that most of the time was spent communicating with the fabric network while obtaining the ACL and updating the LFiles (we already know that an LFiles update takes around 15.8 seconds). With this test, we did a write operation to the blockchain, when updating the LFiles, and a read operation, when asking for
the file’s ACL. We can verify that both times are similar so we may conclude that every access
to the fabric network to perform a read or write operation over an ACL has around the same
cost.

**Update File**

This functionality was evaluated considering two tests. The first test measures the time to
update a file in Dropbox with Hypercloud. The second test measured the time the peers take
to approve the transaction. We ran each test 50 times to update a file of 100KB. We removed
the first 5 and last 5 values each set of results.

The fourth column of Figure 6.7 corresponds to the test performed on this functionality. In
orange, we have the time the peers take to run the chaincode, verify if the user updating the file
has permissions to it and if the values of the ACL he sent are valid, and return a response to the
client application. In green, we have the time the client takes to perform the other operations
of the functionality. These operations are the update of the LFiles, get the ACL from the
blockchain, the upload of the file to the cloud account, and the calculation of the new values of
the ACL. The sum of both portions gives us the time to execute the full functionality, which is
49 seconds with an error of 0.2 seconds.

With the results of previous tests, we know that the time to update the LFiles is around
15.8 seconds, and the time to get the ACL is around 15.7 seconds. With this test, we know
that the time to update the ACL is around 15.8 seconds. We can conclude that approximately
47.3 seconds of the total of 49 seconds the functionality took to finish are spent with operations
that interact with the blockchain, as expected. This test also helps to justify the conclusion
that write and read an ACL in the blockchain takes the same time in this prototype. This can
be explained by the fact that ro read and to write, the peers need to search for a value in the
worldstate, being it to return to the clients or to verify if the value is not duplicated. This time
to search for a value in the worldstate takes most of the time of a read or write operation, so
the times become similar.

**Share File**

The fifth column of Figure 6.7 shows the results of the second test performed to the Share File
functionality. It measures the time that this functionality takes to verify if the transaction sent
by the client application is valid and the time spent doing the other operations. The orange
portion is the time that the fabric network takes to verify if the transaction is valid which
includes, verify if the ACL sent by the client application is valid comparing with the older one,
and verify if the user sending the transaction has permission to share the file. The yellow portion is the time to obtain the RSA public key of the user that will receive permission to the file from the blockchain. The green portion shows the time to obtain the file’s ACL and update the values of the ACL locally. We know that downloading the ACL takes around 15.8 seconds. With this test, we know that verifying the validity of the ACL for this functionality takes 15.8 seconds and the download of the public key takes 15.4 seconds. As we expected, the communications with the fabric network take the most time. With this test, we can verify that time to get a public key of a user is smaller compared with the time to read an ACL. This may be because there were more ACLs than users saved in the world state, so the search for the public key is easier. With a bigger number of users, the time to search for a public key will increase.

### 6.3 Performance comparison

It is not possible to test both systems in the same environment since Hypercloud is running in Ubuntu 18.04 and Crypto Cloud is running in Windows 10. The objective of this evaluation is to verify how does Hypercloud performs and compare how much the increase of security measures impact the system comparing with the values obtained for Crypto Cloud.

![Hypercloud vs Crypto Cloud (100KB)](image)

Figure 6.8: Performance comparison for a 100KB file

The values shown in the graphics of figures 6.8, 6.9 and 6. correspond to the time to perform a Read, Write Share and Revoke Share operation in the first version of the Crypto Cloud and the Hypercloud. They do not include the time to interact with the Dropbox. We can conclude from the results shown in the graphics that the Hypercloud solution is slower than the first version of Crypto Cloud in every operation.

That is an expected result because Hypercloud main objective is to improve the security of
the current cloud storage aggregator systems like Crypto Cloud. To achieve that, Hypercloud explores the blockchain technology to replace the CCDS and some of the key management responsibilities. While Hypercloud uses the blockchain to manage the data without the need to trust any central entity, Crypto Cloud uses a faster central server for the same purpose, but if we apply the same assumptions that are used by Hypercloud, Crypto Cloud is not a reliable system, since it needs the correct behavior of the central server to function.

The values we obtained for the Hypercloud system were higher than expected compared to Crypto Cloud and this may be because every entity was running in the same machine. This is not the most realistic environment and the number of processes running at the same time may decrease the performance of the machine and, consequently, the time to run the functionalities.
increases. Other factors of imprecision are:

- The variable performance of the cloud provider causes variations in the time for the same operation;
- The task management performed by the operating system will cause variable times of execution;
- The imprecision of $\pm 1$ ms of the timers;
- The internet speed is variable, although we used a wired connection.

Although Hypercloud is slower than Crypto Cloud it is much safer. The use of the intermediary entities instead of Hyperledger Fabric helps to increase the performance but adds vulnerabilities that are not covered by our assumptions. The performance of Hypercloud can still be improved as we will explain in the Future Work section.

### 6.4 Security Analysis

In this section we analyse the security aspect of the Hypercloud solution.

The first approach is to identify the possible actors of threats to the system. They can be:

- Users outside of the system
- Hypercloud’s users
- Peer nodes
- Cloud providers

For each of these actors, we applied the STRIDE model [46] over the proposed solution, which is a model used to help to find threats on the system. For each of these actors we verified how could they cause:

- Spoofing – successfully steal an identity
- Tampering – illegal modification of data
- Repudiation – not being able to associate the action with a unique individual
- Information Disclosure – leakage of information
- Denial of Service – make the service unavailable
- Elevation of privilege.
6.4.1 Spoofing

In Hypercloud, a user is identified in the following manners: to the cloud by his credentials or by his access token; to fabric network by the signature of the transactions done with his private ECDSA key. The users identify the cloud provider and the fabric network through the successful establishment of a secure channel. To impersonate a cloud provider or the fabric network the actor would need the private keys of these entities to open a secure channel successfully with a client application of a user. Since the establishment is done by the APIs of the entities, we consider them to be safe, so it is not possible to steal the identity of these entities. An actor could try to impersonate another user and for this, he would need the private ECDSA key of the user. This key is generated by the fabric network, that is considered to be a secure operation, sent through a secure channel to the user and saved in the user’s machine that, following the assumptions we used for Hypercloud, is safe. So, no outside user, Hypercloud user, or peer node could impersonate another user.

6.4.2 Tampering

An actor could try to modify the data stored in the fabric’s network. One of the Hyperledger Fabric’s properties is that the ledger is immutable, so the only way possible to tamper data is to do a valid transaction. It is possible to modify the old entries of the ledger, but that would be detected when running the chaincode since the results would not be the same across the peers. An outside actor could not interact successfully with Hyperledger Fabric because it is a private blockchain, so the user would need to register in the system. An Hypercloud user also will not be able to illegally modify data in the blockchain, because for every transaction the peers run a chaincode where only an authorized user can modify the data. In the case of the LFiles and LToken, every user can update those files with no verification by the peers, but every transaction is signed by the user himself, identifying him. When the owner of the LFiles and LToken tried to access he would ignore the invalid update since he can verify the transaction signature, and continue to search for his last transaction in the immutable ledger. The peers have access to the ledger and can modify any data maliciously, but Hyperledger Fabric uses consensus protocols that support a certain number of peers to malfunction and still guaranteeing that the clients get valid data. One of the assumptions is that the Hyperledger Fabric will have enough well-behaved peers to guarantee the good functioning of the network. The information in the user’s machine is safe, so no entity is able to tamper it. The information in the cloud provider is possible to be tampered by the cloud provider. The files are stored in the cloud provider and if he is not well behaved or if it is plotting with a peer and/or malicious client it is possible to change the
content of the files. Hypercloud does not solve this problem but offers a way of detecting it. The ACL has the hash of the ciphered file with its version. When a client downloads a file from the cloud, he calculates the hash of the file with the version and compares both.

6.4.3 Repudiation

Every action in the blockchain causes the creation of a signed transaction by the author that is stored in the ledger. The actions made to the cloud using this system also communicate to the fabric network, obligating the client to generate signed transactions. These transactions can be traced and therefore it is possible to know the performed action, to which file and by whom.

6.4.4 Information Disclosure

In Hypercloud, every sensitive information that leaves the user’s machine is encrypted with the user’s keys, RK, or FK. The files stored in the cloud are encrypted with the FK that is encrypted with RK. RK is encrypted with the PUs of the users with access to the file, so it is only possible to access the content of the file if the user has a PK to decipher the RK. Since the RKs are in the user’s machine, we assume they are safe. If the access to a file is removed from a user, the FK and RK of the file change, and the file is encrypted with the new FK, modified and updated in the cloud, so the user has no access to the new versions of the file. The peers, the outside users, and the cloud providers can access the shareable link by brute force since it is ciphered with the FK in the ACL of the file, but they would only see ciphered data that they could not decipher. The information in the ACL is also ciphered with the FK. The LToken and LFiles are ciphered with the AES key that is generated by the user and stored in the user’s device. We conclude that there are no problems with information disclosure in Hypercloud.

6.4.5 Denial of Service

We designed this solution assuming that there were no attacks of denial of service because the objective of this solution is not to solve such vulnerabilities. If we assume Denial of Service attacks exist then they are a threat to Hypercloud. This prototype depends on the cloud providers to stay available. The fabric network is susceptible to have its performance slowed down by, for example, a big number of transactions at the same time.

6.4.6 Elevation of privilege

A way to increase the privilege of a user is to gain access to a file or increase its access (e.g., from R access to W, W to S). An outside user could not elevate his privilege over a file because
he needs to be a user of the system. A malicious Hypercloud user or a peer is the remaining possible actors. To ensure the permissions are only assigned by users with the correct access to a file, the fabric network verifies in the chaincode if the user sending the transaction has Share permission over the file, and only if it is true, does the operation proceeds. Since the transactions are signed and the peers verify the signatures, a malicious user could not elevate his privilege. If a malicious user plots with a malicious peer, the malicious peer could consider the illegal transaction of the malicious user valid and give to the user Share permission over a file, but since the Hyperledger Fabric applies consensus and it is not possible, by the assumptions we presented, to have a big enough number of malicious peers for the fabric network to not work properly, this method would not elevate the user’s privilege or the peer’s. We conclude that there is no elevation of privilege threat.

6.5 Summary

Hypercloud was built with the objective of improving the security of systems like the Crypto Cloud. This addition of security increased the time to perform the same type of operations like download or share a file. In the security analysis, we can verify that Hypercloud is a robust solution and that supports malicious components, like byzantine peers, contrary to the Crypto Cloud that cannot support a faulty server.
Chapter 7

Conclusions

Cloud storage is a highly available and portable solution to manage data. The users register accounts and are able to save their data in the provider’s infrastructure. The users may end up with multiple cloud accounts of different cloud providers, making the file management harder. The cloud storage aggregators offer a global view of the user’s files but all present some security weaknesses like the assumption of a cloud provider that does not alter data or the necessity to trust in multiple entities. Hypercloud is a system that solves these security weaknesses.

Hypercloud is a system that focuses on improving the security aspect of Crypto Cloud. To achieve this, Hypercloud modifies the existing architecture, removing the central server and uses the Hyperledger Fabric’s technology, implemented in the entity called fabric’s network. This technology supports byzantine failures through the use of consensus protocols, so it can be trusted while the assumption that a fraction of the peers are working correctly is true. The fabric network is responsible to manage the user’s public keys, the file’s metadata, and guarantee that the operations performed over the data are valid. Hypercloud uses cryptographic algorithms to guarantee confidentiality and verify the integrity of the managed files.

Hypercloud successfully removes the need to trust in a central server, improving the security aspects of previous solutions like Crypto Cloud and Storekeeper.

7.1 Future Work

Hypercloud’s main objective is to improve the security aspects of cloud storage aggregators. This objective was achieved and the prototype was properly implemented. Still, there are possible improvements to the system:

- Use of a remote KMS. To develop Hypercloud we removed any intermediate infrastructure like the KMS or central server from its architecture since we wanted to assume every entity
is not trustworthy. The KMIP is a protocol widely accepted and mature and the HMS adds an even higher degree of trust to the KMS entity. To improve the portability of the solution, future solutions could add a remote KMS to the Hypercloud solution;

- Add the central server as a cache. This dissertation was written to prove that it is possible to build a cloud storage aggregator solution without the need of a central server or any other third entity. However, with the correct verifications, it is possible to use the central server as a way to increase the performance of the system. The central server can be used to save the ACL as they are in the ledger. When the user wishes to download a file, first, he would receive the ACL from the server, get the URL, download the file and compare the hash of the file in the ACL with the hash of the downloaded file. If they do not match, the user would repeat the whole process but with the ACL from the blockchain. The server would work as a cache that the user would choose if he wants to trusts or not. However, this solution is vulnerable to, for example, a server changing the URL of the file. This addition would add new challenges that need to be addressed.

- Addition of more cloud providers. Adding support for more cloud storage would make the solution more useful since it would support more cloud accounts. This can be implemented by making the user choose from which cloud provider he wants to add an account. The information of the cloud provider can be stored in the LToken with the name and token. The client service would be able to know which API he would need to use to interact with the account.

- A Graphical User Interface for the client application. In Hypercloud, the user interacts with the client application through a console. The implementation of a GUI for the client application would make the interaction with the system more user friendly.
Bibliography


