Providing trust to multi-cloud storage platforms through the blockchain

Claudia Fiore
Instituto Superior Técnico, Universidade de Lisboa

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

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

Keywords: Cloud Storage Services, Security, Integrity, Blockchain

I. Introduction

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

In this work, we focus on Crypto Cloud, a privacy-preserving cloud aggregation service. It allows for using multiple cloud providers without renouncing privacy, guaranteeing the confidentiality and integrity of managed files [1]. Crypto Cloud enables file sharing on multi-user multi-cloud storage platforms. The Crypto Cloud application relies on a central server which manages all the information (metadata) related to users, files, permission and cloud storage. In Crypto Cloud, there is the assumption that the server does not modify the metadata actively. He could access them, but it acts honestly. If malicious users attack the server, the integrity of metadata could be compromised.

A. Goals

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

B. Structure of the document

This document is organized as follows: Section 2 presents an overview of the related work on secure cloud storage solutions; Section 3 describes the blockchains background; Section 4 described the blockchain State of the Art; Section 5 details the aspects of the proposed solution; Section 6 describes the implementation of the solution; Section 7 presents the evaluation of the proposed solution and the analysis of the obtained results; and Section 8 concludes this document.

II. Related Work

In this section, we detail the Crypto Cloud system, a privacy-preserving cloud aggregation service that enables file sharing on multi-user multi-cloud storage platforms.

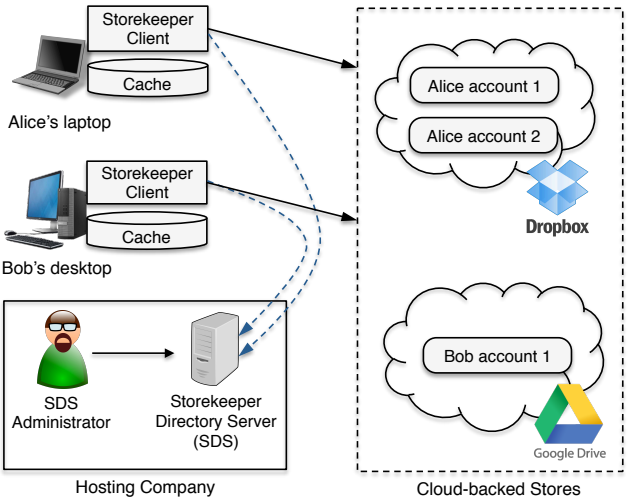

Fig. 1: Storekeeper Architecture [2]

The Crypto Cloud application is based on Storekeeper's approach and extends it by introducing new management of users' identities, authentication and integrity protection over files.

A. *Storekeeper*

Storekeeper [2] supports multiple cloud providers and is supported by a directory server to store metadata (see Figure 1). It is assumed that this server does not launch active attacks, thus metadata protection was not part of the project's scope. The system guarantees data confidentiality using symmetric keys, that are distributed to users using public-key pairs. The approach also provides sharing and access control mechanisms. However, this approach does not consider data integrity and employs a non-efficient key management mechanism that performs extensive number of requests to the Storekeeper Directory Server (SDS), resulting in a decrease of the application's performance.

B. *Crypto Cloud*

Crypto Cloud is a distributed system that provides a secure cloud aggregation service for multi-user multi-cloud storage platforms [1]. The Crypto Cloud system architecture is shown in Figure 2. The Client Application is the central component of the system. This component manages the user's files, and it interacts with all the other components of the system. The Crypto Cloud Directory Server (CCDS) serves the client application and, as an assumption, it does not act maliciously. It can listen to the exchanged messages but follows the system's protocol and does not launch active attacks. The CCDS manages the metadata associated with users, files, shares, and clouds. It implements an Access Control Model based on authentication to provide access control on files. It provides also version control and integrity of files.

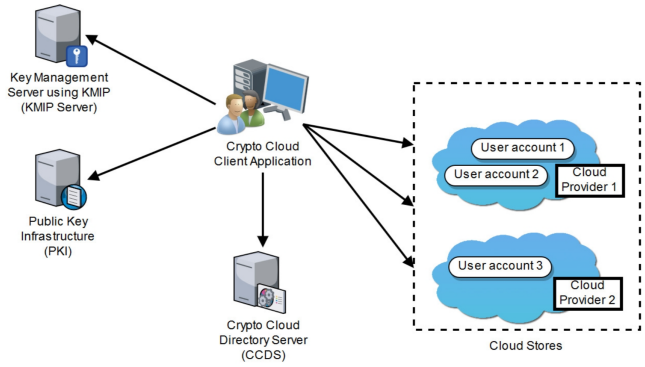

Fig. 2: Crypto Cloud Architecture [1]

Moreover, the CCDS manages the users' identities through a Key Management Server, that follows the KMIP Protocol, and a Public Key Infrastructure. They allow the remote access and management of user's cryptographic keys and certification of the users' identities, guaranteeing proper authentication while protecting and sharing sensitive files.

III. Background

Nowadays, it is possible to create digitally distributed ledgers that can be shared across a network of multiple sites, geographies or institution. This technology is known as Distributed Ledger Technology (DLT). All participants within the network can have their identical copy of the ledger. This technology is used to process, validate and authenticate transactions. The records are only ever stored in the ledger when a consensus has been reached by the parties involved. DLTs drastically reduce the cost of trust, in fact, the main feature of this technology is that the ledger is not maintained by any central authority. A blockchain is a form of DLT. The data on the blockchain are grouped and organized in blocks, which are linked one to the other and secured using cryptography (see Figure 3). Every block holds a complete list of transaction records and it is linked with the previous block maintaining its hash.

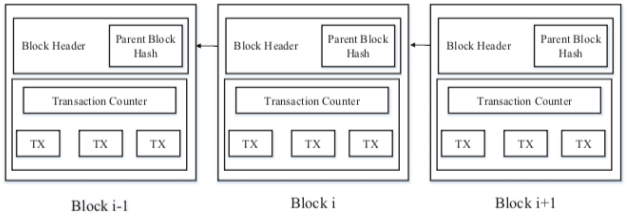

Fig. 3: The blockchain [3]

The blockchain has the following main characteristics [4]:

- Immutability. The hash guarantees that the blocks are linked to each other, but it also guarantees the integrity of the blockchain. If one block is altered, the hash on the block that follows it stops matching the blocks;

- Transparency. Anyone can view the records and verify the authenticity of the data;
- Decentralization. Rather than relying on a central authority to securely transact with other users, blockchain utilizes innovative consensus protocols across a network of nodes to validate transactions and record data in a manner that is incorruptible.

A. Classification of blockchain systems

There are various categorizations of blockchain types, according to whether authorization is required for network nodes which act as verifiers, and whether access to the blockchain data itself is public or private. The first distinction is between permissionless and permissioned blockchains. Permissionless blockchains are systems in which anyone can participate in the verification process. No prior authorization is required, and a user can contribute his/her computational power, usually in return for a monetary reward. While permissioned blockchains are systems in which the miner nodes are preselected by a central authority or consortium. We can also distinguish between public blockchains, in which anyone can read and submit transactions to the blockchain, and private blockchains in which the permission of read/write is restricted to users within an organization or group of organizations. It can be noticed that most permissionless blockchains feature public access, while most permissioned blockchains intend to restrict data access to the company or consortium of companies that operate the blockchain.

B. Consensus

To ensure that only legitimate transactions are recorded into a blockchain, a new block of data will be appended to the end of the blockchain only after the nodes on the network reach a consensus. The main consensus protocols are described below.

Proof of Work (PoW) is the consensus used in the Bitcoin and Ethereum network [5]. In short, PoW works by making the nodes spend computational power until they can find out a hash that satisfies a certain rule, for a certain block. When a node finds this hash, it is allowed to extend the blockchain with that block. The node transmits the new blockchain to all the other nodes. It is assumed that the longed valid chain held by any block is the correct one. Additionally, the creator of a block includes a reward in the block.

Proof of stake (PoS) is an energy-saving alternative to PoW. Miners in PoS have to prove the ownership of the amount of currency, the "stake". It is believed that people with more currencies would be less likely to attack the network [6]. Therefore, the Proof of Stake attributes mining power to the proportion of coins held by a miner. In the PoS system there is no block reward, so, the miners take only the transaction fees.

Practical Byzantine Fault Tolerance (PBFT) is a replication algorithm to tolerate Byzantine faults [7] in which is assumed that some of the involved parties might be corrupt or otherwise unreliable. In private networks, where the participants are whitelisted, costly consensus mechanisms such as Proof of Work are not needed, practically removing the need for an economic incentive for mining. PBFT requires that every node is known to the network. In the consensus, preselected nodes select and order the transactions.

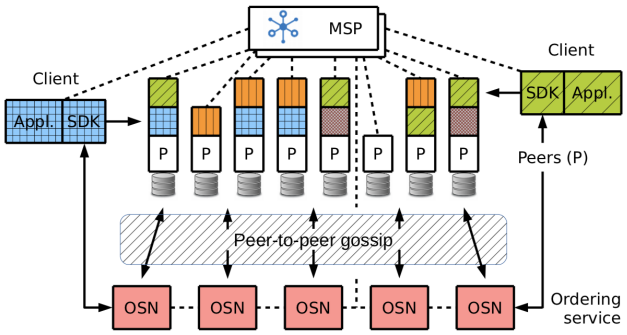

Fig. 4: A complete Fabric network [10]

IV. Blockchain State of the Art

In this section, we analyze three different blockchain systems: Hyperledger Fabric, Ethereum, and Filecoin. We choose Hyperledger Fabric that implements a private blockchain, and Ethereum and Filecoin which implement a public blockchain.

A. Hyperledger Fabric

The Hyperledger platform is an open source framework to build permissioned blockchains usable in business [8]. The Hyperledger Fabric network is formed by a set of nodes that are the communication entities of the blockchain. As depicted in Figure 4, in the network there are clients, which stand between the network and the end-user, Peers, that maintain a ledger and runs chaincode containers to perform read/write operations to the ledger, and Ordering Service nodes, that constructs the ledger as a hash chain of blocks of transactions guaranteeing the total order of the blocks.

Two or more network members communicate through a *channel* which allows private and confidential transactions. Every entity on the network is associated with an identity, that is an X.509 digital certificate [9], and it is used to determine permission. A verifiable identity must come from a trusted authority: the Membership Service Provider (MSP). The MSP identifies which Root CAs and Intermediate CAs are trusted to define the members of a trust domain. MSPs are mandatory at every level of administration (network, channel, Peer, Orderer). Hyperledger Fabric introduces a new architecture for transactions called execute-order-validate. Peers execute a transaction and check its correctness thereby endorsing it. Then, the Orderer Service orders transactions via a consensus protocol and validate transactions before committing them to the ledger. Hyperledger Fabric provides also the possibility of developing smart contracts, known as chaincode. Users can use chaincode to design assets, as well as the logic that manages them.

B. Ethereum

Ethereum is a blockchain platform to create decentralized applications, and it is built on a public blockchain [11]. In Ethereum network there are full nodes that are entities that verify the blocks and maintain the full copy of the blockchain. There are also light nodes that download block headers by default and verifies only a small portion of what needs to be verified. Finally, there are miners which are nodes that validate the new blocks by solving a crypto problem. Ethereum mainly uses the Proof of Work consensus, called Ethash, that involves finding a nonce input so that the result is below a certain difficulty threshold.

Every node is represented by its account. Every account is defined by a pair of keys: a private key, and a public key. One fundamental concept in Ethereum is the concept of fees. Every computation that occurs as a result of a transaction on the Ethereum network incurs a fee. For every executed operation there is a specified cost, expressed in a number of Gas units. Moreover, Ethereum supports smart contract functionality that is a collection of codes and data, that resides at a specific address on the Ethereum blockchain.

a) Ethereum Parity (Private): Parity is an Ethereum client that supports private chain and private network. Parity allows storing encrypted data on the Ethereum blockchain. It is possible to create *private contracts* which are stored encrypted inside a public contract. Moreover, Parity allows the network participants to different permission aspects of the blockchain. All permissioning is based on blockchain accounts, which means that permissions always correspond to an address.

Finally, Parity supports different consensus engines. The main one is the Ethash Proof of Work [12] but the most interesting is Proof-of-Authority consensus. It can be used for private chain setups and uses a set of "authorities" which are nodes that are explicitly allowed to create new blocks and secure the blockchain. In Proof-of-Authority, validators typically represent some real-world entities, which prevents Sybil attacks. These validators are known as validator set, that is a group of accounts allowed participating in the consensus, validating transactions and blocks.

C. Filecoin

Filecoin is a Decentralized Storage Network based on the blockchain. It is intended to be a cooperative digital storage and data retrieval method [13]. The Filecoin Decentralized Storage Network (DSN) is auditable, publicly verifiable and designed on incentive. It aggregates storage offered by multiple independent storage providers and provides data storage and data retrieval to the client. Clients spend Filecoin hiring miners to store or distribute data. Filecoin miners compete to mine blocks with sizable rewards. Filecoin mining power is proportional to active storage. In the Filecoin protocol, storage providers must convince their clients that they stored the data they were paid to store [13]. Storage providers will generate Proofs-of-Storage that allows a user to check if a storage provider is storing the data at the time of the challenge. A verifier can check if a prover is storing her/his data for a range of time.

Any user can participate as a Client, a Storage Miner, and a Retrieval Miner. Clients pay to store data and to retrieve data in the Decentralized Storage Network. Storage Miners provide data storage to the network by offering their disk space. Finally, Retrieval Miners provide data retrieval to the network by serving data to the users.

	Ethereum	**Hyperledger**
Scalability	● High node scalability	● High node scalability.
Throughput	● About 20 transactions per second	● About 3500 transaction per second
Latency	● 10 to 15 seconds	● Less then 1 second
Size	● Limited by the gas limit	● Customizable
Energy consumption	● High	● Low
Usability	● Programmers must learn Solidity to write Smart Contracts	● Easy to program smart contracts (Go, Java, Node.js)

TABLE I: Performance Pros and Cons

Security property	**Hyperledger Fabric**	**Ethereum**	**Filecoin**
Authentication	✓	✗	✗
Privacy	✓	✗	✗
Confidentiality	✗	✗	✗
Integrity	✓	✓	✓

TABLE II: Security properties

D. Conclusions

The Filecoin's network is still in the implementation phase [14], so we do not consider it in the choice of the system. From the analysis presented in Table I Hyperledger performs consistently better than Ethereum. Ethereum incurs large overhead in terms of memory and disk usage because of Proof of Work consensus protocol. From [8] Ethereum execution engine is also less efficient than that of Hyperledger, and the Hyperledger's data model is low level, and its flexibility enables customized optimization for analytical queries of the blockchain data. From Table II emerges that the Fabric platform offers security properties, such as authentication and privacy, that Ethereum cannot deliver mainly because of its permissionless nature.

Given this analysis, of both the performance and security of the systems, it is possible to conclude that Hyperledger Fabric is the most suitable platform for this work. It is permissioned, guarantees privacy, there is no economic cost on transactions and, generally, performs better than Ethereum.

V. Design

The goal of this thesis is to extend the Crypto Cloud application with metadata management that ensures security sensitive operation without relying on the central server. Using the blockchain system we can achieve this goal. The blockchain technology guarantees integrity, immutability, freshness, and authenticity of the stored information, without relying on any third party.

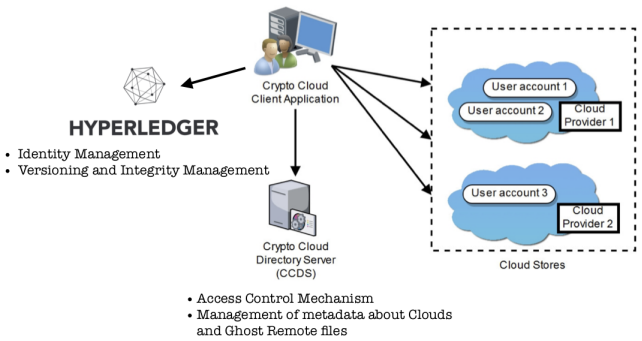

Fig. 5: New Crypto Cloud Architecture

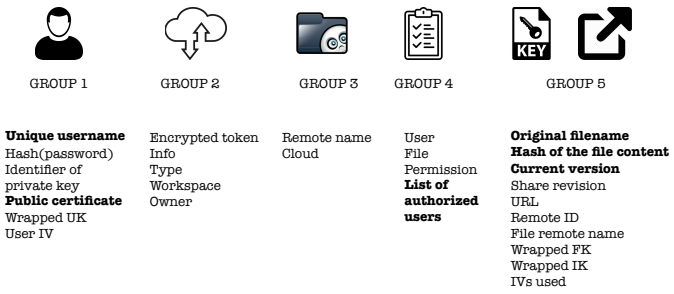

Fig. 6: CCDS Metadata

We propose an alternative version of the Crypto Cloud system, integrating it with the Hyperledger Fabric blockchain. The new Crypto Cloud's architecture is illustrated in Figure 5.

A. Crypto Cloud Client Application

The client application is the central component of the system. It is responsible for managing the users' files and interacting with all other Crypto Cloud's components. The application allows authenticated users to upload and download files, as well as to manage their access permission. In the new version, the client also communicates with the Hyperledger Fabric network. The communication is done using a Fabric API. This API is used to put and retrieve identities and files' metadata on the blockchain. The client also verifies the consistency of the CCDS metadata, comparing them with the ones on the blockchain.

B. Crypto Cloud Directory Server

The Crypto Cloud Directory Server serves the client application. Its functionalities remain the same as the previous version of Crypto Cloud. This component acts as a metadata repository, responsible for the system's metadata associated with users, files, shares, and clouds. Therefore, we can divide the metadata into five groups, based on their function (see Figure 6). We do not want to trust the server, so we should put the metadata associated with each server's functionality both on the blockchain and the CCDS. The sections below explain in detail each of the modified functionality.

C. Management of identities

A key concept in the Crypto Cloud system is the identity. The identity is the mapping between a User and his/her public key. In Crypto Cloud, the KMIP service generates the RSA key pairs: a public key and a private key. The system uses the generated keys to encrypt/decrypt the other cryptographic keys. To guarantee proper certification of the users' keys and to assure their identity when accessing the managed files, the system relies on a Public Key Infrastructure, which acts as a trusted third-party entity. There is the concept of identity also in Hyperledger Fabric. Every member of the Hyperledger network has its own identity represented by the public key. The Fabric system generates the key pair using the Elliptic Curve cryptography. These keys are used to sign and verify transactions.

The approach chosen for our solution is to use two different identities for each user: a Crypto Cloud identity and a Hyperledger Fabric identity. Using two different key pairs, we need to be sure that every file-related transaction is executed from a verifiable identity. It is reasonable to publish the public key of users on the blockchain: in this way, everyone can verify that a user has that specific identity. No one can change the content of a transaction published on the blockchain, so the identity and public key relationship is permanently registered and accessible from everyone. In addition, we also put the signature of the transactor, done with the RSA private key of Crypto Cloud. Every time a user reads transactions from the blockchain, the client verifies that the signature in the transaction can be verifiable with the associated public key. Briefly, when a user enrolls to the system, the client creates an Enrollment Transaction depicted in Figure 7 (1).

D. Access Control List Integrity

If we do not want to trust the server, we must guarantee the integrity of the Access Control List. This is necessary because we have to ensure that users can not update it, adding entries of malicious users, if they are not on the list. When a user create a file, the client also creates the correspondent ACL. The problem arises when a user holds the Read Key (KR), so he/she can access the file, but he/she is not in the list of authorized users. Crypto Cloud assumes that the server is honest but curious. If we do not trust the server anymore, it could tamper with the Access Control List. Therefore he could add an entry corresponding to $\{KR\}_{PublicKeyServer}$. Now it can modify the list, adding users even if it does not have the permission to do that. So it is necessary to check if the list is also valid and not just the file. To solve this problem, we calculate the hash of the access list and publish it on the blockchain. Every time a user reads a file, the client calculates the hash of the ACL retrieved from the server and check if it is equal to the one retrieved from the blockchain. In this way, integrity is always confirmed. We must update the ACL when a user performs a "share" operation. To achieve the correct check, we put both the hash of the old version of the list and the hash of the new version. In this way, clients

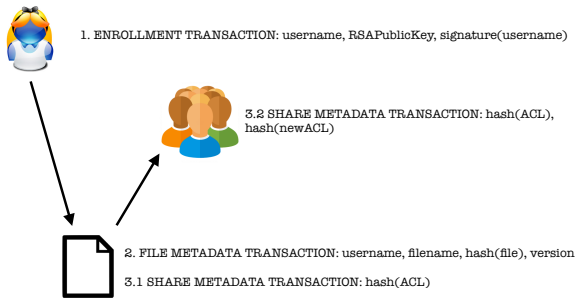

Fig. 7: Crypto Cloud Transactions overview

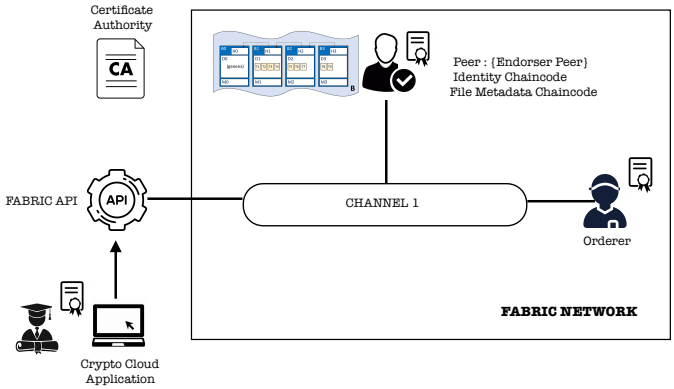

Fig. 8: Crypto Cloud Fabric Network

can always verify the validity of the old version and create a correspondence with the actual one. In other words, when a user adds a new file, the client creates a Share Metadata transaction, as depicted in Figure 7 (3.1). If a user updates an existing file, the client calls the Fabric API to create a new Share Metadata transaction, as depicted in Figure 7 (3.2).

E. Version control and integrity

Crypto Cloud uses the HMAC combined with hash and version of the file to guarantee the integrity and versioning of files. In the blockchain, it is not necessary to adopt that mechanism because everything is published on the ledger is signed, and no one can change the content of transactions. Managing the HMAC will result in a more complex operation because we have to manage also the symmetric key (Integrity Key) associated with it, so it is not useful. Therefore, it is possible to put the content hash of the file and version on a transaction, gaining the certainty that no one can change its content once published. In other words, when a user creates a new file, the client issues a File Metadata transaction, as depicted in Figure 7 (2). Every time a user wants to read a file, the application checks the correspondence between the hash of the file and version on the CCDS and File Metadata transaction. In this way, we are sure that what is on the server is always valid.

VI. Implementation

This section describes the implementation of the proposed Crypto Cloud system with Hyperledger Fabric. The purpose of this work is to create a Proof of Concept of the proposed solution to demonstrate the feasibility of the project. We consider a simplified version of the design, not considering the Access Control List integrity because of time constraints.

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

A. Hyperledger Fabric network

The first implementation step is the creation of the Hyperledger Fabric network. In the simplest scenario, we build an infrastructure which allows creating a complete blockchain environment and tests its potentiality. Therefore, we create a Peer, an Ordering Node and a Certificate Authority. If another University wants to use the Crypto Cloud Application, it will be necessary to add a Peer per University. It is necessary to create a channel allowing the communication between the entities. Figure 8 depicts the resulting Hyperledger Fabric infrastructure. To implement the proposed solution, we create two chaincodes: the Identity Chaincode, to put/get identities, and the File Metadata Chaincode, to put/get metadata of the managed files. In this project we decided to develop the chaincode using the Go language, referring to the examples on the documentation [15].

In the first chaincode, we defined the structure *Identity* which contains the username, the RSA Crypto Cloud public key and the signature of the username, done with the RSA private key of the CC user. It provides two functions: create identity function and query identity function which allows submitting transactions.

In the second chaincode we define the *FileMetadata* structure which is formed by the file name, the content hash, the version of the file and the username of the transactor. It provides two functions: create file metadata function and query file metadata function.

B. Fabric API

Once the Hyperledger network is created, we need to develop a client node which can communicate with it. The Hyperledger Fabric makes available a Fabric SDK Java to

facilitate Java applications to manage the lifecycle of Hyperledger channels and user chaincode. The SDK provides a means to execute user chaincode, query blocks and transactions on the channel, and monitor events. Our Fabric API allows interacting with a Certificate Authority and generating enrollment certificates. These identities are used to sign and verify transactions. These identities will be used when a user wants to query and update the ledger.

C. Users' registration

The system must provide a mechanism to validate the public keys and associated identities of users. When a new user joins the system, it is necessary to generate the keys which represent his/her identity. When a user wants to subscribe to the system, the client registers the user to the CCDS and enrolls him/her to Hyperledger Fabric Membership Service Provider. The subscription to the Certificate Authority returns the Hyperledger Fabric identity of the user. Then, the application creates a transaction which contains the Crypto Cloud identity calling the corresponding function of the Fabric API; it contains the username, the Crypto Cloud RSA public key of the user and a signature of the username.

D. Read operation

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

E. Write operation

The write protocol consists of uploading a local copy of a file to a registered cloud store. To create a new file the client application performs the cryptographic operations, guaranteeing the confidentiality and integrity of the file. After that, the protected file is uploaded to the cloud store using the Cloud Provider's API. After uploading, the Client Application sends a request to the CCDS with the new file's metadata and a request to the blockchain using our Fabric API. We create a transaction that contains the username, the file name, the public key of the user, the hash and the version of the file.

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

F. Share operation

The share operation allows a user to share a specific file with another user, called grantee user. When performing this operation, the user queries the blockchain, calling the Fabric API, and retrieves the last transaction related with the desired file; if the identity of the transactor is already verified it is not necessary to redo the verifications. Otherwise, the identity is retrieved from Hyperledger blockchain. If the signature is verifiable with the correspondent public key, the operation continues. Now the client verifies if the hash of the retrieved file is equal to the hash of the file on the blockchain and if the public key of the user is on the access list. Then, the client finds the grantee user's public key from the blockchain calling the query identity function of Fabric API. Then, it adds the user to the access list, adding $\{KR\}_{PublicKeyUser}$ and send a request to the CCDS with the updated information.

VII. Evaluation

This section presents the evaluation of Crypto Cloud with Hyperledger Fabric (CC-HL), describing the followed methodology and comparing the obtained results with the previous version of Crypto Cloud (CC).

VIII. Performance Evaluation

To evaluate the performance of our solution, several benchmark tests were carried out. The latencies measured from benchmarks were obtained using a profiler software, called JProfiler v10.0 [16], which performs the instrumentation of the running code on a Java Virtual Machine (JVM) and traces its information. This technique has a relatively low overhead associated [16]. The experiments have been performed over an Intel(R) Core(TM) i7 3230M CPU running at 2,5 GHz with TurboBoost technology enabled, with 16GB of DDR3 memory running at 2133MHz, and 256GB of SSD. The OS used was macOs Mojave 10.14.2 (x64) running standard services. For all experiments, the Client Application, the CCDS, the Fabric Network, and the PostgreSQL databases were deployed in the same machine as the benchmarks.

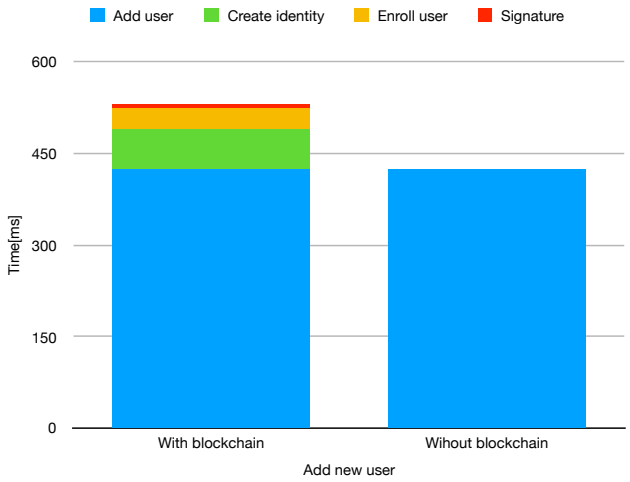

Fig. 9: Crypto Cloud's mean latency of add user

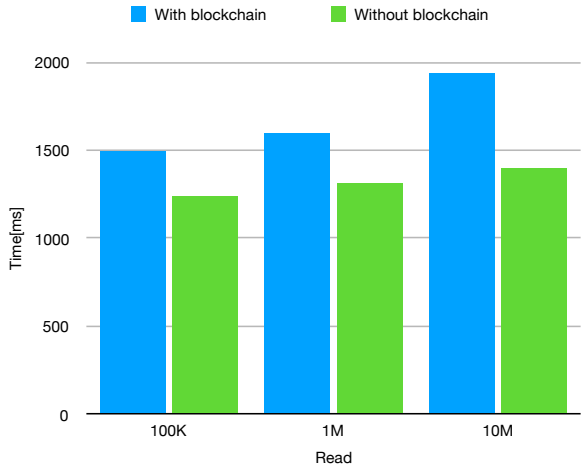

Fig. 10: Crypto Cloud's mean latency of read files

The Fabric instance consists of one Peer, one Ordering Node, a Certification Authority, a CouchDB and two chaincodes. Each Hyperledger Fabric entity runs into a Docker container. The benchmark tests consist of the Client Application performing several operation requests to the system. These operations include: reading a file from the cloud, writing a new file to the cloud, update an existing file and sharing a file with a user. Additionally, we also performed the same benchmarks using the existing Crypto Cloud prototype [1], in order to compare the obtained results. We modified the previous version of the Crypto Cloud application in order to generate and consume local RSA key pairs; so we do not use the KMIP protocol.

All performed benchmarks measured the latency times of each operation on our system. These operations were executed individually 30 times, with approximately 10 seconds of interval between them, and its mean time and standard deviation were analyzed. The experiments took place on January 2019.

A. *Crypto Cloud Performance*

In order to evaluate the performance of the Crypto Cloud system, we obtained latency measures for each of the Crypto Cloud operations using different file sizes: 100KB, 1MB and 10MB. During these benchmarks, we did not consider the latency times obtained from Cloud upload and download operations since these operations are performed outside of our controlled environment and we cannot guarantee that all cloud operations are performed under the same conditions.

The first operation of our interest is the addition of a user. As shown in Figure 9 in CC-HL the operation takes 25% more time than the same operation executed in CC. This is because when a user registers the first time to the system, he/she is both enrolled to the Hyperledger Membership Service Provider and to the CCDS. Moreover, the application puts the identity of the user on the ledger, submitting a transaction to the blockchain; this is done calling our Fabric API. The identity contains also the signature of the username of the transactor, so there is also an additional time due to the calculation of the signature.

The read operation consists of getting the file's content and metadata both from the CCDS and Hyperledger blockchain; then, the application unwraps the file's keys, decipher the content and checks its integrity. From the obtained results, depicted in Figure 10, we can observe that the computation time of this operation increases with the rise of the file size parameter. This is true for both the system and can be explained by the increase in the time during the decipher process. We can observe that the CC-HL read of files took 20% to 40% more time than the same operation in CC. Analyzing the time spent in each read call, we found that the retrieving of the file's metadata from the blockchain consumes a high percentage on the total time spent. The application also contacts the blockchain also to verify the authenticity of the transactions (retrieves the identity and checks the signature), if the identity was not already checked. In addition, in the new version, we retrieve the ACL and the public key of the user who made the transaction from the CCDS. These operations are necessary to check the consistency between the metadata held by server matches and the ones retrieved from the blockchain, resulting in an increase of time of the read operation.

Similar to the read operation, the write operation also depends on the file size parameter. As depicted in Figure 11, when performing the write of new files, CC-HL took 20% to 25% more than the Crypto Cloud's time. The time is higher because every time a file upload operation is issued, the application submits a transaction on the blockchain containing the content hash, the version of the file, the filename and the username of the transactor.

Figure 12 depicts the result of uploading of existing files (the file is modified every time). In the case of upload of an existing file, our Proof of Concept took 10% to 20% more than the CC time. This operation is different from the upload of a new file because the application also finds the last valid

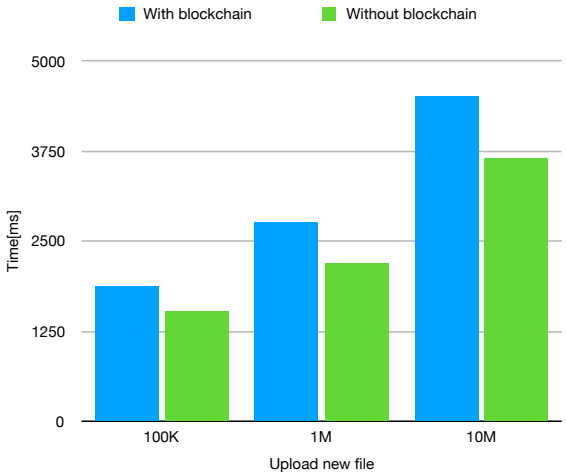

Fig. 11: Crypto Cloud's mean latency of upload new files

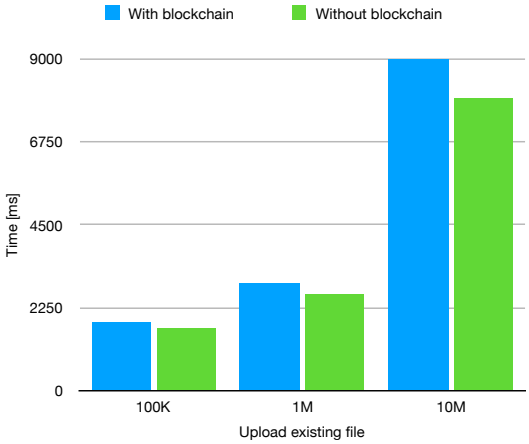

Fig. 12: Crypto Cloud's mean latency of upload existing files

transaction on the blockchain. This is necessary to check the integrity of the metadata held by the server. Moreover, the application verifies the identity of the transactor, which must be valid. To retrieve the public key and the ACL from the server, additional calls were done. All these operations contribute to the increase in the overall time.

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

IX. Conclusion

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

In this work, the integration of Crypto Cloud with Hyperledger Fabric blockchain was successfully implemented, except delivering the integrity of the Access Control List due to time constraints. The new system provides a mechanism to validate the public keys and associated identities of the users, using the blockchain to provide authenticity of the public keys and using the client to verify it. A new component was added to the Crypto Cloud architecture which is the Hyperledger Fabric network. We created two smart contracts which represent, respectively, users' identities and files' metadata. To interact with such smart contracts, we implemented an API using Hyperledger Fabric Java SDK [17].

In addition, the protection of the stored files is enhanced by using the blockchain. We add the metadata which guarantees the integrity of files on the blockchain; every time a user wants to read his/her files, the client application verifies the consistency between the metadata on the server and the ones on the blockchain. In this way, if the server contains tampered metadata, it will be detected and discarded. Regarding performance, the new Crypto Cloud add an overhead to the creation and reading of files: it takes 20% to 40% more time than the previous version. Nevertheless, we gain the advantage of not having to trust the server anymore, always guaranteeing that the metadata held by the server is not compromised.

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

X. Future Work

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

- Integrity of the Access Control List: the hash of the ACL must be stored on the blockchain to prevent that malicious users could modify it without the right permissions. The mechanism is explained in detail on Section V-D;

- Removal of the server: all the most important metadata were moved on the blockchain, so it will be possible to eliminate the server. The remaining metadata could be maintained on the client side.
- Create a distributed storage, based on blockchain technology: there are technologies like Filecoin, which allows users to sell and buy remote storage space for backing up their data. Adopting a technology as Filecoin, it will be possible to substitute the cloud providers with the Filecoin Decentralized Storage Network (see Section 4.3). In addition, it is possible to integrate the functionalities of our solution guaranteeing authenticity, integrity, and freshness of the files.

ACKNOWLEDGMENTS

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

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

REFERENCES

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

[2] S. Pereira, A. Alves, N. Santos, and R. Chaves, "Storekeeper: A security-enhanced cloud storage aggregation service," in *Reliable Distributed Systems (SRDS), 2016 IEEE 35th Symposium on*, pp. 111–120, IEEE, 2016.

[3] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, "An overview of blockchain technology: Architecture, consensus, and future trends," in *Big Data (BigData Congress), 2017 IEEE International Congress on*, pp. 557–564, IEEE, 2017.

[4] Z. Zheng, S. Xie, H.-N. Dai, X. Chen, and H. Wang, "Blockchain challenges and opportunities: A survey," *International Journal of Web and Grid Services*, vol. 14, p. 352, 10 2018.

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

[6] Z. Zheng, S. Xie, H.-N. Dai, X. Chen, and H. Wang, "Blockchain challenges and opportunities: A survey," *International Journal of Web and Grid Services*, vol. 14, no. 4, pp. 352–375, 2018.

[7] M. Castro, B. Liskov, *et al.*, "Practical byzantine fault tolerance," in *OSDI*, vol. 99, pp. 173–186, 1999.

[8] T. T. A. Dinh, J. Wang, G. Chen, R. Liu, B. C. Ooi, and K.-L. Tan, "Blockbench: A framework for analyzing private blockchains," in *Proceedings of the 2017 ACM International Conference on Management of Data*, pp. 1085–1100, ACM, 2017.

[9] P. Yee, "Updates to the internet x. 509 public key infrastructure certificate and certificate revocation list (crl) profile," 2013.

[10] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, *et al.*, "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *Proceedings of the Thirteenth EuroSys Conference*, p. 30, ACM, 2018.

[11] H. D. Initiative *et al.*, "What is ethereum?," tech. rep., retrieved 2017-05-14. Available: http://ethdocs.org/en/latest/introduction/what-is-ethereum.html.

[12] T. Tiwari, D. Starobinski, and A. Trachtenberg, "Distributed web mining of ethereum," in *International Symposium on Cyber Security Cryptography and Machine Learning*, pp. 38–54, Springer, 2018.

[13] P. Labs, "Filecoin: A decentralized storage network," 2017.

[14] J. B. P. Labs, "Filecoin research roadmap for 2017," 2017. https://filecoin.io/research-roadmap-2017.pdf.

[15] Hyperledger Fabric documentation. Available: https://hyperledger-fabric.readthedocs.io/en/release-1.3/.

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

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