Secure Auditable Cloud-backed File System

Filipe Miguel Marcos Apolinário
filipe.apolinario@tecnico.ulisboa.pt

Instituto Superior Técnico

Advisors: Professor Miguel Filipe Leitão Pardal
Professor Miguel Nuno Dias Alves Pupo Correia

Abstract Cloud infrastructures raise important concerns related to the confidentiality, integrity and availability of the outsourced data. This is due to the fact that when outsourcing data to the clouds, clients inevitably loose control of the data and thus all the data becomes controlled by the cloud service provider who administrates the cloud. These problems require clients to have a high level of trust regarding the cloud's service provider who may not have mechanisms to protect clients' data or may maliciously compromise it.
Currently, cloud-backed file systems do not provide complete solutions that address the aforementioned properties. This document presents a cloud-backed file system that addresses all the aforementioned properties and allows clients to work collaboratively and share files between each other. Furthermore, for clients that do not trust clouds with sensitive data, the solution employed allows clouds to prove to their clients that the outsourced data is safely stored. This raises the level of trust between clouds and clients and moves one step forward to bring cloud solutions to critical system such as governments or hospitals, to have their data stored on the cloud.
In this document, the proposed file system will be carefully explained, showing its structure and behavior and, carefully explaining the techniques and systems that motivate this proposal. Moreover, evaluation metrics will be proposed to evaluate this system and obtain clear results of the contributions to the commercial data storage clouds and work-plan will be proposed in order to fully deploy the first implementation of the this file system.

Keywords: Cloud computing, security, confidentiality, integrity, availability, file system, data as a service, cloud-of-clouds, proof-of-storage, homomorphic authenticators

Contents

1 Introduction

Cloud computing is a computational model that is emerging nowadays. In this model clients outsource the allocation and management of resources (hardware or software) that they rely upon to external entities known as *clouds*. Clouds are scalable infrastructures hosted on the Internet, capable of providing seemingly unbounded resources in an ubiquitous and scalable way, making their infrastructure invisible to the users when accessing their resources. Moreover clouds are administrated by a *cloud service provider* (CSP) responsible for maintaining and allocating clouds resources, selling them to the clients in the form of services. Such services are delivered to the clients with easy-to-use interfaces (GUIs[1] and APIs[2]) and are defined in contracts involving both the CSP and its clients. Furthermore, contracts bind CSPs with *service level agreements* (SLAs), to the *quality-of-service* (QoS) they provide, and bind clients with billing policies, for payment of such services [16].
Clients can demand from the CSP several resources by requesting different types of services.[16] For individual hardware components (such as CPUs, memory, etc.) clients request an *Infrastructure as a service* (IaaS) model. For aggregated components to run and execute computational environments clients request a *Platform as a Service* (PaaS). For databases and storage units, that store user data and provide maintenance of the underlying storage systems, clients request

[1] Graphical user interfaces
[2] Application Programming Interface

Data as a Service (DaaS). Finally for using fully built applications that are provided and maintained by the CSP, capable of running independently of the clients' environment, they request a *Software as a Service* (SaaS). This work will focus on DaaS models.

Currently, outsourcing data to the DaaS clouds provides several potential benefits to the clients. Flexible and powerful infrastructure capable of providing scalability on storage capacity and access independent of the clients geographical location. Optimality and dynamic storage allocation, allowing cost savings and worry free from both allocation and maintenance, thus ensuring: that clients do not pay for unused space or have increased costs related with the storage infrastructure; and that clients can focus on the tasks they want to accomplish independently of the storage infrastructure they use.
Despite the benefits of using DaaS clouds and their widespread growth, the current DaaS commercial clouds suffer from several problems. These problems hinder the provided QoS and, consequently, clients and their data,and are heavily related with the loss of control regarding clients and the fate of their data. This is due to the fact that when DaaS is employed, the client uses the service as is, thus the responsibility of mitigating these problems is delegated to the provider. This reduces client awareness and implies critical trust concerns on the provider, regarding the provider's activities and how he uses the client's data. Furthermore, due to the level of trust needed on the CSPs, these problems are an hazard for deploying DaaS to organizations that store sensitive data (such as governments and hospitals), which they need to protect at all costs. As a consequence in DaaS clouds, clients and their data are subjected to the following problems:

- High Communication costs - The communication between the clients and the clouds in the DaaS model, can be affected due to heavy latency and bandwidth consumption. Communication between clients and clouds are affected by latency because clouds are hosted on the Internet (may be far away from the client) and therefore the communication may take long times between the two endpoints and therefore violates ubiquitous and transparency making the user aware of this problem as well and may make delay user's urgent tasks and therefore hindering client's business that depends on the offloaded data. Aggravating these problems the bandwidth consumption needed by this model affects clients by inducing more communication delays between the client and clouds and thus may imply a latency increase, leading to the aforementioned problems.

- Weak consistency guarantees - Regarding consistency guarantees on storage operations, most DaaS commercial clouds nowadays only provide *eventual consistency* among operations. This means that, if no file operations are performed then all cloud servers will eventually have the most recent updated data, thus clients can read stale data and perform operations concurrent operation that may lead to an inconsistent state. This consistency model may not be suitable to all clients, specially those whose data is being constantly updated and need to keep up with the latest values. For instance, institutions

that offload the access control list of the building entrances to the cloud may suffer from the problems mentioned above. If the access control privileges are revoked to an employee and cloud servers are not consistent, the read access control list may not be the most recently updated version and thus intruders may surpass access control and enter the building.

- Security violations - Regarding security of the offloaded data, DaaS clouds are subjected to different types of threats. Some internal due to full control CSPs have on their storage resources, others external due to intrusions on the clouds and poor access control configuration of the clouds resources. These threats can have the following effects on the user data. First, data can be spied or leaked, by CSPs or intruders, to external unauthorized parties and thus posing confidentiality problems. Second, data can be tampered by these entities and thus posing integrity problems and possibly damage user data beyond repair. Third, access control can be poorly developed and allow non authorized users to access privileged data. Fourth, because CSPs can be guided to reduce costs, data non frequently used can be discarded in order to reclaim space and, this way guarantee more available resources to the CSPs while giving data loss to the clients. Finally, because contingency and reaction plans to such threats may not be effective or are non-existent, data may not be recoverable and thus violate QoS and hindering clients beyond repair if critical data is affected.

- Poor availability resilience - Regarding availability of the offloaded data in DaaS models, clients need to access their storage in order to perform their day-to-day tasks and therefore clouds should be available for accessing the data at all times. While there are several mechanisms known to provide these guarantees, there have been reports of commercial clouds subjected to problems of service availability and data loss. These problems are related to several potential server faults. First, servers may be subjected to *transient faults*, becoming unreachable due to shutdowns or power outages, and thus storage services may become temporarily unavailable. Second servers may be subjected to *crash faults*, due to overheating or malfunctions, leading to unrecoverable state on the cloud's resources and, thus may compromise severely the service's availability (until crashed servers are replaced) or may lead to huge data loss if data was not replicated to unaffected servers. Finally, servers can be subjected to *byzantine faults*, due to hardware malfunctions or possession by an adversary (attacker). They compromise service the same way transient and crash faults do, and also affected servers may send arbitrary responses to clients, tampering or deleting data, and thus raising trust problems between clients and clouds.

- Service dependency - Regarding service dependency problems of using DaaS models, clients are bound to billing policies and therefore they can be subjected to the *vendor lock-in* problem. This problem happens when cloud increases DaaS prices to unfordable costs, such that clients cannot retrieve

all their data from the clouds and, this way, become trapped and obliged to use CSPs services in order to access the necessary offloaded data. Currently billing policies do not protect clients from price changes or retrieving freely their information when such contracts are canceled, leaving client helpless when confronted with problems of this nature.

To address the aforementioned problems, the present document proposes a secure auditable DaaS cloud-backed file system for collaborative environments. To do so, this system is built on top of the SCFS file system that already provides an easily mountable collaborative file system capable of assuring, strong consistency on all file operations and, by outsourcing data to several commercial clouds using DEPSKY, assuring confidentiality, integrity and availability of all the stored data (as described in more detail in Section 4). Moreover, for clients that do not entirely trust clouds with sensitive data, the solution provides on top of the SCFS file system, a detection and recovery service to detect and reconstruct the original data when unauthorized modifications are performed. This way the solution proposed addresses consistency, confidentiality, integrity and availability of the outsourced data, while providing the necessary tools for raising client awareness of such problems, raise trust in CSPs and give more power to the clients, regarding the fate of their data.

The remainder of this document is structured as follows. Section 2 provides a detailed overview of the goals of the proposed solution relies upon and the expected results obtained in the evaluation phase. Section 3 analyses the existing systems and mechanisms present on the state-of-the-art that mitigate the aforementioned problems. Section 4 presents in a detailed form the architecture of the proposed solution and how the goals of the system are to be ensured. Section 5 proposes some evaluation methodologies for the proposed solution with the baseline and discusses the trade-off between the assured properties and the overheads imposed by the proposed solution. Section 6 provides a schedule for deploying the first version of the proposed solution, alongside with a dissertation presenting an in-depth discussion of the system developed. Section 7 concludes the document.

2 Objectives

This document addresses problems related with outsourcing data to the DaaS clouds. Moreover, the collaborative cloud-backed file system proposed is designed to address the following goals:

1. **Easy to use interface** - The solution must be easily mounted on the end-user's machine and require almost no user effort.
2. **Atomic semantics to all operations** - Data operations should always seem performed in a single instant in time. Moreover after a value is changed no other reads can show previous values of the change. Only the changed value or more recent values are allowed to be shown [22].
3. **Concurrency control between clients** - Any concurrent changes to shared data should only produce correct and consistent results, while generating results as quickly as possible [18].

4. **Confidentiality and key distribution** - Data outsourced to the clouds needs to be confidential and resilient to eavesdrops, without requiring the client to worry about key management.
5. **Data availability** - Outsourced data should be ready to be used at all times.
6. **Integrity** - All the outsourced data should protected, by the cloud, against unauthorized modifications.
7. **Minimize data stored on clients** - Cloud system should be designed to minimize the data needed to store on the client side, e.g. only store authentication keys, server locations and any other metadata lightweight required for using the proposed solution.
8. **Storage failure detection service** - Clients can detect when any block of their outsourced data is lost or corrupted irregardless on how regular is its use.
9. **Recovery data service** - Clients can recover data corrupted, lost or hosted in a cloud subjected to vendor-lock-in problems.
10. **Scalability and latency resilience** - The system must be designed to handle growing amount of outsourced data, clients or adaptation of its infrastructure, in order to accommodate that growth and providing mechanisms for allowing clients to be affected to the minimum possible, by latency of system operations [8].

3 Related Work

The solution proposed in this document leverages knowledge obtained from studying several concepts and systems from the current state-of-the-art. In this section an overview of those concepts and systems will be given, stating for each of them their advantages and disadvantages.

This section is structured as follows. Section 3.1 presents systems that provide to their clients DaaS cloud-backed storage using the *single cloud model*, where all the outsourced data is stored on one cloud. Section 3.2 presents systems that strengthen the data availability provided by DaaS cloud-backed storage using the *multiple cloud model*, where all the outsourced data is stored on several clouds. Subsection 3.3 presents DaaS cloud-backed storage systems that make use of multiple cloud model for improving availability and confidentiality of all the outsourced data. Subsection 3.4 focus on distributed file systems that can be mounted on the clients for interacting with DaaS clouds with little user intervention. Subsection 3.5 presents mechanisms for enabling clients to detect whenever outsourced data has been subjected to unauthorized modifications. Section 3.6 compares all the systems studied in the related work with the solution being presented in the current document, highlighting the benefits and weaknesses of these systems and carefully explaining how they influenced the crafting of the solution being proposed.

3.1 Single Cloud Storage

DaaS clouds are subjected to several problems regarding: confidentiality, integrity, access control, key distribution and trust concerns regarding the CSP's actions on the outsourced data. Although several solutions have been proposed in order to mitigate these problems, little to no commercial clouds provide solutions for fully mitigating these problems.

The remainder of this sub-section presents two systems: CloudProof and Cumulus. The two systems will be carefully explained, while highlighting the mechanisms they employ, benefits and weaknesses.

CloudProof: [24] is secure collaborative cloud storage service for storing and sharing data among several clients, that offers: (1) Rigid access control of all the outsourced data. (2) Key distribution to all the authorized clients for accessing all the shared data. (3) Integrity (I) of all the outsourced data. (4) *Write-serialization* (W) of all writes performed to shared data, i.e ensures that all concurrent write operations are performed in the same order and thus do not lead to inconsistent states. (5) *Freshness*(F) of all the data read from the storage service, i.e where no stale data is provided to the clients. (6) Attestations in all the operations performed in the cloud regarding the aforementioned IWF properties, for raising the trust on the cloud and detect whenever these properties are violated and whoever is responsible for such violation.

Regarding the assurance of IWF properties, CloudProof provides on each action performed to the storage service attestations, that serve as an acknowledgment that an authorized entity (data owner, authorized,client) has requested that operation. To do so, along with each request issued to cloud, a signature of the content of the request issued is provided, signed by the authorized entity using its private asymmetric key and also, for ensuring that clouds performed the requested action, along with the reply a signature of the content of each reply is provided signed by the cloud, using its private asymmetric key. Both signed messages are non-repudiable and allow to prove whenever one of these entities is violating the service agreements, by tracing the action and detecting which entity (data owner, user or cloud service provider) is responsible for the violation. Specifically, each IWF property is guaranteed differently by the attestation. For integrity, whenever data is written to the cloud the writer provides a cryptographic hash of the data signed by its asymmetric key; in order to prove that integrity is not violated. For write-serialization each attestation provides the file-block version where the operation was performed and the hash of that block. Finally, for ensuring freshness of the data sent to the clients, clouds need to prove that the sent data is not stale. To do so, in every attestation a *chained hash* is provided. This chained hash provided is an hash the data sent in the response and last previously sent chained hash. Therefore guaranteeing that any stale data received is easily detectable (by tracking previously performed operation on the chained hash structure).

Regarding access control CloudProof is designed to ensure that access control is not violated. To do so, this system uses *access control lists* (ACLs), for allowing data owners to specify which clients have access to the data, key encryption

techniques to enforce Read/Write access control to the data and *user access revocation* techniques for revoking access to authorized clients whenever an ACL is changed.

Data blocks stored in CloudProof with the same ACL form a block family, thus allowing the use of the same key for every block in the family. To enforce the read/write access, each block family has a read key, public verification key and a private signature key. Each block in a block family is encrypted with the same read key, to provide confidentiality of data. Blocks written are signed with the clients private signing key. Whenever a user sends an update of the block, the cloud verifies the integrity structure with public verification key and accepts if valid. If cloud accepts a non valid update, it is considered a violation and is easily proven by attestations mentioned above.

Data owners may want to revoke access of the user to a block family. CloudProof implements two types of revocation, immediate and lazy revocation. *Immediate revocation* is employed on the block by the owner. To do so, the owner changes the access control list of the given block and forces the re-mapping of the block to another block family. When the remapping is performed, the block is re-encrypted with the keys of the new family block and users who do not have access to the family block keys are immediately revoked. *Lazy revocation*, on the other hand, is applied by the owner on a block family. This assures that the revoked user will not have access to future updates of any block in the block family. With lazy revocation, blocks are re-encrypted with new keys when updated to new versions. To support this in a scalable and lightweight manner, all CloudProof family block read keys support *key-rotation*. This technique allows the creation of a sequence of keys, produced by an initial key and a master key, which is only known by the cloud. Only the cloud can produce next key in the sequence using the master key, while authorized clients who have access to a produced key can produce all prior keys. Therefore this method allows forward secrecy on generating keys, while allowing clients to use one key of the block family to access encrypted data of old block family keys [24].

Regarding key distribution, data owner can outsource the key-distribution of the keys needed for the access control to CloudProof, by employing *Broadcast Encryption*. Broadcast encryption allows the cloud to send the block family read keys to all its clients in block family ACL without sending having to send updates of the keys to each user individually. To do so clients in the ACL form a set, containing a different secret key for each user. When revoking privileges the new read key is encrypted and passed to a subset of the clients in the ACL (the clients that were not revoked), such that only the clients in that subset can decrypt the read key using its dedicated secret key[14,30].

CloudProof has major advantages: allows auditing, key-distribution with minimum effort to the data owner and access control to the cloud. However CloudProof has limitations, first CloudProof does not address the issue of untrusted users leaking block family keys to other entities and also to the cloud itself. Regarding Access control CloudProof addresses does not address access

control hierarchies, thus does not allow in owner to specify which blocks should a group of users be allowed to visit, without specifying each block individually.

Cumulus: Cumulus[26] is a system that provides backups (snapshots) over a cloud where no backups are supported. It allows the user to take several snapshots of his data over time and stores them in a space efficient way at the cloud, while requiring no server code to be executed in the cloud.

With the main goal of economizing storage space, Cumulus opts to aggregate small files belonging to the same snapshot into one large unique file (segment). In this way, a snapshot is composed of one or more segments and the corresponding segment metadata. Each segment metadata provides mechanisms for mapping the snapshots, segments and files but also provides ways to verify the integrity (using signed hashes) of the files within a segment and consequently the integrity of the snapshot.

Whenever a snapshot is added, the most recent snapshot is compared with the new one. Old segments that contain common unchanged files are reused (adding a reference on the segment to the new snapshot) and the changed files are aggregated into a new segment. This way, by reusing space Cumulus allows the storing of multiple snapshots in efficient way.

Although Cumulus handles space efficiency in file system and offers integrity guaranties, it has a limitation worth mentioning. Regarding the storage of files, Cumulus groups small files into single large files (segments); addressing both storage space problems and reducing drastically the latency regarding the transmission of a big set of small files. However by doing so, fetching one or a few small files implies fetching one or more segments (depending if the small files are located on the same segment or not). This deteriorates performance drastically because instead of fetching a file of 1MB (for example) the user has to fetch a whole segment which may be 1GB. This consumes bandwidth, and consequently undesired overhead latency.

3.2 Replicated Block Storage with No Confidentiality

When outsourcing data to the cloud, data availability becomes a problem that clients cannot easily control and need to comply with the clouds mechanisms for mitigating this problem. One way for clients to improve data availability under this setting is to outsource data to several clouds using the multiple cloud model. This allows clients to circumvent the availability problems of an affected cloud, by relying on the unaffected clouds for accessing the affected data.

The remainder of this subsection is will be focused on the systems that employ the multiple cloud model in order to improve availability of the outsourced data. These system will be carefully explained, while highlighting the techniques they employ, benefits and weaknesses.

RACS: [1] is a system that allows users to disperse their data over several cloud providers in order to be resilient against vendor lock-in problems, crash failures and data loss. RACS is designed to serve as a set of proxies, between the users and their cloud providers, and provide atomic guarantees in storage access. Each RACS proxy is coordinated by an external entity (Zookeeper [17]) and

assumes a *key-value store* (KVS) interface, where each value stored is associated under an unique key and read or updated using the unique key.

In order to provide the aforementioned guaranties, RACS encodes all outsourced data to the clouds using an *erasure coding* technique to allow the dispersal and redundancy of the clients data over the several cloud providers. Thus using erasure coding technique, each data block is encoded into x equal sized chunks ($size(data)/x$) and created m parity blocks of the same size of an x chunk (obtaining $n = x + m$ different chucks), where any $n - m$ chunks are sufficient to reconstruct the original data. The n chunks are then stored to n different cloud providers (one chuck per provider) and thus allowing replication of data and, thus guaranteeing availability of the outsourced data if at least $n - m$ clouds of the n used clouds are reachable.

There are three scenarios where RACS plays an active role over the fate of the users data. First, regarding vendor lock-in problems, RACS is able maintain the lowest price by detecting when a cloud is about to raise price, compare the price over all combinations of cloud providers and migrate the data to a new clouds if viable. Second, because RACS disperses the data over the clouds in a redundant way using erasure coding, RACS is able to provide crash tolerance and file recovery (if active clouds are able to provide all the segments of data split by the erasure code). Finally, regarding data access, RACS provides a KVS interface with the mapping needed for the clients to access their data dispersed in the several clouds.

Although RACS is able to provide the aforementioned guarantees, it comes with costs to the client. First, regarding storage cost, because RACS applies redundancy, user's data is replicated over the several clouds implying an increase in storage occupation (depending on the paying model it can be a costly approach). Second, because RACS is in client side and applies erasure code, the client has to perform more request in order to retrieve a file (number of segments needed to retrieve a file), thus raising user's access latency and bandwidth consumption.

RACS in overall solves the problem of crash tolerance availability, data loss and vendor lock-in in a cost effective way but imposes costs in the client.

Robust Data Sharing with Key-Value Stores: [3] implements a system that provides atomic storage access over multiple key value store storage providers (KVS) and unbounded number of clients, while having no single point of failure and being crash tolerant.

Designed to withstand any number of clients and a minority of KVS crashes, this solution provides a wait-free algorithm and allows multi-reader multi-writer storage access in asynchronous way. Clients in the presented system, are responsible for coordinating their operations and thus are required to contact KVSs and manage their results. To do this, clients can perform two types of operations read or write. These operations are based on versioning, write-backs, and because they are wait-free, are guaranteed to eventually respond to clients.

In this system, a write requires two steps to be performed by the client. First, the client needs to contact the majority of KVS and discover the latest version of the targeted key present in a majority of KVSs. Secondly (after determining the

latest version of the key), client issues a write request, with a new (incremented) version and the corresponding data, to a majority of the KVSs and waits for their response. If client has majority of success responses from the KVSs, client is assured that its data was safely written.

Besides writing, clients can perform read operations. Read operations play a crucial role in this system. This is because besides providing values to clients, write-backs are performed in all replicas, thus allowing for strong consistency in all replicas. Read require two steps to be performed by the client. First, the client contacts a majority of KVSs and requires their latest version of the key and the corresponding data, and waits for their response. Secondly (after determining the latest version and its data) client issues a write-back request for all the KVSs to update their value to the obtained version.

The proposed algorithm [3] has some important limitations. First, the fact that this system requires coordination by client on its performed operations, forces the client to contact a majority of KVSs at least two times by operation. Posing problems of latency and becoming an hazard in terms of scalability and overhead. Secondly, in this schema, because readers are required to perform write-backs to the KVSs, reader need write privileges for this operation and thus taking the role of writers. This last limitation, makes this system inviable for supporting collaborative environments where clients have different access privileges (only read or read and write).

PoWerStore: PoWerstore [12] is a system that provides atomic consistency, byzantine fault tolerance and integrity guarantees while maintaining the same performance as crash-tolerant solutions. It is based on a quorum protocol and designed to withstand Byzantine faulty servers (f servers if 2f +1 servers are active), any number of byzantine readers or crash faulty writers while providing atomic consistency and wait-freedom on read/write operations.

The protocol employs erasure coding to disperse data to the several servers, and a 2-round write procedure, Proof of Writing (PoW), prove that the write was successfully written before exposing to a reader. In addition, by using cryptographic hashes in PoWs, this protocol also assures the integrity of all the blocks written in PoWerstore.

PoWerstore allows two operations, write and read, both require clients to send two rounds. For writing, a client generates a nonce and on the first round writes a cryptographic hash of the nonce together with its data to a quorum of servers. On the second round writer discloses the nonce to a quorum. For reading, a client fetches the nonce from the quorum and sends (write-back) the nonce to a quorum of servers and matches the hash of the stored nonce.

3.3 Replicated Block Storage with Confidentiality

The systems that use the single cloud model for providing DaaS cloud-backed solutions are subjected to several problems regarding: confidentiality, integrity, availability, access control, key distribution, availability and trust concerns regarding the CSP's actions on the outsourced data. One possible solution for improving the availability of data is to use the multiple cloud model as shown in

the Subsection 3.2. Besides improving availability of the outsourced data, the multiple cloud model may also improve confidentiality, key distribution and trust concerns regarding the CSP's actions on the outsourced data.

The remainder of this subsection is will be focused on the systems that employ the multiple cloud model in order to improve availability and complement it with confidentiality mechanisms to allow high availability and confidentiality of all the outsourced data. These system will be carefully explained, while highlighting the techniques they employ, benefits and weaknesses.

Secret Sharing Made Short: [19] presents a *secret-sharing scheme* using an *m-threshold scheme*, "where m shares recover the secret but $m-1$ shares give no computational information on the secret, where $m \geq size(secret)$". The presented secret-sharing scheme relies on encryption and information dispersal, allowing users to disperse files over several remote entities while providing confidentiality and perfect forward secrecy.

This paper also provides a robust version of the m-threshold scheme capable of assuring that when less than half of the total entities are malicious, the user is always able to retrieve from m well behaved entities its correct data using secret sharing scheme (m is the number of shares needed to retrieve its data). This solution applies public key signatures for fingerprinting to the distribution protocol, appending to each encrypted data on the cloud the corresponding user fingerprint; this way assuring integrity of the data and byzantine fault tolerance on m-threshold scheme.

POTSHARDS: [25] is an archival storage that provides a long lifetime confidentiality, transient availability of data and integrity of the stored data. Confidentiality is achieved by using secret sharing scheme combined with an information dispersal scheme based on a secret splitting technique. Transient availability is achieved through storing redundantly data with the secret sharing scheme over several archives and through a *Redundant Array of Independent Disks* (RAID) storage technique, that stores information redundantly over several disks within each server in order to tolerate some disk failures. Integrity is achieved signing the splits of the secret with an algebraic signature, allowing to prove, without exposing the data, that the data is retrievable and its contents have not been tampered. Furthermore POTSHARDS assumes that archives are prone to be untrusted but an aggregate of archives is trustworthy.

To enforce that data is only viewable by authorized readers , POTSHARDS uses secret-splitting combined with approximate pointers. Secret-splitting method allows provable confidentiality instead of computational confidentiality, thus making sure that no adversary is able to reconstitute the original secret if he does not obtain the all the necessary data shares, regardless of the amount of computing resources he has.

To enforce transient availability POTSHARDS employs RAID erasure coding in order to distribute redundant shares among archives. This way support transient faults. Integrity checking of secret shares is provided by the archives in two ways: hash checking for intra-archive verification and a protocol based on algebraic signatures, for detecting integrity violations on redundant archives. This protocol

relies RAID the erasure coding (where each archive that belongs to the RAID group receives on of the redundant share excepting one that receives the parity) and works as follows: archive request to each member of a redundancy group an algebraic signature over a specific interval of data; then the requester compares the parity of the all algebraic signatures of the redundant share (XOR'd) with the signature of the parity of the blocks; if equal integrity is assured if not integrity is violated.

POTSHARDS has limitations. First, it relies to much on the archives and assumes adversaries are internal to the archive and adversary only controls individual archives. Because integrity verification and detection of malicious insiders is delegated to the archives (by the hash and by the approximate pointers) if archives do not report malicious intrusions, approximate pointers could compromise information without user knowing it. Secondly, data scalability in POTSHARDS does not hold because it performs secret splitting to data itself making data overall stored size of the stored secret(S) times the number of secret shares(n), furthermore because it employs RAID on secret shares stored in each archive to guarantee redundancy of the shares, stored space becomes $S * n * m$ where m is the number of segments needed to retrieve a fragment generated by the erasure code.

Belisarius: Belisarius[23] is a distributed KVS storage system that provides confidentiality, integrity and availability, by combining a byzantine fault tolerant protocol with a verifiable secret sharing scheme, if only f out of the total (n=3f+1) system storage units are faulty. Furthermore, Belisarius is asynchronous and withstands any number of byzantine readers and strong adversaries capable of coordinating faulty storage units and requires no server code.

The secret sharing scheme is composed of two modules: secret splitting and secret sharing. First, the client performs secret splitting, client first generates n shares of the original data (secret), using n points from a polynomial of degree $t-1$ (where $t > f$) and at least t shares are needed to recompute the share. Then, disperses n shares to the n servers using a verifiable byzantine fault tolerant (BFT) quorum secret sharing. The BFT quorum sharing scheme employed by Belisarius. Whenever the client wants to retrieve the secret, it broadcasts a request for the shares and waits for the response of a quorum of $2f + 2$ servers. Then computes all combinations of the shares received until it has gathered $f + 2$ identical combinations. By doing so, the client knows which shares are suitable for computing the secret and which shares are not. Furthermore, by tracing them to the servers, client can find which servers are malicious.

This system has good properties and provides information security without key management and integrity checking preserving homomorphism. However because this is uses verifiable secret sharing, this system does not scale well in terms of space. This is because, for each secret the amount of space needed to store in Belisarius is the size of secret times the number of servers (each share is at least the size of the secret as reported in [19].

DEPSKY: [5] is a storage cloud-of-clouds system, built with commercial storage clouds as base objects. DEPSKY provides proportional consistency storage

(where consistency of whole system is the same as the weaker consistency base object) and tolerance against: loss of availability (due to outages or byzantine fault tolerance); loss of privacy (confidentiality and integrity); and vendor lock-in. It requires no server code and is provided to clients as a software library to allow operations read, write, list, create and delete on data stored on the several cloud storage base objects. In this setting, a client can use DEPSKY to access all his commercial clouds in order to provide the aforementioned guarantees to all its data and by configuring each commercial cloud's own access control list (ACL) regulate which other clients have access to the each client's data. Among the features that the DEPSKY system provides, the feature that influences the solution proposed in the present document is the protocol DEPSKY-CA:a byzantine fault tolerance protocol to provide availability of the stored data, storage efficiency and confidentiality. To do this, DEPSKY-CA relies on secret sharing-schemes with erasure codes, along side with a byzantine quorum protocol of $3f + 1$ clouds, where f can be subjected to a byzantine, crash, or transient fault. This protocol is divided into two algorithms, the write algorithm and the read algorithm.

Regarding the write algorithm, writing data to the base object clouds is as follows: First, the client generates a random symmetric AES key and encrypts the data. Second, the client partitions key and cyphered data each into a set of $3f + 1$ shares (requiring $2f + 1$ shares to be collected to reconstruct the key or data). For the key the secret sharing scheme presented [19] is used, while with cyphered data erasure-codes are used. Third, shares are written to the $3f + 1$ clouds, each of them receiving an unique data and key share. Finally, when the client receives confirmation of successful storage from the $3f + 1$ clouds, client writes metadata corresponding to the data (version and digest) to $2f + 1$ clouds and signed with the clients private key to ensure integrity.

Regarding the read algorithm, the DEPSKY-CA first fetches the metadata files from $2f + 1$ clouds. Chooses the highest version between all valid signed metadata and fetches the key and data shares from the $3f + 1$ clouds. When shares were received from $2f + 1$ clouds, it reconstructs the key and cyphered data corresponding to the digest in the metadata. Finally when a valid data content is read, user cancels all pending requests to other clouds and returns the data.

The DEPSKY-CA protocol has two characteristics worth mentioning. Regarding storage costs, DEPSKY-CA requires: extra replication cost when storing the cyphered data, because erasure codes are used data shares have size $(data.length/(2f + 1)) * (3f + 1) = data.length$; and $key.length * 3f + 1$. This overhead seams reasonable for providing byzantine availability, confidentiality, vendor lock-in of the data stored on the clouds, and tolerate any leakage problems of data or keys originated from up to f clouds. Regarding the use of metadata, it addresses: integrity problems by using signed messages and digests of the data sent to the cloud; and to concurrent (read/write) operations, because metadata is only read if the data has already been written to the cloud (writer waits for the clouds to report status of the data before writing metadata).

Aside from the assured properties, DEPSKY still has some limitations. First, for writing data, $3f + 1$ clouds need to be contacted at least two times, which may lead to increased communication time overhead. Secondly, consistency of the system is the same of the weaker cloud. Thus, for clients that may need all strong consistency models all their base objects need to provide strong consistency, or they need to resort to external entities for coordination such as Zookeeper[17] or DepSpace [7].

Hybris: [13] is an hybrid cloud KVS storage system that stores data in private and public clouds and ensures byzantine fault tolerant availability, confidentiality, integrity, multi-writer multi-reader and atomic consistency of data. Hybris uses a trusted private cloud, located on safe premises, to store metadata (cryptographic hash, timestamp of the last write and encryption key) of each key and public cloud for storing the data of each key. Public clouds provide eventual consistency (if no writes are performed then all replicas will eventually stabilize) while private cloud provide atomic consistency, in order to provide atomic consistency, Hybris relies on Zookeeper[17] to coordinate public and private clouds.

Regarding integrity, Hybris stores cryptographic hashes on the private cloud (which is assumed trusted) of each data placed on the public clouds.

Regarding confidentiality, Hybris encrypts data with symmetric keys, erasure codes the encrypted data into $m + x$ (where m is the number of additional parity blocks and x is the number of partitioned blocks) and disperses them to the several public clouds, storing the symmetric key on the private clouds.

Regarding availability, Hybris uses requires at least $n = 2f + 1$ public clouds for tolerating f faults. For each operation Hybris works as follows. For writing a value(v) under key k the writer: obtains the last time-stamp of the last write of k from the private clouds; generates a fresher time-stamp; writer then encrypts data with symmetric key; erasure codes the data to $f + x$ clouds and waits for their response; and (if they do not respond in configured timer), writer sends to f more clouds totaling of $2f + 1$ clouds involved; finally checks if generated timestamp is still fresher than the one stored in the KVS (to avoid stale timestamps) and if so sends generated timestamp to the private cloud under key k. For reading, the client obtains timestamp and cryptographic hash of key k from the private cloud; and fetches from $f + 1$ clouds where the data was written the data; if reconstructed and decrypted data passes the cryptographic hash verification, then the value is returned; else fetches the data from the remaining clouds sequentially until the correct value is found.

This system has some limitations. First, regarding private clouds because they do not have the same scalability as public clouds (both in terms of space and in terms performance) and are required in every operation they can become a bottleneck, in terms of latency regarding location and regarding unbounded growth in the number of clients. Moreover, because its infrastructure they are normally localized in a specific region they are prone to outages and may compromise the availability of the whole system. Also, because they do not have the same replication capabilities of public clouds and do not scale well, over the years metadata can become difficult to manage and can be lost (which will make clients

unable to access their encrypted data and map their data to the public clouds). Furthermore this solution is not applicable for collaborative environments where data is shared by different entities each one with a different private cloud.

3.4 Distributed and Cloud File Systems

In order to abstract and automate the process of storage and retrieval of data into DaaS clouds data, easy to use interfaces need to be provided to clients. One possible way to automate the process is to offer interfaces that the clients are familiarized with, such as file systems.

In the remainder of this section, three distributed file systems that can offer easy-to-use interfaces will be at focus. Presenting the mechanisms they offer and the properties they address.

BlueSky: [27] is cloud-backed file system that outsources the storage of collaborative file system to the cloud, using a cache based proxy to serialize the updates of the several clients and provide incremental updates to the cloud storage. BlueSky provides confidentiality and integrity of all the offloaded data while relying on clients to provide key management and on clouds to provide robust availability guarantees.

BlueSky is composed by four entities clients, cloud and a proxy. Clients access data and make regular updates to the collaborative file system, by iterating with the proxy. Proxy receives all the clients' updates and serializes them into an incremental ordered updates. For each update on the file system, proxy updates its cache and signals the client the commit operation. Then performs write-back of the update to the cloud.

The BlueSky file system is composed of four objects for representing data and metadata: data blocks; inodes; inode maps; and checkpoints. These objects are aggregated into log segments for storage efficiency. Files are broken apart into fixed-sized data blocks and the corresponding file metadata (including: ownership; ACL; time-stamps; and the pointers to the data blocks) is stored in the corresponding file inode. Inode maps list the locations in the aggregated log segment of the most recent version of each inode. Furthermore, checkpoint objects store the root file system containing pointers to the most recent versions of the inode maps and provides pointer to the last previous checkpoint object, thus allowing versioned backup.

Confidentiality and integrity of the offloaded data is provided through, respectively, symmetric key encryption using AES on the objects' payload (data blocks; inodes; inode maps; and checkpoints) and keyed message authenticated codes.

BlueSky successfully provides a file system abstraction that allows multiple clients to work on a cloud file system, almost seamlessly as working on local access network file system environment, however it has some limitations. First, the proxy is a single point of failure and thus if proxy crashes, all updates that have not been sent to the cloud are forever lost and furthermore until a new proxy is available clients cannot access BlueSky. Second, because data encryption is used to store data in the cloud using symmetric key encryption, which is computationally secure, meaning that an adversary that gets the encrypted

data from the cloud can disclose the information after some certain time, thus not allowing long-term confidentiality. Finally because key management is not delegated to the cloud, clients are delegated to perform that task meaning that persistent storage of key may be at risk.

Cryptree: [15] is a cryptographic tree for providing confidentiality, integrity and access control of a file system stored on an untrusted storage. In Cryptree access control is provided by hierarchical access control lists (ACLs) (hierarchical in a sense that users with access granted to a folder can inspect all the subfolders and files of that folder). There are two access privileges read and write access.

In order to provide read access control in Cryptree, users in the read ACL of the stored object have access to the clearance control key (a master key) of the stored object (file or folder). This key grants access to the other keys of the respective stored object (symmetric keys for data and meta-data encryptions and link keys for linking the object to sub-objects) and allows users to traverse folders they have access to and have read access to the sub-folder or sub-files contained within the folder they have access to.

Regarding write access and integrity of the stored objects, each object has a corresponding private/public key pair assigned, respectively, for signing and verification on the signature of the written object. The signing key is only accessible to the writers, while the verification key, on the other hand, is accessible to everyone for ensuring the integrity of the object.In order to provide write access to users in the write ACL, each stored object has a writing clearance control key. This key gives access to the signature keys needed to write on the stored objects as well as on all the objects in lower hierarchies of that object (sub-files and subfolders).

The last note worthy mechanism is the revocation privileges of the user. Revoking privileges of a user to a stored object can be triggered in two ways: user is removed from the access control list and therefore the keys of the lower hierarchy objects of the folder are marked for rotation; or the lower hierarchy objects, which user is not a member of the ACL, are moved to other directories, where user is not a member of the ACL, and therefore user is revoked (immediate revocation).Additionally, if the user has write access, revoking the user implies changing public/private keys of the objects in the lower hierarchies.

Cryptree has limitations. First the key renewal and lazy revocation employed in this scheme has the problem of an attacker uncovering the clearance key of some stored object and this way obtain access to all the lower level hierarchy stored objects. If root folder clearance key is obtained attacker has access to all the stored objects. Also, the cost related to key re-computation when revoking a user can be expensive according to the number of files present in the lower hierarchies. Finally clearance keys need to be managed by the users, in case an organization looses the clearance keys data is forever lost.

SCFS: SCFS [6] is collaborative cloud-backed-storage file system that ensures strong consistency storage, access control, availability, confidentiality and integrity of data over weakly consistent storage clouds in a cost efficient way. This system is designed as three base entities: storage services (storage clouds), for storing

and maintaining file data; coordinators, for storing and managing metadata and synchronize operations; and SCFS Agent, placed in the clients for abstracting the iterations needed to the other SCFS entities in a POSIX fashion API file system.

For providing strong consistency SCFS ensures that synchronization between colliding operations (write and read) is achieved. To do this, SCFS relies on: synchronization, provided by coordinators (such as Zookeeper or DepSpace) and the metadata stored in these entities, for concurrent write-read operations; and file locking for concurrent write operations. Regarding concurrent write-read operations, coordinators provide strong consistency storage and management of metadata, serving as *consistency anchors* in the operations performed by SCFS agents. This consistency anchors guarantee that the whole coordinated system follows the same consistency they provide, even if the coordinated objects (cloud storage) only offer weaker consistencies. The metadata, on the other-hand, gives information of the files stored in the clouds and is composed of a file identifier and an hash of the file. The coordination takes place during two file operations: writing and reading. When writing a file, the writer first computes the hash of the file and writes the value and the corresponding metadata to several stored clouds, and, finally, stores the metadata on the coordinator. For reading, the reader first fetches the metadata from the coordinator and then requests the data corresponding to the metadata from the storage clouds until it is received. By this method, read-write concurrency is mitigated because: reading operations will only finish when the data corresponding to the metadata is on the cloud (after the write); and colliding writes will have no effect on previous reads since metadata (and more specifically the hash) is different. Regarding concurrent write operations, SCFS makes use of file locking provided in the SCFS Agent by external coordination services like Zookeeper and DepSpace. Whenever write operations to storage clouds are eminent a write lock is obtained by the writer for the metadata and for the storage data, implying that all other writers have to wait until the lock is released and thus, with this method, atomicity on write operations is achieved.

For collaborative environment between multiple clients, SCFS allows clients to share files by stipulating access control to stored files. SCFS provides access control lists relying on the storage services and the coordinator services access control implementations. To do this, each client has associated a list of cloud canonical identifiers which is kept in the coordination service. Whenever a client (O) wants to give access to an other user (U) several steps need to be taken. First, O updates the ACL's of the storage services that stores the file he wants to give access to, with new permissions to the corresponding U's canonical identifier. Then O updates the ACL's metadata associated with the file on the coordination services.

Regarding availability, confidentiality and integrity of data SCFS is plugged into several storage backend clouds, using the DEPSKY-CA for providing these properties to the data stored on the commercial clouds (as explained in Section 3.3) and relying on one of the two supported coordination services: Zookeeper

or DepSpace; to provide small byzantine fault tolerant storage for storing the metadata.

Regarding client interaction with SCFS, each client has a dedicated SCFS Agent. This agent encapsulates in a POSIX fashion API file system all the communication to the other SCFS entities. The SCFS Agent is built with two key ideas: avoid reads to the cloud whenever possible, by providing caching mechanisms on the client for metadata and data in order to lower cost and latency as much as possible; and delay writes to the cloud until the file is closed, in order to aggregate them into a single write to avoid overloading the SCFS with writes and avoid some unnecessary costs and latency. With these key ideas in mind, SCFS Agent provides four POSIX operations to a file: open; read; write; close.

Because coordination services are a potential bottleneck, for scalability purposes, SCFS avoids contacting coordination services when no synchronization is required, i.e when files are not shared. To do this, SCFS divides client's files in two categories: shared and private. For shared files, data is stored as explained early, metadata is stored on coordination services and storage services, while data is stored on storage service. However, for private files (files that are used only by the user) synchronization is not needed, thus SCFS aggregates all private metadata into a single object and puts it into the storage services along with the data. This object is referenced by a private name space object stored both on the client and on the coordination services and changed whenever an ACL of a file changes. Because private name space is stored in the client, there is no need to contact coordination services whenever file operations are performed and thus avoiding the bottleneck of such services on private files, which according to the paper are the large majority of files stored in the cloud.

On overall SCFS performs guaranties strong consistency storage, access control, durability, availability, confidentiality and integrity of data over weakly consistent storage clouds while being efficient, effective and scalable for the majority of files. However, SCFS has limitations one of which related to shared files. Since SCFS centralizes coordination coordination services, if a majority of files stored are collaborative overhead of lock synchronization can become expensive and thus, not scalable for collaborative environments. Also, one possible way to mitigate the bottleneck of the coordination system is to have more than one of these systems and map the clients according to some criteria to the several coordination systems. This solution would remove the centralized coordination service, but would raise network partition make it hard for clients mapped to different coordination services to share files among each other.

3.5 Proof of Storage

Outsourcing data storage to other entities (such as clouds) may lead to several problems when regarding the outsourced data. For detecting integrity and availability problems of the outsourced data, clients can use proofs of storage for this task. These proofs serve as guarantees to the client, that the outsourced data is safely stored (and retrievable) and that the original content of the file

was not affected by any unauthorized operations performed. One way to provide these proofs is by using *homomorphic authenticators* in *homomorphic linear identification protocols*.

Homomorphic Authenticators: are authentication schemes, such as *homomorphic message authentication codes* and *homomorphic signatures*, used for proving integrity of messages. These authentication schemes benefit from being truly *homomorphic*, such that it becomes possible to perform linear computations over encrypted data, without having to access the decrypted data [11]. This allows, in the client-cloud setting, clients to tag outsourced files with these authentication schemes using their keys (symmetric pseudo-random functions in case of homomorphic message authentication codes and asymmetric or symmetric pseudo-random functions in case of homomorphic signatures) and this way clouds can prove storage without having to know the clients' keys.

Homomorphic Linear Authenticators Identification Protocols: are protocols that involve both the clients and the external entity for providing proof of storage of all the outsourced files. To do so, as described in [2], the generic protocol is as follows. First, before outsourcing the files, the client performs the *tagging process*, where each file-block f_i to be outsourced is tagged using an homomorphic linear authenticator θ_i based on the file-block content and a pseudo-random function, and sent to the external entity with the outsourced block, serving has signature/message authentication of the block and for later proving its integrity. Furthermore, for each file block tagged by the client a verification state metadata is obtained and stored locally on the client. Secondly, whenever a client wants to obtain proof of storage of the file f composed of file-blocks $f_i...f_n$, client sends to the external entity a challenge composed of a random numbers, one for each file block of the file $c = c_i...c_n$. The external entity then computes a linear combination $\mu = \sum_i^n = v_i * f_i$, using the file-blocks $f_i...f_n$ and the challenge received $c_i...c_n$, and an *homomorphic linear aggregator* ($\theta = \prod_i^n \theta_i^{v_i}$), using the challenge and the homomorphic linear authenticators of each file-block, and sends it to the client as proof of storage. Finally, the proof is then validated by the client, against the challenge sent and verification state metadata (computed in the tag process), and the pseudo-random function used as key on the tagging process, and thus detected if integrity is preserved in every file-block.

The homomorphic linear authenticators identification protocols (HLA protocols) are used in order to obtain *probabilistic proofs of storage* of the whole outsourced data, while providing constant overhead costs in communication regardless of the size of the storage.

1. Regarding the **probabilistic property**: in order to obtain proof of storage of the whole outsourced data, clients need to issue a challenge for each outsourced file to the external entity; instead clients can generate a random sample of files from the outsourced data and for each file of the sample challenge the external entity. The higher the sample, the more guarantees the clients have that the outsourced data is safely stored, but more external entities resources are needed to compute the proofs (in clouds, this may

imply an increase on computational costs). Moreover, because it is a random sample, clients can have a quantifiable measure of how safely the file is stored, independently on how regular is its usage.

2. Regarding the **communication costs**: since the homomorphic linear aggregator allows to prove integrity of the whole file without having to compare with the data itself, and only depends on the homomorphic authenticators of each file block and the challenge issued from the clients (which are constant size per block), these protocols allows a substantial reduce on bandwidth consumption and allows constant overhead in communication which makes them more viable for acquiring proof of storage than other commonly used methods such as hash functions, where data is used in validation and thus needs to be sent along with the proof. Furthermore, for reducing costs while providing integrity in the communication of the homomorphic aggregator sent to the clients, an efficient combination scheme needs to be employed for supporting the *batch* of proofs. To do this, one commonly used mechanism is to use a signature scheme with a Merkle Hash Tree [21], as proposed in [28], for sending several authenticators in a message with only one signature for authenticating them. To do so, each homomorphic aggregator is hashed and concatenated with one another, to form a pair, and hashed again, repeating the same process with the resulting hashes until an unique hash is formed. The hash is then signed by the external party and provided along with the all homomorphic authenticators, thus guaranteeing low communication cost while ensuring integrity in all communication.

In order to produce the Homomorphic authenticator tags needed for proof of storage using HLA protocols, clients need to use keys. The keys used for tagging divide the HLA protocols into two categories: the *private-verifiable schemes*, where only the client and the entities the client gives access to a secret key can verify the proofs; and the *public-verifiable scheme* where anyone that possesses the public key of the client can verify the proofs. One advantage of the public verifiable schemes is the possibility of allowing *audits performed by third parties* [29,28,10], where the client is released of the burden of challenging and verifying external entities by contracting a third party entity to perform that task.

The remainder of this section will present two systems that provide proof of storage: HAIL[9], that uses private-verifiable schemes based on MACs; and PANDA[10] that uses public-verifiable schemes based on cryptographic hash functions signed with asymmetric keys. Finally, a study [29] will be mentioned on how to provide storage failure detection and recovery by combining erasure-codes with proof-of-storage so that it could be applied to multiple external entities in the cloud-of-clouds model.

HAIL: this system provides an highly available integrity layer for cloud storage for providing resilience to attacks targeting least frequently used files, by combining Reed Solomon erasure-codes with a private-verifiable HLA protocol using MACs in order to allow detection and recovery of the outsourced content. To do so, each file is erasure-coded into n shares, each share sent to n servers (where m can be faulty), and each block of the share is tagged using a MAC (each

with a secret key shared between the user and corresponding server) and sent to the server, while keeping verification state metadata locally. From time to time, clients send a challenge to every server in order to test if the file is retrievable in every server. Each server then computes an homomorphic linear aggregator, using the shares' content, MAC and the corresponding challenge (received from the client) and sends the result to the client as proof-of-storage. The client then validates the proof received and if any proof fails (m can fail), client recovers the shares affected, by recomputing the shares using the servers that were not affected by corruptions.

HAIL allows clients to have guarantees that integrity of the outsourced content is preserved. This approach could be easily adopted into the cloud-of-clouds model where each server is substituted into a different CSP and thus allowing less dependability against vendor lock-in and resistance against outages. However, HAIL for being a private-verification scheme does not allow clients to outsource the verification and validation process to third parties and thus, may have increase in costs regarding the communication overhead and the monetary cost for performing the challenge requests.

PANDA: is a mechanism that uses public-verifiable an HLA protocol to allow collaborative cloud-backed file systems to prove integrity of any outsourced data. In this environment, clients (data owners) share files among other clients (authorized users) by specifying write and/or read access to each authorized user using ACLs. These ACLs, can be modified at any moment to add or revoke access to the clients.

In order to build proof-of-storage using PANDA and guarantee integrity, each client has an unique asymmetric public/private key pair and stores alongside with each outsourced file-block a cryptographic hash signed with a the client's private key. This signature is then used by the client alongside with metadata publicly available to tag the file with an homomorphic authenticator and send it to the cloud. Whenever any modification is performed to an outsourced file-block, the writer updates the signature with the new content and signs it with his private key, updating also the homomorphic authenticator with the new signature.

Whenever clients want to acquire proof-of-storage for a given set of files, clients (or third party auditors assigned to delegated to perform the task on clients behalf) challenge the cloud using a random computed challenge vector. Clouds then generate the homomorphic aggregator and a linear combination using the challenge issued and each file-block's: content; homomorphic authenticator; and the corresponding writer's public key. The homomorphic aggregator and linear combination are then sent to the client and, verified (by the client) that the homomorphic aggregator is valid for each file-block using the linear combination, the issued challenge, public metadata and the corresponding writer's public key.

Regarding revoking access of a writer to a file, when PANDA is employed, implies that all file-blocks that have signed homomorphic linear authenticators by the revoked client are now untrusted and need to be reconstructed by an authorized client. PANDA deals with this problem, by employing *proxy re-signatures* that allow the cloud to act as a translator between revoked user and

an authorized user, thus allowing to re-sign revoked client's file-blocks with new signatures of an authorized user. The cloud, however, does not learn any signing key in the process of re-signature and cannot sign arbitrary messages on behalf of either the revoked client or the authorized one. This method then allows the cloud to recompute revoked data blocks without involving any authorized client.

As an overall this mechanism allows efficient and scalable public auditable clouds in scenarios where data is constantly changed and access revocation of clients are common. However it does have a concern that is worth mentioning, since clouds are able to re-sign data without clients awareness, clouds can re-sign data with authorized clients and can generate problems of repudiable signatures, where clients say that the data written and signed on the clients behalf was not signed by the client but re-signed by the cloud. This problem may be problematic is some systems that do not trust clouds entirely.

Ensuring Data Storage Security in Cloud Computing: The approach followed by [29] is a proposed idea for allowing public-verifiable HLA protocols (such the one employed in PANDA [10]) to be performed over erasure coded data so that detection and recovery of integrity violations can be employed to all the outsourced blocks in an efficient and scalable way. To do so, before outsourcing a file, clients: compute an erasure code for their data into several file shares using Reed-Solomon erasure codes; tag each block of the shares using public-verifiable homomorphic linear authenticators (based on private key of the client and a public verification metadata); and send them to the cloud with the corresponding shares.

Whenever a client wants to detect any integrity violation of a set of file-blocks, the public-verifiable HLA protocol is employed in each cloud (as described in PANDA). A challenge (composed of random numbers) is issued to the cloud for computing the proof-of-storage. Clouds then generate the homomorphic aggregator and a linear combination using the challenge issued and each file-block's: content; homomorphic authenticator; and the corresponding writer's public key. The homomorphic aggregator and linear combination are then sent to the client and, verified (by the client) that the homomorphic aggregator is valid for each file-block using the linear combination, the issued challenge, public metadata and the corresponding writer's public key.

When the client detects an integrity violation of the homomorphic aggregator, clients can recover the original state them by relying on the unaffected erasure codes. To do so, the paper proposes an erasure correction algorithm, where: data is retrieved from erasure coded blocks; reconstructed and erasure coded again along with the recomputed blocks; and send the affected shares to the cloud.

The approach proposed in this paper is very useful to strengthen the outsourced data against tampering and availability. This is due to the use of erasure codes on outsourced data and on the recovery of data. If data is partitioned into n shares, where m are necessary to reconstruct the data, data can withstand $n-m$ data losses or corruption and still reconstruct data. Thus by employing the proposed approach clients can detect whenever any their critical data's shares have been lost or corrupted and still reconstruct the affected blocks, so that no n

shares are affected with this problems. Furthermore, because public-verifiable schemes are employed clients can outsource the detection phase to an external third party, delegating the challenge and verification phase the this party and informing the client whenever integrity problems are detected, so that the client can proceed to the reconstruction phase.

This paper provides the same guarantees of HAIL (integrity layer for cloud storage for providing resilience to attacks targeting least frequently used files). The main difference is that instead of the private-verifiable HLA protocol used in HAIL, PANDA uses a public-verifiable HLA protocol. Thus allowing public auditing while preserving all the advantages of HAIL.

3.6 Discussion

Properties|System	Availability	CIWF Properties	SLA Proofs	Consistency	Access Control	Code in Cloud	Cloud Model
CloudProof [24]	No	IWF[1]	private attestations IWF	Total order	immediate and lazy	Yes	Single
Cumulus [26]	No	I	No	No	No	No	Single
POTSHARDS [25]	BFT	CI	Yes	No	-	No	Multiple
Secret Sharing made Simple [19]	-	C	-	-	±[4]	No	Multiple
Belisarius [23]	BFT	CI	No	No[2]	±[4]	No	-
DEPSKY [5]	BFT	CI	No	proportional consistency	±[4]	No	Multiple
Hybris [13]	BFT	CI	No	atomic[3]	-	No	-
HAIL [9]	BFT	I	private PoS I	-	±[4]	Yes	Single or Multiple
PANDA [10]	-	I	public PoS I	-	proxy re-signatures	Yes	Single
BlueSky [27]	-	CIWF	-	Serializability	-	Yes	-
Cryptree [15]	-	CI	No	-	lazy	No	-
SCFS[6]	BFT[5]	CI	No	atomic	±[4]	Yes	Single or Multiple
System Proposed	BFT	CI	public PoS I	atomic	±[4]	Yes	Multiple

[1]confidentiality needs to be provided by the clients by external means.
[2]requires total order
[3]when Zookeeper is employed as consistency anchor
[4]provided by each storage unit (individual server or cloud)
[5]provided by the cloud, BFT if used as default with DEPSKY.

Table 1. Comparison of the systems studied in the related work and the solution being proposed in this document.

In this section several distributed data systems and concepts related to the envisioned solution of a collaborative secure cloud-backed file system were presented and discussed. Regarding the concepts and systems presented, all provided valuable properties related with: 1. security: confidentiality, integrity, access control

and key distribution; 2. availability, i.e. tolerance against failures: transient, crash or byzantine; 3. file systems, regarding the usage of segments or blocks and the decision to aggregate files and store them into a big file or keep them separately stored; 4. consistency (principally: atomic and eventual consistency) and auditing (regarding proofs they provide).

The proposed solution was crafted by leveraging the knowledge acquired over the studied systems of the state-of-the-art. As final result this document proposes a public auditable collaborative secure cloud-backed file system with an easily mountable interface using POSIX semantics, while providing a strong consistency environment and where the data is safely stored, available and resilient to intrusions or eavesdrops.

Table 1 compares several studied systems and the solution proposed regarding the following assured properties:

- availability: byzantine fault tolerance; crash fault tolerant or transient fault tolerant.
- CIWF (confidentiality, integrity, write-serialization and freshness) properties.
- SLA profs (regarding the type of proofs and which CIWF can be proven).
- Consistency guarantees.
- Access Control.
- Cloud model, represented as S if it is a single or a M multiple cloud model.

The proposed file system was inspired by the SCFS and PANDA since the main motivation for the system is to implement a scalable file system for normal users who want to store their data safely on the cloud and also be able to share their data among other users. This is something that can be easily done in SCFS over a set of commercial clouds (such as Google Drive and Amazon) without requiring the user change its current CSPs. Other important remark regarding this system is that it uses a centralized coordination service (Zookeeper or DepSpace) as consistency anchor, thus allowing to collaboratively independently of the user location, something that cannot be easily performed in similar studied systems (such as Hybris or BlueSky) which were designed for working collaboratively same private network (using proxies or private clouds for coordinating the outsourced data operations to the cloud). Additionally, due to the fact that SCFS does not offer efficient mechanisms for ensuring integrity of the whole storage (to prove integrity of the whole storage clients need to request reads of every data block outsourced to the clouds and verify its signature) the proposed solution provides public verifiable proof-of-storage services based on the HLA protocol proposed in PANDA. With this the proposed solution will be publicly verifiable thus allowing clients to make use of third party auditors and improving trust concerns clients have regarding CSPs' actions on the outsourced data, benefiting with reports regarding their private data and reports about the overall reputation of the CSPs the client is using. This benefits are something that can only be done in public verifiable systems, where third party auditors can aggregate (without any confidentiality problems to their clients) the results of the proofs squired, something that can not be done in the private verifiable system (CloudProof and HAIL). Finally, regarding access control the proposed solution relies on the

access control implemented by the commercial clouds (approach also followed by SCFS). This is due to the fact that access control needs to be implemented by the cloud itself, something that solution does not control and leaves to the clients choosing (in order to support wider range of commercial clouds.)

4 Architecture of the Solution

The solution proposed in this section is to develop a DaaS cloud-backed file system for collaborative environments where data is shared among multiple clients, in order to address the goals mentioned in Section 2. As mentioned in the Section 1, the file system being proposed is an improvement of the SCFS [6] and DEPSKY [5], by adding a public-verifiable HLA protocol as described in PANDA [10] and data recovery mechanism over erasure codes (as described in [29]). These additions provide to the clients: guarantees that all their outsourced data is safely stored on the cloud and retrievable; while providing all the necessary mechanisms to detect and reconstruct the original data when unauthorized modification are performed to the client's data.

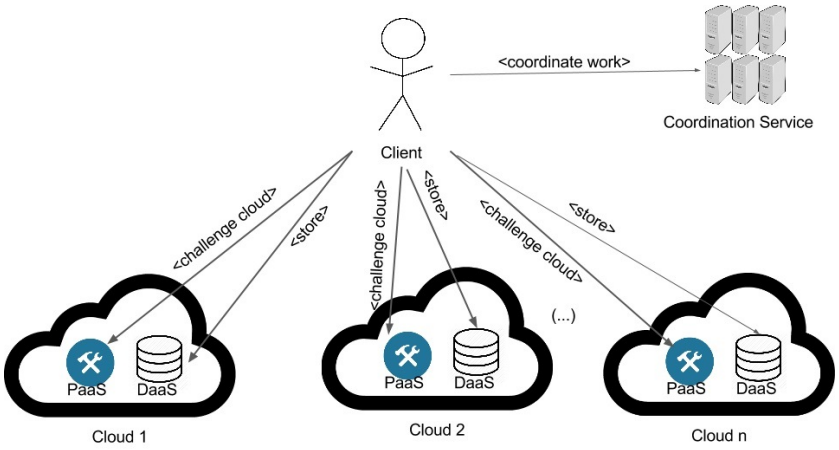

Figure 1. Entities involved in the proposed cloud-backed file system and their main functions to safely store and retrieve data on the clouds.

In this system three types of entities are involved (as can be seen in Figure 1), client, clouds and a coordination service, each performing different functions:

– Clients use the file system to outsource their data to the storage clouds, while issuing from time challenges to the cloud, for detecting if there where any unauthorized modifications to the outsourced data.

– Clouds are responsible storing clients' data in a DaaS storage unit. Also, they are also responsible for answering any challenge issued by the client (or a third party delegated by the client to perform such task), by executing a component of this solution in a PaaS for automating such task.
– Coordination Service for serving as consistency anchors and concurrency control between clients and their clouds. Providing an important role on all the coordination of shared data to allow stronger consistency models than the provided by the clouds.

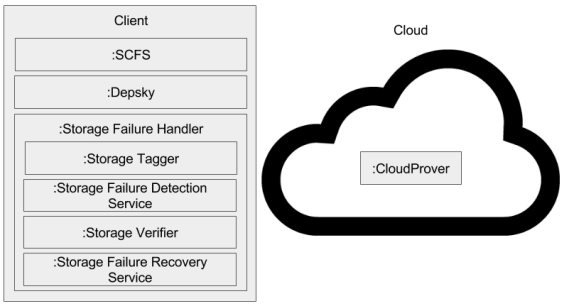

Figure 2. Cloud-backed file system component structure. The components that compose the solution are mapped on two entities: clients and clouds. For the clients, cloud-backed file system is installed in the SCFS and DEPSKY modules, alongside with the Storage Failure Handler library, for constructing HLA tags (Storage Tagger), issuing an HLA proof-of-storage challenge to the clouds (Storage Failure Detection Service) and recovering from storage failures (Storage Failure Recovery Service). For the clouds, Cloud Prover component is used in order to respond to the HLA proof-of-storage challenges issued to the cloud.

In more detail, the solution being proposed is structured into several components and can be seen in Figure 2. the file system uses four components:

– **Improved SCFS version** that bundles all the necessary storage components into an easy-to-use cloud-backed file system mounted on the client;
– **Storage Failure Handler (SFH)** that implements all the necessary libraries and services for the client to challenge the cloud and recover the affected data. This component is structured into:
  – Storage Tagger - Library that computes the tag used on the HLA protocol.
  – Storage Verifier - Library that verifies the proof issued by the cloud.
  – Storage Failure Detection Service - Service run at the client which automates the HLA protocol for issuing a request for a set of files (picked by the client).

– Storage Failure Recovery Service - Service run at the client which retrieves from all the client's clouds all the erasure coded shares and reconstructs the corrupted or lost shares.
– **Improved DEPSKY version** as storage back-end for providing cloud-of-clouds model and generating tags by using the Storage Tagger library of the SFH component.
– **CloudProver** designed to ran on a PaaS of each cloud delegated to store the user's data, and by communicating with DaaS cloud, allows clouds to respond to their clients a proof that the storage is safely stored.

System|Goal	1	2	3	4	5	6	7	8	9	10
SCFS	x						x	x		x
DEPSKY				x	x	x	x	x	x	
DepSpace		x	x				x	x		
SFH	x							x	x	x
Cloud Prover								x		

Table 2. Mapping of the goals proposed in Section 2 and the cloud-backed file system components that address those goals.

As will be detailed along the rest of this section, each component of the solution proposed is responsible for addressing some subset of the goals specified Section 2. Each subset is easily mapped to the systems in the Table 2 as follows.

(1) **Easy to use interface** - SCFS and SFH address this goal. SCFS provides an easily mountable interface for interacting with cloud-backed file system, through POSIX semantics on all operations. Thus, requires no user effort for storing and retrieving files. SFH provides two services that provide interface to the clients for obtaining proofs and recovery of files. Although SFH requires user intervention, it is designed to be easy to use by providing an interface to the client with simple semantics operations.
(2) **Atomic semantics to all operations** - DepSpace provides atomic consistency to all operations performed by the clients. Thus by using this entity with SCFS, data stored on eventually consistent clouds can be retrieved or stored with atomic consistency guarantees.
(3) **Concurrency control between clients** - DepSpace provides concurrency control, by using distributed lock-services, thus multiple concurrent operations to be performed on shared data.
(4) **Confidentiality and key distribution** - DEPSKY addresses this goal, by sending data to the clouds using secret sharing scheme with erasure codes, allowing data storage and key-distribution without any risk of eavesdrop of individual service providers or intrusions to less than $f + 1$ clouds, where f can be misbehaved or subjected to intrusions.
(5) **Data availability** - DEPSKY addresses this goal, by replicating data through several clouds using erasure codes and thus providing resilience

to byzantine, crash and transient cloud failures, if $3f + 1$ or more clouds are used and only f of those are subjected to such faults and byzantine fault tolerance to all metadata stored on this coordination service.

(6) **Integrity** - DEPSKY achieves this goal by using public key signatures on all offloaded data, and thus only allowing modification when the signatures provided are valid and from authorized users.

(7) **Minimize data store on clients** - This goal is addressed by the SCFS by storing all the needed data on DEPSKY and metadata on DepSpace.

(8) **Storage failure detection service** - This goal is provided by SFH by employing an efficient and scalable detection mechanism for data corruption or loss in the clouds (based on a generic public HLA challenge response protocol), Cloud Prover to allow clouds to respond to such challenges and the SCFS, DEPSKY and Zookeeper to store and propagate the necessary building blocks for the detection mechanism to work.

(9) **Recovery data service** - SFH assures this goal with the help of DEPSKY, by providing mechanisms for the client to retrieve data from unaffected clouds recover it and outsource to the clouds chosen by the user.

(10) **Scalability and latency resilience** - SCFS and SFH ensure this property. SCFS provides caching mechanisms for reading data in order to compensate latency and service congestion, while SFH allows the challenge-response detection protocol to be independent of the size of the file being verified, thus requiring constant overhead communication on the data detection mechanism.

In the rest of this section a detailed explanation of the solution will be provided as follows. 4.1 to 4.3 presents in detail the role and behavior of each component in the solution.

4.1 File system

The file system is provided to the clients by an improved version of the SCFS component alongside with an also version of DEPSKY. These components as described in Section 3, already provide a fully operational file system with POSIX semantics in order to be easily mounted on the client. Furthermore, using the coordination service provided by DepSpace as consistency anchors, SCFS and DEPSKY provide a collaborative environment with strong consistency, availability, integrity and confidentiality of all the outsourced data, and thus addressing goals 1-7 of the goals specified in Section 2. However in order for the solution to provide all the goals set in that section, some changes need to be performed for integrating the file system with the SFH and CloudProver components. Particularly, DEPSKY (and any affected operation of SCFS) will have to use the Storage Tagger component of SFH for tagging each outsourced file, so that proof-of-storage all the files stored in the cloud can be performed (using the Storage Failure Detection Service). It is expected that the only changes required are on the write operations of both DEPSKY and SCFS.

4.2 Storage Failure Handler

The Storage failure handler component is responsible for providing to the clients detection and recovery mechanisms for data loss or corruption on the outsourced clouds. Currently, there are no systems that provide both these mechanisms to the clients in order for the solution to rely upon. Therefore, the SFH module proposed will be implemented from the ground up and designed to be easily integrated with other systems. To do so, the detection mechanism being proposed will be based on the ideas presented about publicly homomorphic authenticators in [2,8,10,28] and will be provided by to the clients by Storage Failure Detection Service using the Storage Verifier component; while the recovery mechanism will be based on the recovery mechanism proposed as future work in the DEPSKY[5] paper and on the recovery mechanism presented in [20] and provided to the client by the the Storage Recovery Service presented in Section 4.2.2.

The remainder of this subsection will explain in more detail the role and behavior of each sub-component of the Storage Failure Handler.

4.2.1 Libraries

Storage Tagger: In order for the clients to obtain proof of storage using HLA protocols, clients need to tag all their outsourced data blocks using homomorphic authenticators. This component aims to facilitate this task in a library so it can be easily integrated with other components such as DEPSKY.

Storage Verifier: In order to verify cloud's proof of storage and detect whether outsourced data is lost or corrupted. clients need an easy and automatic way for performing this task. The Storage Verifier component allows clients to achieve this goal, by providing a library that can be integrated with other components such as the Storage Failure Detection Service.

4.2.2 Storage Failure Handler Services

In order to automate the process of detection and recovery provided by the solution, clients need alongside to their file system an interface to interact to the clouds and react according to it. The services Detection Storage Failure Service and Recovery Storage Failure Service provide the following interfaces:

Storage Failure Detection Service: this service allows clients to provide and challenge the cloud with a random challenge c regarding a set of file-blocks of a given client file, and receive the proof that the cloud is safely storing the file. The proof is then automatically verified by the Detection Storage Service, using the Storage Verifier component of the SFH and the result sent to the client.

Storage Failure Recovery Service: this service automates the process of recovering a file. The main objective of this is to guarantee that clients can always have their data redundantly stored even if data is corrupted from time to time. Thus, clients select which file and version to be recovered and trigger the Recovery Storage System, which will fetch the metadata needed from the coordination system, and obtain the file from the clouds $f + 1$ files. Then re-write the file to DEPSKY so that $3f + 1$ clouds can have the information safely stored.

4.3 Cloud Prover

Similarly to the storage failure handler, where clients need to have a tool that can provide automatic detection of outsourced data loss or corruption, clouds need an automated and standardized tool to allow effective proof of storage, without imposing to much overhead on clouds performance or increasing prohibitively the clients' operational costs. The Cloud Prover service was designed for this purpose.

This component should be easily integrated on the clouds to allow the clients to have guarantees that their storage is entirely retrievable. To do this, this module will automate the computation of the homomorphic aggregator and the linear combination required for the proof-of-storage, based on the received from the client challenge and the client's public key.

5 Evaluation

In order to evaluate the proposed solution, the evaluation phase will be composed of several tests, in order to validate the implementation of the clients' service (file system and detection and recovery failure services) and the clouds' service CloudProver. To do so, each service will be subjected to different test classes as will be explained throughout the remainder of this section.

Regarding the cloud-backed file system, the main goal of this phase will be to show how much additional storage space, performance and bandwidth overhead are imposed on the clients and clouds, these components are used. Specifically, since this file system is an adaptation of the SCFS systems, the Storage Tagger integration will at main focus to infer how much of this overhead is imposed on the regular SCFS; and, also, how much does it will imply in terms of client's computational power and costs increases on storage price.

Regarding the Storage Failure Detection Service, it will be tested regarding: the overhead imposed on the clients (by Storage Verifier) and on the clouds (by CloudProver) both in terms of performance and bandwidth; and also the precision and accuracy of this detection service, both in terms of false positives and false negatives. Moreover, price tests will be performed to the communication between this module and the CloudProver and to measure the cost it implies for each computation of proof-of-storage and how much it varies with the sample requested.

Regarding the Storage Failure Recovery Service test phase, the recovery service will be tested through performance, bandwidth and cost measurements. Thus hoping to achieve conclusive results related to how much cost would imply for the client to perform this type of approach and how would it affect the overall performance of the SCFS while this Service is being performed.

Finally, CloudProver will be at test to provide experimental results related on how much computational cost and computing time is required from the cloud to generate a valid proof-of-storage. Moreover, based on this experimental results, this test phase will provide a monetary price table on how much the size and and the number of files sampled affects the price raised on the PaaS costs.

6 Scheduling of Future Work

Future work is scheduled as follows:

1. **January 9 - March 29, 2016**: Design and implementation of the first deployable version of the proposed architecture:
   – January 9 - January 22: Design and implementation of Storage Tagger and Storage Verifier components of the Storage Failure Handler Library.
   – January 23 - February 5: Integrate Storage Tagger with SCFS and DEPSKY components for implementing the fully deployable file system service.
   – February 5 - March 18: Create the Storage Failure Detection Service and the Cloud Prover Service.
   – March 18 - March 29: Create the Storage Failure Recovery Service and integrate with DEPSKY
2. **February 12 - May 3:** Complete experimental evaluation of the results.
   – February 12 - March 18: Test cloud-backed file system.
   – March 19 - March 29: Test Storage Failure Detection Service and Cloud Prover Service.
   – March 29 - May 3: Test Storage Recovery Service.
3. **May 4 - May 30:** Write a paper describing the project.
4. **June 1 - June 29:** Finish writing the dissertation.
5. **June 30:** Deliver the MSc dissertation.

7 Conclusion and Outlook

This document proposes a cloud-backed file system that allows clients to work collaboratively and share files among each other, while providing security, available and strong consistent storage with little effort to the client. Furthermore, for clients that do not trust clouds with sensitive data, the solution employed allows clouds to prove to their clients that the outsourced data is safely stored, raising the level of trust between clouds and clients and moving one step forward to make cloud based solutions viable to critical system such as governments or hospitals, to have their data stored on the cloud. To make this possible, the solution proposed combines SCFS [6] (based on DEPSKY [5] and DepSpace[4]) with an homomorphic linear authentication protocol proposed by PANDA[10] and a recovery data corruption mechanism [29].

Along the document and particularly in the related work section (Section 3) an overview is presented about the several distributed systems related with the solution being proposed. This systems were analyzed regarding their ability of addressing problems related with: service availability (tolerance of the server faults); CIWF properties (confidentiality, integrity, write-serialization and freshness); ability to provide proofs that the service level agreements (SLA) established in contracts between clients and clouds are being respected by the cloud; the level of consistency provided; access control to the outsourced data; the ability to only run code on the clients and only use cloud's interfaces to store data (without running any code on clouds); and cloud model (single or multiple clouds). The

presented systems were summarized and compared with the document's proposed solution in Table 1.

The proposed solution was carefully presented along the document. The objectives of the proposed solution were presented in Section 2. A detailed explanation of the solution's architecture was presented in Section 4. Evaluation metrics were proposed in Section 5, for extracting concrete results on how the storage space and performance overheads affected both clouds and clients, when they interact with each other using the solution proposed. Finally, in Section 7 a work plan was scheduled and presented for building and deploying the first fully functional version of the solution performed, backed by a dissertation for presenting an in-depth discussion of the system developed.

The author hopes that this work, allows a clear understanding and awareness of the problems faced on the current clouds solutions, and hopes to contribute for the change needed in order to mitigate these problems and allow sensitive data to be more safely outsourced to the cloud.

Acknowledgments

I thank my advisors Professor Miguel Pupo Correia and Professor Miguel Filipe Leitão Pardal for all the help, effort and support employed during the development of this work. I thank all my colleagues of Safe Cloud project: André Joaquim; Diogo Raposo; and Karan Balu, for all the reviews and help on improving my work and solution. Finally, I thank IST and INESC-ID for giving me the opportunity and providing the resources needed for this work to be achieved.

References

1. Abu-Libdeh, H., Princehouse, L., Weatherspoon, H.: RACS: a case for cloud storage diversity. SoCC pp. 229–240
2. Ateniese, G., Kamara, S., Katz, J.: Proofs of storage from homomorphic identification protocols. Advances in Cryptology–ASIACRYPT 2009 pp. 319–333
3. Băsescu, C., Cachin, C., Eyal, I., Haas, R., Sorniotti, A., Vukolić, M., Zachevsky, I.: Robust data sharing with key-value stores. Proceedings of the International Conference on Dependable Systems and Networks pp. 1–12
4. Bessani, A., Correia, M., Fern, B.Q., Sousa, O.A.P.: P.: Depsky: dependable and secure storage in a cloud-of-clouds. in: Proc. of Eurosys (2011 (4)
5. Bessani, A., Correia, M., Quaresma, B., Sousa, P., André, F., Sousa, P.: DepSky. Proceedings of the sixth conference on Computer systems - EuroSys '11 (00), 31
6. Bessani, A., Mendes, R., Oliveira, T., Neves, N., Correia, M., Pasin, M., Verissimo, P.: SCFS: a shared cloud-backed file system. Proceedings of the 2014 USENIX conference on USENIX Annual Technical Conference pp. 169–180
7. Bessani, A., Alchieri, E.: DepSpace: a Byzantine fault-tolerant coordination service. ACM SIGOPS Operating ... pp. 163–176
8. Bondi, A.: Characteristics of Scalability and Their Impact on Performance. Proceedings of the 2nd international workshop on Software and performance pp. 195–203
9. Bowers, K.D., Juels, A., Oprea, A.: HAIL : A High-Availability and Integrity Layer for Cloud Storage. Ccs pp. 187–198

10. Boyang, W., Baochun, L., Hui, L.: Public auditing for shared data with efficient user revocation in the cloud. INFOCOM, 2013 Proceedings IEEE pp. 2904–2912
11. de Carvalho, N.T.F.: A practical validation of homomorphic message authentication schemes (2014)
12. Dobre, D., Karame, G.O., Li, W., Majuntke, M., Suri, N., Vukolić, M., Vukoli, M.: PoWerStore. Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security - CCS '13 pp. 285–297
13. Dobre, D., Viotti, P.: Hybris : Robust Hybrid Cloud Storage (2014)
14. Fiat, A., Naor, M.: Broadcast encryption. Proceedings of the 13th annual international cryptology conference on Advances in cryptology pp. 480–491
15. Grolimund, D., Meisser, L., Schmid, S., Wattenhofer, R.: Cryptree: A Folder Tree Structure for Cryptographic File Systems. 2006 25th IEEE Symposium on Reliable Distributed Systems SRDS06 (Section 7), 189–198
16. Hassan, Q.: Demystifying cloud computing. The Journal of Defense Software Engineering pp. 16–21 (2011)
17. Hunt, P., Konar, M., Junqueira, F., Reed, B.: ZooKeeper: Wait-free Coordination for Internet-scale Systems. USENIX Annual Technical ... pp. 11–11
18. Jong, J.J., Park, H., Kim, C.: Lecture Notes in Electrical Engineering 224
19. Krawczyk, H.: Secret sharing made short. Advances in Cryptology - CRYPTO´ 93 pp. 136–146
20. Kui Ren, C.W.Q.W., Lou, W.: Ensuring Data storage Security in cloud computing. Dept of ECE, Illinois Inst. Of Technol., Chicago, Il, USA pp. 1–9 (2009)
21. Merkle, R.C.: Secrecy, authentication, and public key systems. (1979)
22. Netto, H.V., Lung, L.C., Souza, R.L.D.: Tolerância a Faltas e Intrusões para Sistemas de Armazenamento de Dados em Nuvens Computa- cionais pp. 2–48 (2014)
23. Padilha, R., Pedone, F.: Belisarius: BFT storage with confidentiality. Proceedings - 2011 IEEE International Symposium on Network Computing and Applications, NCA 2011 pp. 9–16 (2011)
24. Popa, R., Lorch, J., Molnar, D.: Enabling security in cloud storage SLAs with CloudProof. Proc. USENIX ... pp. 355–368
25. Storer, M.W., Greenan, K.M., Miller, E.L., Voruganti, K.: POTSHARDS: Secure Long-Term Storage Without Encryption. Atc pp. 143–156
26. Vrable, M., Savage, S., Voelker, G.M.: Cumulus: Filesystem backup to the cloud. ACM Transactions on Storage pp. 14:1–14:28
27. Vrable, M., Savage, S., Voelker, G.G.: Bluesky: A cloud-backed file system for the enterprise. Proceedings of the 10th USENIX conference on File and Storage Technologies p. 19
28. Wang, C., Ren, K., Lou, W., Li, J.: Toward publicly auditable secure cloud data storage services. IEEE Network 24(4), 19–24 (2010)
29. Wang, C.W.C., Wang, Q.W.Q., Ren, K.R.K., Lou, W.L.W.: Ensuring data storage security in Cloud Computing. 2009 17th International Workshop on Quality of Service pp. 1–9 (2009)
30. Waters, I.B.: Lecture 12 : Broadcast Encryption pp. 1–4 (2009)