D3.1
Architecture for Data Management

Project number:	643964
Project acronym:	**SUPERCLOUD**
Project title:	User-centric management of security and dependability in clouds of clouds
Project Start Date:	1st February, 2015
Duration:	36 months
Programme:	H2020-ICT-2014-1
Deliverable Type:	Report
Reference Number:	ICT-643964-D3.1/ 1.0
Work Package:	WP 3
Due Date:	Oct 2015 - M09
Actual Submission Date:	2nd November, 2015
Responsible Organisation:	IBM
Editor:	Marko Vukolić
Dissemination Level:	PU
Revision:	1.0
Abstract:	In this document we present the preliminary architecture of the SUPERCLOUD data management and storage. We define the design requirements of the architecture, motivated by use cases and then review the state-of-the-art. We then present designs for the overall unifying architecture for data management as well as novel security and dependability data management features.
Keywords:	architecture, cloud-of-clouds, data management, dependability, security, storage

This project has received funding from the European Unions Horizon 2020 research and innovation programme under grant agreement No 643964.

This work was supported (in part) by the Swiss State Secretariat for Education, Research and Innovation (SERI) under contract number 15.0091.

Editor

Marko Vukolić (IBM)

Contributors (ordered according to beneficiary numbers)

Mario Münzer, Benjamin Walterscheid (TEC)
Sébastien Canard, Marie Paindavoine (ORANGE)
Alysson Bessani (FFCUL)
Caroline Fontaine (IMT)
Krzysztof Oborzyński (PHHC)
Meilof Veeningen (PEN)
Paulo Sousa (MAXDATA)

Disclaimer

Executive Summary

In this document we present the preliminary architecture of the SUPERCLOUD data management and storage. We start by defining the design requirements of the architecture, motivated by use cases and then review the state-of-the-art. We survey security and dependability technologies and discuss designs for the overall unifying architecture for data management that serves as an umbrella for different security and dependability data management features. Specifically the document lays out the architecture for data privacy, secure sharing schemes, geo-replication and fault and disaster tolerance for SUPERCLOUD data management. We further discuss our federated identity and key management approach.

Contents

List of Figures

Chapter 1 Introduction

In WP3, we will design and implement SUPERCLOUD protection of user data and assets in the distributed cloud, focusing on autonomic and end-to-end security as well as dependability, but also performance and cost. This WP provides a common user experience of data protection across multiple underlying clouds through modular and on-demand data security services such as blind computation, integrity and verifiability, and data availability.
This deliverable presents the preliminary architecture of SUPERCLOUD data management. The state-of-the-art in data management in geo-replicated, cloud-of-cloud context (see Fig. 1.1) does not adequately address users' requirements. For example, SUPERCLOUD specific use cases stemming from healthcare ask for data management solutions that either do not exists or are not practically proven today or sometimes both.

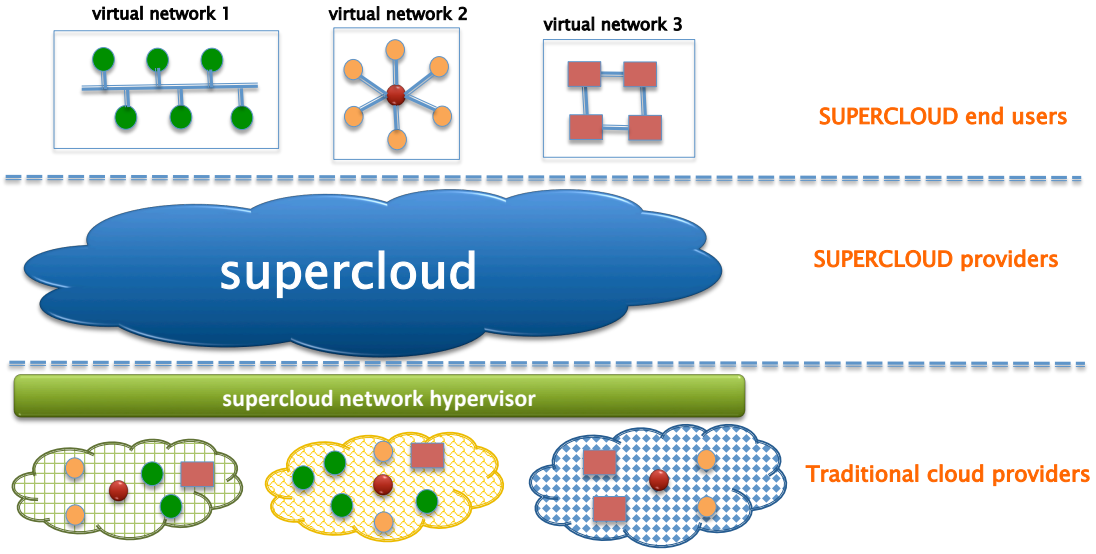


Figure 1.1: SUPERCLOUD cloud-of-cloud context.

In a nutshell, existing cloud data management solutions glaringly lack efficient *privacy* mechanisms and have limited support for *secure data sharing*. Improving the security of data management in these two aspects is a specific focus in SUPERCLOUD. When it comes to dependability, each cloud provider offers specific geo-replication solutions, and unifying these into a common infrastructure [171] is not obvious. SUPERCLOUD aims at rectifying this and offers a flexible easily extensible data management architecture tailored for the cloud-of-cloud context.
Regarding for example dependability, the main problem of existing solutions is that they often take single-cluster (datacenter) solutions and port them to the geo-replicated context of cloud-of-clouds. This is specifically true with replication protocols such as popular Paxos [116] and RAFT [142] protocols, as well as their BFT counterparts [59] that are not tailored to geo-replication. Exceptions to this rule exist (and we overview those later in this deliverable in Chapter 3) but they are both rare

and non unified in a single architecture. The goal of this deliverable is to propose an architecture to rectify this and that would feature security and dependability as first class citizens.

Structure. This deliverable is organized as follows.

- Chapter 2 overviews design requirements for the SUPERCLOUD Data Management Architecture. The Chapter starts with two specific use case requirements we consider in SUPERCLOUD, the Healthcare Laboratory Information System by MAXDATA and the Medical Imaging Platform by Philips Healthcare. Then it proceeds to define more general requirements that SUPERCLOUD Data Architecture should satisfy. These include both functional and non-functional requirements, including those that pertain to security, dependability and compliance.
- Chapter 3 gives an overview of state of the art in security and dependability of distributed data management systems with strong focus on cloud-of-clouds.
- Chapter 4 gives the overview and a high-level logical architecture of the SUPERCLOUD Data Management Architecture. The Chapter defines the main SUPERCLOUD storage entities and lists and discusses security and dependability features that SUPERCLOUD Data Management shall offer. The Chapter also introduces the SUPERCLOUD approach to identity and key management.
- Chapters 5 and 6 offer details of the architectures of individual security and, respectively, dependability features while mapping them to the common architecture defined previously in Chapter 4. Focal security features consist of novel schemes for users' data privacy and security of sharing (including secure deduplication). Focal dependability features have to do with novel ways of replicating state machines across clouds as well as facilitating data access across separated geographical locations.

Deviation from Workplan. This deliverable conforms to the DoA/Annex 1, Part B.

Target Audience. This deliverable aims at IT professionals in general, including researchers and developers of security and management systems for cloud-computing platforms. The deliverable assumes undergraduate knowledge in computer science technology.

Relation to Other Deliverables. D3.1., together with its Compute and Network Architecture counterparts (D2.1 and D4.1, respectively) constitutes the basis of the SUPERCLOUD architecture and, as such, informs D1.1, which presents the overall architecture.

Chapter 2 Design requirements

We develop a specification of the design requirements of SUPERCLOUD data management architecture starting from two SUPERCLOUD use cases: MAXDATA's Healthcare Laboratory Information System and the Medical Imaging Platform by PHILIPS HEALTHCARE. We then generalize these requirements to cover some potential additional use cases more broadly.
In a nutshell, our storage requirements of our targeted use cases can be grouped into several categories:

- Functional requirements (e.g., APIs, storage volumes, sharing capabilities);
- Dependability requirements (e.g., number of nines of availability, ability to tolerate untrusted cloud resources, data integrity requirements);
- Security requirements (e.g., privacy enabling features, identity and key management requirements);
- Compliance requirements (e.g., data location awareness and control, data protection);
- Other non-functional requirements (e.g., performance, latency, throughput, cost, monitoring).

These requirements, together with the traits of the cloud storage infrastructure we will be building upon, drive the SUPERCLOUD data management architecture decisions later presented in Chapter 4. In the rest of this Chapter we first introduce and justify data management design requirements from the perspectives of MAXDATA (Sec. 2.1) and PHILIPS HEALTHCARE (Sec. 2.2) use cases. Then, in Sec. 2.3 we summarize and overview more generically the set of requirements that drives the SUPERCLOUD data management layer architecture.

2.1 Healthcare Laboratory Information System

The healthcare laboratory information system, henceforth mentioned as CLINIdATA®LIS, is a cross-platform web application where server components may run on any common operating system (e.g., Linux, Mac OS X, Solaris, Windows) and relational database (e.g., MySQL, PostgreSQL, Oracle, SQL Server). This system needs to integrate with dozens of other clinical and non-clinical information systems (e.g., ICU, patient identification, billing, regional health portals) and includes a set of real-time interfaces with physical electronic equipments, namely automated analysers.
CLINIdATA®LIS stores medical data along with other personal data, so the SUPERCLOUD storage infrastructure should comply with Directive 95/46/EC and the soon to come General Data Protection Regulation (GDPR). This implies requirements for isolation between tenants, and the following requirements:

- **Location-awareness.** Users should be aware of the physical locations where data is stored.
- **Location-control.** Users should be able to define the set of possible physical locations, at country level, where users' data may be stored.

CLINIdATA®LIS is used by different types of healthcare organizations, ranging from small laboratories with a few dozens of professionals and hundreds of transactions per day, to very large hospital

clusters with thousands of professionals and tens of millions of transactions per day. Hence, these organizations should be able to control how their resources are protected using SLAs including for instance agreed levels of availability, redundancy, backup, disaster recovery, etc. This implies the following requirement:

- **Control over Architecture and Data.** The user should be able to monitor actively its allocated resources, while enabling a high level of customizability of the storage architecture and its services.

Especially in hospitals, CLINIdATA®LIS is a critical application that needs to be constantly available in order to ensure non-stop operation of various departments including the ER (emergency room). Therefore, although concrete availability in each deployment should comply with the agreed SLA, the SUPERCLOUD infrastructure should be able to offer high availability whenever necessary:

- **High Availability.** Storage should be able to have high availability, up to 99.999%.

As mentioned before, CLINIdATA®LIS includes a set of real-time interfaces with physical electronic equipments, namely automated analysers. These interfaces are used mostly to send commands to analysers and to receive exam results. Both communication flows have real-time requirements and need to get data from storage. This implies the following requirement:

- **Real-Time.** Storage access should be able to offer real-time guarantees, namely predictable and bounded storage access time.

2.2 Medical Imaging Platform

There are three main Philips Healthcare use-cases that aim at deployment into SUPERCLOUD infrastructure, namely:

- Cloud data storage and disaster recovery use case
- Cloud data storage and processing use case
- Distributed cloud data storage and processing use case

2.2.1 Cloud data storage and disaster recovery use case

The cloud data storage and disaster recovery use-case focuses on helping a hospital to ease management of the patient data stored in the hospital archive (see Figure 2.1). Current hospital archives are on-premise solutions that needs to handle all the clinical data, especially imaging studies which can be as large as 1GB. Therefore, it would be attractive to offload this data into cloud, such that storage size on-premise archive can be limited. For example, the data from the last 6 months could be stored on premise, whilst cloud stores for longer periods (e.g., 10+ years). In this use-case the cloud becomes an extension of the hospital archive. The core value of this solution is to ensure that patient data is not lost as a result the cloud storage is also a disaster recovery solution for the hospital. The workflow of data extends to the cloud but performance is not a primary concern here, as patient data is always prefetched from the cloud to the on-promise hospital archive prior to the scheduled examination.
From data management perspective, the use-case requires:

- Privacy of medical data (it is top priority)
  - Storage must be robust against any security breaches.
  - Data must not be interpretable in transit and storage, covering disk, file system and database layers.

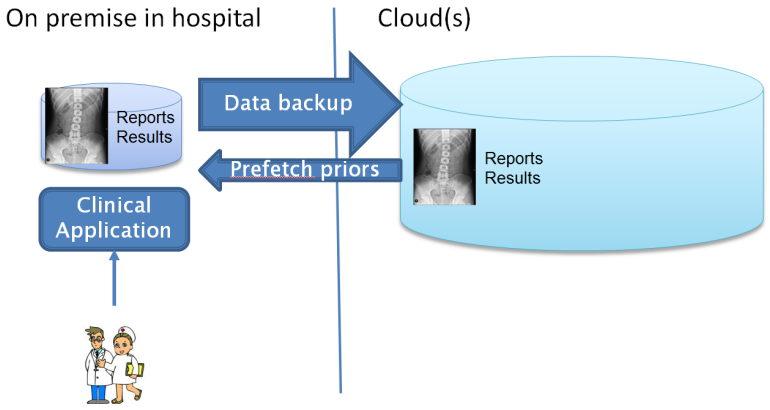


Figure 2.1: Cloud data storage and disaster recovery use case.

  - Data may be de-identified, as long as data can still be fetched.
  - Role Based Access Control (on operation level).
  - Detailed audit trails and activity monitoring.

- Medical data characteristics
  - Volume: High volume of data needs to be stored, in the order of 100+ tera-bytes, and kept at least for 10+ years.
  - Format: DICOM [134] and non-DICOM.
  - Organization: files, DBs.

- Correctness of stored data
  - Data may not get tampered and must be complete and correct.

- Medical data may not cross certain legal country boundaries
  - Guaranteed cloud storage within certain countries.
  - Distributed storage, across multiple cloud and countries, in a privacy compliant manner; e.g. via a secure encrypted storage where data cannot be interpreted.

2.2.2 Cloud data storage and processing use case

The cloud data storage and processing use-case focuses on easier access of the medical personnel to the medical data processing applications (see Figure 2.2). The access to the applications can be possible then not only from the hospital laboratory where the equipment is installed but also from PCs in the hospital/home or secured mobile devices. This way the doctors collaboration can improve as well due to easier availability of data. The main processing of data is related to image analysis to help doctors in correct diagnosis. The image analysis is done by applying various algorithms ranging from simple image enhancing ones to detail analysis that focuses on finding potential anatomical anomalies, such as aneurysms, tumors, etc. Since processing of the data is separated into the cloud then performance and latency become critical, as imaging results and user interaction must be streamed semi real-time to the clinical user. This use case involves only a single hospital organization.
From a data management perspective, the use-case requires:

- The same storage demands as in Cloud data storage and disaster recovery use case
- Storage and processing isolation per hospital group

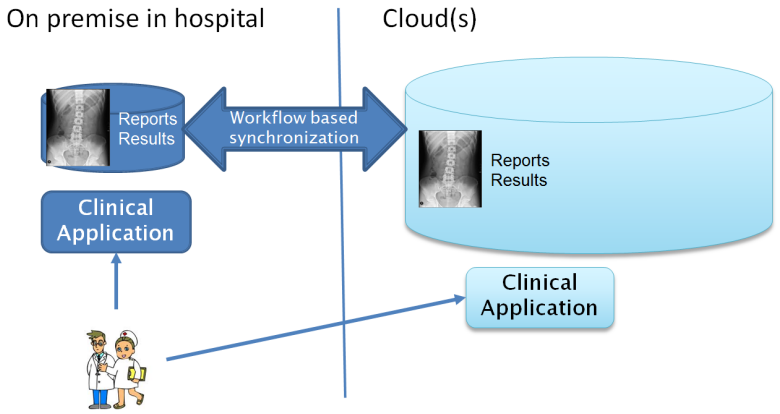


Figure 2.2: Cloud data storage and processing use case.

  - Special permissions are required to access data outside own hospital context (role-based-access control).

- Performance
  - Low latency of accessing data in the cloud by the clinical applications to ensure quick availability of the processed results.

2.2.3 Distributed cloud data storage and processing use case

The distributed cloud data storage and processing use case focuses on easier access of the medical personnel to the medical data across hospitals, i.e., this use-case ensures a patient centric view to the healthcare professional by showing all data of the patient; so not only data managed by the local hospital, but also data managed by other hospitals (see Figure 2.3). This would enable providing the complete longitudinal patient record. This use case has the highest complexity, as it involves multiple hospital organizations managing patient data. Performance and latency are critical, as imaging results and user interaction must be streamed semi real-time to the clinical user. The user may view mashup of data from multiple clouds and hospitals, or search for comparable reference studies (across the clouds), to assist in diagnosis of the treated patient. Advanced processing may even include comparing large study data, across clouds. As a result, the identity management of the clinical user is critical in this use case, as it is accessed from multiple organizations.

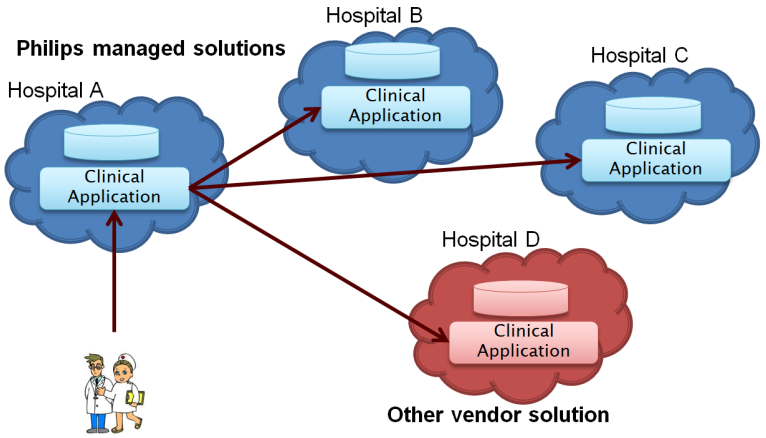


Figure 2.3: Distributed cloud data storage and processing use case.

From a data management perspective, the use-case requires:

- The same storage and processing demands as in Cloud data storage and processing use case
- Medical data characteristics
  - High volume of data needs to be accessed, in order of peta-bytes.
  - Format: DICOM, non-DICOM, unstructured.
  - Organization: files, DBs, raw.
- Privacy
  - Identity management across clouds of healthcare professionals.
  - Auditing across clouds of accessed patient data.
  - Privacy of stored data, whilst access and queries are still supported.
- Interoperability across clouds
  - Supporting Philips managed cloud solutions and 3rd party vendor solutions.
- Performance
  - Search performance and privileges across clouds.

2.2.4 Detailed requirements summary

Interfaces and functional requirements:

Data format: any (DICOM, non-DICOM, unstructured).
Data organization: files, databases, raw.
Data location: data has to be guaranteed to stay in prescribed legal (country) boundaries.
Distributed storage: storing data across multiple cloud and countries has to be done in privacy compliant manner.
Data interoperability: queries of ANY data over available clouds shall be possible (while complying with privacy rules).
Data storage: supporting Philips managed cloud solutions and 3rd party vendor solutions.

Dependability requirements:

Fault tolerance: Data may not get tampered and must be complete and correct.
Data availability: 99.99%.

Security requirements:

Storage security: Storage must be robust against any security breaches covering disk, file system and database layers.
Data security: data must not be interpretable in transit and in rest.
Data access: Special permissions are required to access data outside own hospital context (role-based-access control).
Data access monitoring: Detailed audit trails and activity monitoring shall be available for accessed data in any location.

Scalability and performance:

Data volume storage: High volume of data needs to be stored, in order of 100+ tera-bytes.
Data volume processing: High volume of data needs to be accessed, in order of peta-bytes.
Data lifespan: 10+ years.

Data access: Low latency of accessing data in the cloud by the clinical applications to ensure quick availability of the processed results.

Efficient data retrieval: search performance and privileges across clouds if efficient.

Key management:

Data identification: Identity management across clouds of healthcare professionals.

2.3 Generalized Requirements

Starting from the requirements of the above use cases, we generalize them to derive overall requirements for the SUPERCLOUD data management layer.

2.3.1 Functional requirements

SUPERCLOUD data and storage management should support the APIs traditionally offered by commodity cloud providers to facilitate migration of existing applications to SUPERCLOUD. Therefore, SUPERCLOUD data management is required to support the following storage interfaces:

- Object store interface, for storing files and objects in a key value store;
- Block store interface, for implementing storage volumes, backing SUPERCLOUD virtual machines;
- File system interface;
- The data management abstractions should cater for efficient, dependable and secure execution of database engines, although SUPERCLOUD will not be required to natively offer SQL database interfaces.

Furthermore, the sizes of allocated storage should be virtually unlimited as the practically unlimited storage from commodity cloud providers may be used to this end. SUPERCLOUD data management should be capable of handling petabytes of storage, supporting various data formats.
Secure sharing capabilities and seamless VM migration across clouds are a requirement. SUPERCLOUD shall remain flexible supporting different user-specific storage and data management policies.

2.3.2 Dependability requirements

The SUPERCLOUD data management layer should accommodate the availability requirements stipulated by the users. As a rule of the thumb, we expect 99.99% (four nines) and 99.999% (five nines) of availability to be commonplace.
There is no clear fault model that applies to all SUPERCLOUD deployments, so the SUPERCLOUD architecture should accommodate for both crash model of underlying cloud resources (e.g., L0 cloud services can be unavailable), all the way to Byzantine fault model (where L0 cloud services are untrusted), covering arbitrary faults and intrusions. The network should not be assumed to be synchronous and mechanisms for dealing with network asynchrony and partitions should be put in place. Models exploring the design space of correlation between L0 cloud component faults and network faults will be useful.
Consistency and data integrity are of primary concern in SUPERCLOUD. SUPERCLOUD targets sensitive applications that require consistency guarantees stronger than what can be found in typical cloud offerings (e.g., eventual consistency [169]). Therefore, the primary consistency criteria for SUPERCLOUD will be *linearizability* [98], that gives an illusion of traditional, sequential access to data. However, SUPERCLOUD data layers should remain flexible enough to support weaker consistency models, in response to the explicit demand of a user/administrator, to boost performance. Therefore, SUPERCLOUD should support tunable consistency semantics.

2.3.3 Security requirements

Data privacy is of primary concern for SUPERCLOUD. SUPERCLOUD should enable and implement several redundant and complementary privacy mechanisms so the end user can choose from such privacy offering the mechanism that best suits its needs. Among other techniques, the data management layer will put in place isolation techniques to ensure that data of different tenants cannot be accessible (unless sharing capabilities are explicitly enabled). Privacy of data shall be protected for both data in transit and at rest. Sensitive data shall be amenable to anonymization.
SUPERCLOUD should put in place mechanisms to support auditability of storage including tracking histories of accessed patient data. To this end, SUPERCLOUD shall support role based access control policies.
Performance and cost optimizations, such as storage deduplication, shall not break security (security comes first principle). Where explicitly requested and acknowledged by the user, such performance and cost optimizations may gracefully degrade security guarantees, while providing clear indications of a tradeoff to the user.
Federated identity management across clouds should be supported. Secure key management solutions based on proven industry technologies (e.g., Openstack Barbican, Key Management Interoperability Protocol (KMIP)) should be tailored to the multi-cloud use cases relevant to SUPERCLOUD.

2.3.4 Compliance requirements

SUPERCLOUD data management shall implement mechanisms for compliance with data location awareness and control as well as data protection. The user shall be able to control which cloud resources are used to store its data, taking into account their location. The data management architecture should remain flexible enough and designed with future-proofing in mind to accommodate directives and laws that may impose further compliance restrictions.
Laws and directives of the European Union as well as member and associated countries (incl. Switzerland) shall receive priority in fulfilling compliance requirements.

2.3.5 Other non-functional requirements

SUPERCLOUD data management shall attempt at delivering the best performance to the users while satisfying constraints of other requirements outlined above. In particular, our solution shall deliver solutions with the goal of minimizing latency and maximizing throughput.
The cost of SUPERCLOUD storage solutions will often be at odds with other requirements. The SUPERCLOUD data management layer shall allow flexibility to the users in terms of tradeoffs between cost and other guarantees.

Chapter 3 A short survey of secure and dependable cloud data management systems

In this chapter, we survey the related work on secure and dependable cloud data management systems. Specifically, in Section 3.1 we first overview privacy enabling mechanisms. In Section 3.2 we survey mechanisms for secure sharing and data deduplication. In Section 3.3 we overview integrity protection mechanisms in the context of a single untrusted cloud. Then, our focus turns to solutions based on multiple clouds. Namely, in Section 3.4 we overview techniques for the geo-replication of state machine to ensure dependability for critical storage metadata. In Section 3.5 we discuss techniques that manage data in hybrid cloud and multi-cloud contexts. We conclude with Section 3.6, which highlights the key techniques and proposals that materialize in practice the secure and dependable data management into real systems.

3.1 Privacy enabling mechanisms for untrusted clouds

In this section, we review the way to introduce privacy protecting mechanisms inside a cloud storage system. We focus on techniques that permit the manipulation of individual's personal data in a privacy protecting manner. Within SUPERCLOUD, we will have two different interlaced approaches: one based on cryptography, one based on anonymization techniques. The idea behind is either to encrypt the data so that only authorized entities can access it, while permitting some blind computation over these data, or to de-identify the data so that it is no more possible to make the link between the data and the individual behind.

3.1.1 Searchable encryption

Searchable encryption is a special form of data encryption which permits to perform some search over protected data. As a result, either the searched data is there, and the information is given, but the rest of the data is still protected, or the searched data is not there, and no information is given on the true underlying plain data.
Searchable Symmetric Encryption (SSE) was introduced by Song, Wagner and Perrig in 2000 [157]. A client encrypts her database using a dedicated algorithm before sending it to an external server. Then, the client is able to perform a search over the encrypted data and obtains only the encrypted result of her search. More formally, the syntax of such a scheme can be described as follows. A database DB is a set of n files. Each file is identified with an index ind_i and contains a set of keywords W_i. We denote $W = \cup_i W_i$. A query is a Boolean formula $\phi(\overline{w})$ with $\overline{w} \in W^m$, m being the formula's number of arguments. A SSE scheme consists in two protocols, $\mathsf{EDBSetup}$ and Search. $\mathsf{EDBSetup}$ takes a database DB as input and outputs a secret key sk and an encrypted database EDB. Search is an interactive protocol between a client and server. The client takes as inputs a query $\phi(\overline{w})$ and a secret key sk and the server takes as input an encrypted database EDB. The client then sends an encrypted version of her query to the server. The server responses with an encrypted set of indices, which is decrypted by the client. The protocol is correct if the set of indices output by the client refers to keywords which satisfy the formula $\phi(\overline{w})$.
Efficient schemes were designed for *single-keyword search*, such as [157, 87, 61, 71, 64, 106]. In these schemes, complexity is scaled with regard to the size of the returned data set (independently of the

database's size).
For conjunctives queries, dedicated solutions propose a complexity that increases linearly with the size of the database [89, 31, 50].
Cash et al. [58] achieved a sub-linear complexity for conjunctive queries, as well as for more general Boolean queries, and unlike precedent schemes, their scheme also allows free-text search. They assessed the efficiency of their scheme with experimental results over medium sized and large sized databases, containing millions of documents and indexing all the words, thus allowing free text search. However, they achieve such an efficiency by allowing some leakage on access patterns (such as the intersection between two sets of results) and on queries (such as the repetition of queries).
Searchable encryption has also been investigated in a public key setting, yielding Public Key Encryption with Keyword Search (PEKS). In this setting, Alice provides a trapdoor to search for keywords in data encrypted under her public key. It was introduced by Boneh et al. [46] and further investigated in [173, 19, 34, 156]. The case of conjunctive queries was investigated in [47]. However, public key settings remains less efficient than symmetric key setting.

3.1.2 Secure multi-party computation

Multi-party computation (MPC) – also known as secure multi-party computation (SMC) – allows multiple participants/parties to compute a function over a given input. As a result of the participation on this computation, each party holds the common output and no party knows anything else other than the own result of the function.

3.1.2.1 Multi-party computation

Multi-party computation protocols provide outsourced computation without leaking sensitive inputs to the party performing the computation. Achieving privacy is hard when outsourcing to a single computation party, but feasible if the computation is distributed between several parties. Indeed, having a single computation party performing arbitrary computations on encryptions requires fully homomorphic encryption, a cryptographic primitive that is still impractical for realistic applications [78]. But distributing computations between multiple computation parties in a privacy friendly way is possible, and getting more and more practical, using multi-party computation protocols (e.g., [44, 73]). Such protocols typically distribute the values that are computed between multiple parties.
Interactive protocols exist to perform operations on these shared values like multiplication, comparison, bit decomposition, and others. Guarantees on the privacy of the inputs and correctness of the results depend on the protocol at hand; e.g., [44] proposes to protect against one eavesdropping computation party whereas [73] protect against $n-1$ computation parties that may actively manipulate the result. However, in general, such protocols guarantee privacy and correctness only if at least one of the computation parties is honest.
Practically speaking, secure multi-party computation addresses the setting where a user wants to perform a query over data held at different locations. In general, this data can come from any number of data sources. For instance, consider the query "can I have the average age of all people with sleeping problem", where the answer should be aggregated from hospitals from different countries. Note that often, it is not possible to collect this data in one place and compute the query result as a local computation – for instance, due to locality regulations (as discussed in the requirements). Some queries, such as the sum of certain values, may be computed directly using a "local computation + aggregation by client" paradigm without significant privacy impact. For other queries, such as the median of a set of values, it is however not obvious that this can be done. In these cases, multi-party computation still allows queries to be answered without affecting privacy, by computing query answers without the need for collecting information at one place. More generally, such settings fall under the umbrella term "privacy preserving data mining" (PPDM). In real life applications, more precisely in the region of business and research such as medical or financial databases, privacy preservation is an important demand. The main aspect of PPDM is to facilitate the public usage of private data (e.g.,

in order to use data for statistical investigations), while still preserve the privacy. These types of data access are negligible if "meaningful information" or "knowledge" cannot be extracted.

3.1.2.2 Secret sharing

A (n, t) secret sharing scheme [154] allows a secret s to be shared among $n > t > 1$ parties in such a way that (1) no party has access to the original secret s and (1) at least $t + 1$ correct parties have to cooperate for recovering the s. This is implemented by two primitives: *share*, which generates n shares of the secret to be distributed among the parties; and *combine*, which combines $t + 1$ of these shares to recover the secret. Shamir's secret sharing scheme [154] is based on polynomials over a finite field, in which *share* is implemented by generating a polynomial $y = f(x)$ of degree t such that $f(0) = s$. Using this polynomial we generate the n secrets, which are pairs (x, y). The secret can be recovered by combining (i.e., *combine*) $t+1$ secrets by using Lagrange interpolating to generate the polynomial, recovering $f(0)$ [70].

3.1.2.3 Combining multi-party computation and verifiable computation

Verifiable computation allows proving correctness of a computation without any additional information leakage. The goal of verifiable computation is to allow a client to outsource a computation to a worker and cryptographically verify the result with less effort than performing the computation itself. Based on recent ground-breaking ideas [93, 85], Pinocchio [146] was the first implemented system to achieve this for some realistic computations. Recent works have improved the state-of-the-art in verifiable computation, e.g., by considering better ways to specify computations [37], or adding access control [22]. Moreover, [30] have shown how to perform verifiable computation on authenticated data; that is, it allows outsourcing queries on a dataset to an untrusted third party, where the correctness of the query result can be efficiently verified with respect to some small authentication information of the dataset.
One interesting avenue of research is to combine the privacy guarantees of multi-party computation and the correctness guarantees of verifiable computation. An initial proposal, with still long proof verification time, was done in [153]. In [74], performance improvements for the particular application of performing linear optimization were improved, with experiments running on a single PC showing encouraging results. In theory, it should be possible to produce Pinocchio-like proofs in a multi-party fashion. However, the practical performance of this approach, and also of the techniques of [74] when the different workers are geographically separated, remain to be studied.

3.1.3 Oblivious storage

Oblivious RAMs (ORAMs) [88] enable the storage of outsourced data while ensuring the access pattern of the data is not leaked. ORAM complements encryption, that hide the contents of files, but not the order or frequency of accesses. This is useful in many situations in which, apart from file contents, these access patterns are sensitive. For instance, in the setting of medical research on DNA records, access patterns reveal the requester's points of interest [108]; also in the context of encrypted e-mail, it has been shown that access patterns may reveal up to 80% of search queries on an encrypted e-mail database [101].
The ORAM cryptographic techniques hide access patterns from the storage system by touching multiple files on each access in such a way to completely hide which file is actually being accessed. The recent "*Path* ORAM" [161] is particularly efficient, with optimizations for secure storage use cases (e.g., [125]). However, while ORAMs hide the access pattern to the server, existing systems do this at the expense of partly revealing it to other users of the system, which can be equally undesirable. The only current architecture in which access patterns are also hidden from other users, employs several additional "anonymizers" that have to perform a lot of work for each access (in particular, the files touched by an ORAM access have to pass through all anonymizers), and at least one of them needs to be trusted [102].

3.1.4 Anonymization

Data opening provides clear benefits for society, individuals and organizations. It can for example be used in transports, for web services and for tourism to characterize the movements, for hearing measures. To the extreme, one can imagine to deploy an Open Data on all these data. But these data contain user personal data, so the European data protection law applies. The direct consequence is that each treatment must be declared to the designated authority, such as the CNIL in France. In the case of Open Data, this can be a prohibitive point since any kind of treatment can be executed by any party. But once a dataset is truly anonymized, and individuals are no longer identifiable, European data protection law no longer applies.
The European Directive 95/46/EC, Recital 26 gives a conceptual definition of data anonymization. It explains that a data is said to be anonymized if the data subject can no longer be identified. More precisely, the dataset must be processed in such a way that it can no longer be used to identify a natural person by using all the means likely reasonably to be used by any party in or outside the process. One consequence is that the anonymization processing must be irreversible.
There are then several techniques that can be applied to anonymize a data set: some basic methods (substitution, blurring, aggregation, etc.), k-anonymity types [163], differential privacy based [80], synthetic data, etc. Within SUPERCLOUD, we will mainly focus on k-anonymity and differential privacy, as these are among the most relevant ones for the SUPERCLOUD use cases dedicated to health services.

3.1.4.1 k-Anonymity

k-anonymity computation allows the public release of sensitive information without privacy disclosure. The aim of this technique is to allow a client to choose the degree of anonymization of an original database. The modified data source can then be published without revealing any personal information of the records because the sets are indistinguishable from each other. Sweeney invented k-anonymity in [163]. Since this seminal paper, where it is proposed to replace an attribute by a more general value (generalization) or a value's digit by an asterisk (suppression), many improvements have been proposed, for example with multidimensional greedy algorithms or by adding equivalence classes and distribution within the algorithm. Authors of [139] have shown that considering data quality and integrity, the best way to publish anonymized data is to combine generalizing and suppression on a "cost-effective" level with the distribution of sensible attributes inside a data source.
In practice, organizations such as hospitals are often asked to publish their statistics about medical data for research or other purposes. These data contain different kind of attributes: *explicit identifiers* which clearly identify individuals (name), *quasi identifiers* that could lead to data disclosure when their values are taken together (zip code, gender, age) and *sensitive attributes* like disease or salary. When statistics are published, the released data may not lead to information disclosure.
By usage of k-anonymity, the anonymity of data is satisfied if each sequence of values k in a data table occurs at least k times. In a k-anonymized table, each record is indistinguishable from at least $k-1$ other records with respect to the quasi identifiers. However, k-anonymity is limited in its assumption of adversarial knowledge and protects data only against specific attack scenarios, which give observers the chance to gain enough information to break it [82]. An adversary gains information about patient's diseases from the records of the released data table with similar sensitive attributes and quasi identifiers, and learns from it especially when datasets are close to each other and have similar semantic accordances. With this in mind, an observer with enough background knowledge can easily guess which record belongs to which patient. To avoid this scenario, a distribution, called t-closeness, is added to the released data. Such principle is based on the distribution of sensitive attributes within an equivalence class e. Within the whole table, the distance of one entry to another may not be more than a threshold t. In practice, Earth Mover's Distance (EMD) evaluates similarities between two distributions and calculates the amount of work to transport one distribution to the other by moving mass between them [139]. The t-closeness uses EMD to calculate the distance between categorical

attributes.
EMD distributes sensitive attributes from a data table such that no attributes out of the same categorical attribute group are next to each other which limits attribute disclosure attacks. The combination of k-anonymity and t-closeness is a way to release data, and maintains data quality and integrity at the same time, in contrast to other information disclosure techniques [139].

3.1.4.2 Differential privacy

Among the new techniques used in anonymization, differential privacy [80] is one of the most promising ones. It is at the same time a way to measure the reached level of privacy, and a method to perform anonymization. As the underlying basis mainly concerns advanced mathematics, we only provide here the main notions and do not go deeply into details. Informally speaking, a differentially private mechanism ensures that any of its outputs is essentially equally likely to occur, independent from the presence or absence of any individual in the database. From the point of view of the individuals present/mentioned in the database, computations which provide differential privacy behave almost as if their records had not been included in the analysis.
Consider for example the queries "average age of all people with sleeping problems" and "average age of all people *except* X with sleeping problems". While these queries may individually be answered without the danger of privacy leakage, the combination of the queries obviously leaks whether person X has sleeping problems (and if so, it gives information about person X's age). The problem is that the information about one single person (i.e., record) has a direct influence on the query result. Differential privacy remedies this by adding noise to the answers of queries: intuitively, by adding noise that is bigger than the variation in the query answer due to any one individual, it is ensured that the answer does not leak information about anybody.
A common way to design an ϵ-differentially private randomized mechanism is to add (Laplace) noise to some query on the considered database.

3.2 Security of sharing and secure deduplication mechanisms

We now review in this section the different cryptographic techniques that today permit users to encrypt their personal data, while giving them the possibility to share them with third parties. We also review the deduplication problem over encrypted data. The way such tools can be used with the SUPERCLOUD a architecture will be given in Chapter 5.

3.2.1 Proxy re-encryption

Proxy re-encryption [43] (PRE) allows a user to delegatee his/her decryption capability in case of unavailability. To do so, this user, Alice, computes a *re-encryption key* $R_{A\to B}$ with her secret key sk_A and her delegatee's public key pk_B, which is given to a semi-trusted proxy. This re-encryption key allows the proxy to transform a ciphertext originally intended to Alice into a ciphertext intended to Bob. While doing this transformation, the proxy cannot learn *any* information on the plaintext. There are then several ways to characterize a PRE.
In this way, a PRE is said *unidirectional* if, with a re-encryption key $R_{A\to B}$, a proxy cannot translate a ciphertext intended to Bob into a ciphertext intended to Alice. On contrary, schemes which allow this symmetrical transformation are called *bi-directional*. Then, a PRE scheme is *single-hop* if once a message has been moved into a ciphertext intended to Bob, no more transformation on the new ciphertext to Bob is possible. A scheme which allows several translation of ciphertexts is called *multi-hop*. It also exists a way to combine these properties, as proposed in [55].
Numerous papers on PRE schemes exist. Some of them are unidirectional and single-hop [27, 122, 155, 25, 175, 67, 56] and some others are bidirectional and multi-hop [43, 57, 131].
In [27], Ateniese *et al.* have proposed a privacy-preserving architecture for distributed storage which makes use of a proxy re-encryption (PRE) scheme. A similar PRE based storage system has also been

proposed in the case of cloud storage in [166]. With such a system, where the cloud plays the role of the proxy, the access to a plaintext is only permitted to authorized users, while the cloud cannot derive the plaintext from the stored ciphertext. A data can *e.g.* be stored on a dedicated cloud storage using Alice's public key. If Bob is authorized to access this document, the proxy/cloud makes use of the re-encryption key from Alice to Bob. Similarly, if Alice owns several devices, one document encrypted with the key on one of them can be re-encrypted for another one. This is done without needing them to share the same secret decryption key, and in a private way since the storage provider does not obtain the data in plain.

3.2.2 Attribute-based encryption

The concept of attribute-based encryption [151] (ABE) permits the encryption and decryption of data based on the user's attributes. There exist two variants of ABE: *ciphertext-policy* attribute-based encryption (CP-ABE) and *key-policy* attribute-based encryption (KP-ABE). In a CP-ABE scheme, the secret key is associated with a set of attributes and the ciphertext is associated with an access policy (structure) over a universe of attributes: a user can then decrypt a given ciphertext if the set of attributes related to his/her secret key satisfies the access policy underlying the ciphertext. In contrast, in a KP-ABE scheme, the access policy is for the secret key and the set of attributes is for the ciphertext. In the context of ABE, the set of privileged users is determined by an access policy. To date, several types of access policies are investigated. For example, AND-gates access policy means that there is only AND operator in an access formula. Threshold access policy means that there is no distinction among attributes in the access policy, anyone who possesses enough attributes (equal or bigger than a threshold chosen by the sender) will be able to decrypt. In some modern applications, finer-grained access control is needed such as Disjunctive Normal Form (DNF, i.e., with disjunctions (OR) of conjunctions (AND)) or Conjunctive Normal Form (CNF, i.e., with conjunctions (AND) of disjunctions (OR)).
Since their introduction in 2005, one can find many papers proposing ABE schemes [151, 92, 62, 144, 81, 63, 104, 99, 29, 141, 150, 65, 178]. The authors in [29, 178] introduced KP-ABE schemes with constant-size ciphertext. The works in [92] extended the Sahai and Waters' work [151] to propose the first schemes supporting finer-grained access control, specified by a Boolean formula. Non-monotonic access structures permitting to handle the negation of attributes has been considered in subsequent works [144, 29, 178]. Adaptive security for ABE scheme was considered in [118, 65, 28, 174] using composite order group, and then in [141, 66] using prime order groups. Similarly, dynamic ABE scheme (unbounded attributes) was first investigated in [119] using composite order groups and then in [150] using prime order groups.
In traditional ABE schemes, one single "central authority" issues all secret keys, introducing a single point of failure in any architecutre using it. In [62], it is shown that the issuing of the part of a secret key related to particular attributes can be delegated to so-called "attribute authorities" in a secure way; however, the central authority still needs to issue a part of the secret key to each user. Hence, this scheme mainly simplifies the organisation of key issuing and not its security. In [63], it is shown that, in such settings, the functionality of the central authority can be distributed between the different attribute authorities so that one single central authority is no longer needed. In this case, the single point of failure is eliminated because there is no single party holding a key that enables the decryption of all ciphertexts.
As for proxy re-encryption scheme, an ABE can be used to share data inside a cloud infrastructure. In fact, one can upload encrypted documents and then permit other customers to obtain the right to download and read the document by adding an access control policy based on customers' attributes.

3.2.3 Secure deduplication

Data deduplication is a method to reduce the storage needs, by eliminating unrelated redundant data [148]. It roughly consists in: finding copies of identical data, storing it once, and replacing all

copies by pointers to that single instance. It allows cloud providers to save costs in a significant manner: if several users upload the same data, the cloud provider stores only one copy and returns it to each user who requests it. When the files are encrypted, two main challenges arise. First, encrypting twice the same plaintext does not yield the same ciphertext. Second, as the keys are generated independently by users, none of them can decrypt a ciphertext encrypted by another user.
A deduplication algorithm is secure if it correctly achieves the space reduction while protecting data from unauthorized accesses. It requires balancing several trade-offs that differ for each use case.
Intra-user deduplication compares data from a single user, while inter-user deduplication does so for several users that may own identical files. We focus on inter-user deduplication, which is generally more efficient, but incurs in increased security risks.
Deduplication can be performed at the client or server side. In the former, clients either store a consistent global index locally or verify data duplicity by sending fingerprints to a global index server. However, both methods are vulnerable to side-channel [97, 54] and vector attacks [136]. Solutions for these problems are based on presenting challenges for users and include proofs of retrievability [103], possession [26], or ownership [95, 177, 42, 90].
In the latter, clients send data to a server that transparently performs the deduplication against previously seen data from all users. The security problem for this case lies in sending the data in clear through the communication channels (e.g., the Internet), which may be unacceptable in some use cases. Ways of circumventing this problem encompass encrypting data at the client side. However, standard encryption techniques obstruct data deduplication since the same data input results in different encrypted data for different users (i.e., every user has its own encryption key).
Baracaldo *et al.* [32] proposed a deduplication framework that has a trusted server that stores the clients' keys, decrypts received data, deduplicates it, and re-encrypts the compressed and deduplicated data in the back-end storage. A solution that has gained traction is the convergent encryption [76, 36], which uses fingerprints from the original data as the encryption key, and allows inter-user deduplication over encrypted data. Users encrypt their data with a deterministic scheme, and use a key corresponding to the deterministic hash value of the message. Thus, the same message encrypted independently by different users always yields the same ciphertext and deduplication can be operated on the encrypted data. This encryption scheme is widely used in research and in practice [24, 72, 83, 162, 130]. Additionally, the same convergent approach can also be applied in secret sharing schemes [121].
However, convergent encryption remained deterministic. Bellare, Keelveedhi and Ristenpart (BKR) introduced message locked encryption (MLE) in [36] and formally investigated security issues, in order to capture a better security notion for secure deduplication. Schemes which followed improved security achieved with MLE, however, this comes at the cost of efficiency [18, 35]. Finding an efficient MLE scheme with best achievable possible security remains an open problem.

3.3 Integrity protection mechanisms for an untrusted cloud

In the Byzantine fault model [117], faulty processes and storage objects (or *clouds*) may modify data structures (within the limits of cryptography) or perform other arbitrary operations to deliberately disrupt executions. Therefore, it is not surprising that in the context of cloud computing, where there are inherent trust limitations [52, 171], a lot of research on algorithms and protocols was expressly conceived to deal with Byzantine faults, i.e faults that encompass arbitrary and malicious behavior. Together with these algorithms, new consistency models were defined that reshaped the correctness conditions in accordance to what is actually attainable when coping with such strong fault assumptions.
The forefather of the line of research that tried to define what level of consistency and data integrity is possible to achieve with an untrusted cloud (server), is the work of Mazières and Sasha [133] that introduced the notion of *fork* (or fork-linerizable) *consistency*. A fork-linearizable system ensures that the operations issued by processes are linearizable and guarantees that if the storage system causes the histories of two processes to differ even for just a single event, they may never again see each other's updates after that without the cloud/server being exposed as faulty. Namely, any divergence in the

histories perceived by different groups of correct processes can be easily spotted by using any available communication protocol between them (e.g., out-of-band communication, gossip protocols, etc.).
A subsequent consistency and integrity model named *fork* consistency* was defined in [120] in order to allow the design of protocols that would offer better performance and liveness guarantees. Fork* consistency relaxes the conditions of fork consistency by allowing forked groups of processes to observe at most one common operation issued by a certain correct process.
Another variant of fork-consistency, namely *fork-sequential consistency* [143, 51], requires that whenever an operation gets visible to multiple processes, then the history of events occurring before the operation is the same at all these processes. For instance, when a process reads a value written by another process, the reader is assured to be consistent with the writer up to its write operation. Essentially, similarly to sequential consistency, a global order of operations is ensured up to a common visible event.
Mahajan et al. define in [127] *fork-join causal consistency* (FJC) as a weakening of causal consistency that can preserve safeness and availability in spite of Byzantine faults. In a fork-join causal consistent storage system if a write event e issued by a correct process depends on a write event e' issued by any node, then e' becomes visible before e at any correct node. In other words, FJC enforces causal consistency among correct processes. Besides, partitioned groups of processes are allowed to reconcile their histories through merging policies, since inconsistent writes by a Byzantine process are treated as concurrent writes by multiple *virtual processes*. *Bounded fork-join causal* [126] refines this clause by limiting the number of forks accepted from a faulty node and thus bounding the number of virtual nodes needed to represent each faulty node.
Finally, in the context of a single untrusted cloud, *weak fork-linearizability* [53] relaxes fork-linearizability conditions in two ways: (1) after being partitioned in different groups, two processes may share the visibility of one more operation (i.e., *at-most-one-join*, as in fork* consistency) and (2) the real-time order of the last visible operation by each process may not be preserved (i.e., *weak real-time order*). These two conditions enable the design of protocols that allow for improved liveness guarantees (i.e., *wait freedom*). Weak fork-linearizability ensures linearizable operations in case of executions enacted only by correct processes, whereas in the presence of Byzantine failures it guarantees causal consistency.
Feasible consistency semantics in the context of interactions of correct clients with an untrusted cloud have been captured within the family of the above *fork-based* consistency and integrity guarantees. On the other hand, in the context of multiple untrusted clouds, relevant to our project, Byzantine fault tolerance could be applied to mask certain fault patterns [171, 38, 33] and even implement strong consistency semantics (e.g., linearizability) [39, 75]. Theses mechanisms, that SUPERCLOUD data management will build upon, are described in more detail in Section 3.5.

3.4 State Machine Geo-replication

In the state machine replication (SMR) model[115, 152] an arbitrary number of clients send requests to a set of servers, which hosts replicas of a statefull service that updates its state after processing those requests. The goal of this technique is to make the state at each replica evolve in a consistent way, resulting in a service which is accurately replicated across all replicas. Since the service state was already updated by the time clients receive a reply from the service, this technique is able to offer *strong consistency*, i.e., linearizability [98]. To enforce this behavior, it is necessary to enforce three properties:

1. client requests are delivered to the replicas via total order broadcast;
2. replicas start their execution in the same state; and
3. replicas modify their state in a deterministic way.

Practical SMR protocols for tolerating crashes such as Paxos [116] and RAFT [142] are usually employed for supporting critical services in Internet-scale infrastructures. For example, these protocols

have been used in coordination services [100, 49], metadata storage [39], management services [147] and even general purpose raw data stores [45, 40].
Although replicating services within the same datacenter significantly improves the availability and durability of the service, the criticality of modern internet-scale services have created the need for geo-replication protocols for disaster tolerance, including whole-datacenter failures (e.g., [69]). Following this trend, several works propose (1) fast geo-replication systems providing "as consistent as possible" services, (2) optimizations for transactional systems spanning multiple sites and (3) strongly consistent wide-area SMR protocols. In the remaining of this is section we focus our discussion on (3), as it perfectly fits the consistency and data integrity requirements of SUPERCLOUD.
Steward [23] is a state machine replication system that employs a hierarchical Byzantine fault-tolerant protocol for geographically dispersed multi-site systems. It is a hybrid algorithm that runs a BFT agreement protocol within each site, and a lightweight, crash fault-tolerant protocol across sites. This protocol needs $3f_i+1$ replicas at each site i to run the BFT agreement protocol and uses a lightweight protocol among sites. Even though Steward is able to perform well in WANs (when compared with PBFT [59]), that comes at the cost of a very complex protocol (over ten specialized algorithms that run within and among sites) that demands plenty of resources (e.g., each site requires at least 4 replicas).
Mencius [129] is an SMR protocol derived from Paxos [116] which is optimized to execute in WANs. Like Paxos, it can survive up to f crashed replicas out of at least $2f+1$. Replicas take turns as the leader and propose client requests in their turns. Clients send requests to the replicas in their sites, which are submitted for ordering when the replicas becomes the leader. According to the authors, this rotating coordinator mechanism can significantly improve the system throughput (by distributing the load of being the leader among replicas) and reduce latency in WANs (since the communication between client and leader is within the site).
The main goal of Mencius is to replace the single leader of classical SMR protocols such as Paxos, with multiple leaders to decrease the latency (as leaders would be closer to the clients). However, synchronization among leaders themselves introduces another tradeoff. To illustrate this, consider Mencius' multiple leaders scheme that evenly partitions the sequence number space across all replicas, so that each replica/leader can propose requests (e.g., every leader k sequences requests with sequence number $m = k \ (mod \ n)$, where n is the number of leaders). In case a leader lags behind, it skips the missing sequence numbers to catch up with other leaders. However, a skipping leader must let its peers know it skips sequence numbers — leading to what is known as the "delayed commit" problem.
Clock-RSM [77] addresses the delayed commit problem of Mencius using loosely synchronized clocks. In Clock-RSM, each replica proposes requests piggybacked with its physical clock timestamp, instead of a logical sequence number. Requests are then ordered by associated clock timestamps. Replicas synchronize their physical clocks periodically. We note that Clock-RSM only attenuates the delayed commit problem but does not entirely solve it, bringing the synchronization latency from a round trip message delay between leaders to a single message delay.
Egalitarian Paxos (EPaxos) [135] is a recent SMR protocol also derived from Paxos and designed to execute in WANs. Unlike most SMR protocols inspired by Paxos, EPaxos does not rely on a single designated leader for ordering operations. Instead, EPaxos enables clients to choose which replica should propose their operations, and employs a mechanism for solving conflicts between interfering operations. However, in order to correctly enforce such conflict resolution - and contrary to most SMR protocols - EPaxos requires information from the application to determine operations interference.
EBAWA [168] is a Byzantine-resilient SMR protocol optimized for WANs. It considers a hybrid fault model in which each replica uses a local trusted/trustworthy service (that cannot be compromised) to provide tolerance to up to f Byzantine faults using only $2f+1$ replicas. Similarly to Mencius, it uses a rotating leader to allow clients to send requests to the replicas that are close to them.
Recently, and within the context of SUPERCLOUD, we conducted an extensive experimental study to assess which of the well-know optimizations proposed for improving the latency of SMR protocols on wide-area networks bring most benefits in practical deployments [158]. The key findings of the study is that rotating the leader is much less effective than having the leader deployed in the most

well-connected replica and that assigning weights to the replicas (in accordance with their relative speeds) brings substantial benefits. These results lead to the development of WHEAT, a protocol that employs all optimizations that lead to substantial improvements in the SMR latency, including a novel weight assignment scheme that ensures faster (i.e., well-connected) replicas have more importance in deciding the order of messages.
In practice, geographically distributed systems such as Spanner [69] often employ sharding of SMR to synchronize disjoint state partitions. Similar approaches have been suggested in other research works (e.g., [111]).

3.5 Storage in Cloud of Clouds

In this section we overview storage solutions spanning multiple clouds. We start with hybrid cloud solutions, spanning a private and (typically) a single public cloud (Sec. 3.5.1), and then we proceed to discuss solutions spanning more than two clouds (Sec. 3.5.2)

3.5.1 Hybrid Cloud

There are many commercial storage proxies to transparently integrate local systems with cloud storage forming basic variants of hybrid cloud storage (e.g. [11, 15]). Most of them follow an architecture similar to BlueSky [170] and Iris [160], where a CIFS/NFS proxy is used to mediate access to a single cloud provider. In general, such systems do not support the coordination of file accesses between different proxies, and thus are inappropriate to share geographically-dispersed data, as we are considering in SUPERCLOUD.
Almost every leading storage vendor provides its own proprietary hybrid cloud solution addressing these issues. Hybrid cloud solutions typically consist of deploying storage appliances at the premises of a client, which are then coupled with remote storage on the public cloud of the storage vendor. As a rule of thumb, more sensitive data is stored on private premises whereas less sensitive data is outsourced to public clouds. In a complementary use case, public cloud is often used to boost reliability in the hybrid cloud architecture, storing backup or hot replicas of data that anyway resides on private premises.
These proprietary hybrid cloud solutions are rarely interoperable with commodity public cloud storage services and are tailored to specific solutions of a cloud and storage vendor, resulting in so called "vendor lock-in". In recent years, this picture started changing with vendors such as IBM announcing solutions for integration of public cloud storage services with on-premises storage.[1] Complementary solutions, that mix the hybrid cloud concept with that of public multi-clouds (that we review in the next section), also started to appear [75].

3.5.2 Multi-cloud storage

In the last few years, several research teams have been proposing the use of multiple cloud providers, beyond private/public hybrid clouds, for improving the integrity and availability of stored data [21, 33, 38, 109, 176, 39, 75]. The idea was first introduced in archival systems like SafeStore [109] and RACS [21], with the latter requiring no modifications to the storage clouds or servers running in the cloud. More recent systems like ICStore [33] and DepSky [38] provide object storage (i.e., variable-size read/write registers) considering different fault models and unmodified storage clouds.
SPANStore [176] is a system that tries to optimize data placement and consistency guarantees taking into account several metrics of cloud providers (e.g., cost and access latency). The main limitation of this approach is that it requires servers deployed in all cloud providers, which implies additional costs and management complexity. The same limitation applies to modern (single-provider) geo-replicated storage systems such as Spanner [69] and Pileus [165], if deployed in multiple cloud providers.

[1] `http://www.theregister.co.uk/2013/12/12/big_blues_crosscloud_migration/`.

Some recent cloud-backed storage systems employ a hybrid approach in which unmodified cloud storage services are used together with few computing nodes for storing metadata and coordinating data access [39, 75] across multiple commodity clouds. SCFS, in particular, provides controlled sharing of desktop-size files stored in multiple providers by using a cloud-deployed coordination service to implement locking [39]. Hybris [75] stores metadata on private cloud which are then used to boost the reliability and consistency of data stored in multiple public clouds.
A slightly different kind of work proposes the aggregation of multiple file synchronization services (e.g., DropBox, Box, Google Drive, etc.) in a single dependable service [68, 96, 164]. A key asset of these works is that they do not require any external service or component that need to be trusted. CYRUS [68] does not implement any kind of concurrency control, allowing different clients to create different versions of files accessed concurrently. These divergent versions are expected to be solved by users. MetaSync [96] solves concurrent writes in a different way: all writes on shared files are applied in total order, using a variant of the Byzantine Disk Paxos [20] called pPaxos, that works on top of file synchronization services. Finally, UniDrive [164] employs a locking protocol based on quorums, in which a client has access to a shared folder as long as it is able to write a lock file alone in a majority of the services.
Orthogonally to multi-cloud storage that aims at boosting reliability and dependability of cloud storage [171], which we overviewed so far, multi-cloud solutions that aim at integrating cloud (storage) services from multiple clouds have started to emerge. Openstack Mercador [12] is a recent and emerging project in the Openstack ecosystem that aims at facilitating Openstack deployments that would consist of cloud services other than those directly deployed on bare-metal.

3.6 Summary of real-world technologies

In this section, we review some real-world technologies dedicated to the security of data management systems in the cloud. We focus on some existing products permitting to share encrypted data, to protect the data that are stored on the cloud, and on anonymization tools.

3.6.1 Sharing encrypted data

There are today multiple solutions permitting customers to store their own data on the cloud (*e.g.* Google Drive, Dropbox) but most of them do not permit user-centric data encryption. One reason is that it does not easily permit data sharing. Despite few commercial initiatives, but these are mostly not based on advanced cryptographic tools.
The MEGA platform [9] is one of such example. After the closing of Megaupload, the founders have decided to create a new platform for data hosting for which they do not have the possibility to obtain the stored data in clear, whereas it was still possible to share data among customers. They propose a system in which the encryption key is shared among the customers having access to the encrypted data.
A similar technical approach has been proposed by Boxcryptor [2], which proposes an application permitting the customers of the most popular cloud storage service providers (Google Drive, Dropbox, etc.) to continue to store their data but in an encrypted way.
Numerous other commercial products permit encrypting data on cloud, and putting the security on customer's hands. Most of them make use of key sharing (with the inherent problem of key loss and the resulting important problem) of data duplication (with good security but poor flexibility regarding the dynamicity of the data and customers). To the best of our knowledge, no existing commercial product makes use of advanced cryptographic tools as we propose to use in SUPERCLOUD (except the AlephCloud in the USA, which has been closed recently).

3.6.2 Middleware encryption platform

Another way to protect data stored in the cloud is to provide a middleware dedicated to the data encryption and decryption (or tokenization). The data confidentiality is then not verified *w.r.t.* the middleware itself, but *w.r.t.* the rest of the world. It then permits to perform computations over protected data since the resulting information can be close to what can be obtained with non-protected data.
Today, several tools provide such kinds of middleware and the most well-known are Perspecsys [16] and CipherCloud [5], which moreover permit to interact with existing cloud service providers such as Microsoft or Salesforce.

3.6.3 Anonymization tools

Regarding data anonymization, several commercial products exist that provide "simple" ways to anonymize data, using basic methods such as data substitution or data blurring. This is for example the case for the IBM Optim product [7], but also for Oracle Data Masking Pack [13], Informatica Data Privacy [8], or Data Masker [6].
Identity Mixer (idemix) is an anonymous credential system developed at IBM Research - Zurich that enables strong authentication and privacy at the same time. As of 2015, it is deployed in demo version on IBM BlueMix Platform as a Service cloud [17]. With Identity Mixer anonymous credentials, a user can selectively reveal any of the attributes contained in the credential without revealing any of their information whatsoever. Thus, anonymous credentials are a key ingredient to protect one's privacy in an electronic world. With Identity Mixer, users can obtain from an issuer a credential containing all the information the issuer is ready to attest about them. When a user later wants to prove to a service provider a statement about her, she employs identity mixer to securely transform the issued credential. The transformed credential will only contain the subset of the attested information that she is willing to disclose. The user can apply this transformation as many times as she wants and still none of credentials can be linked to each other.

3.6.4 Dependability techniques

Today, almost all major Infrastructure as a Service cloud providers (e.g., Amazon, Google, Microsoft Azure, IBM Softlayer, Rackspace, etc.) offer the option of replication across multiple datacenters. These however rely on proprietary protocols and cannot be used to orchestrate multiple clouds under different administrative domains, i.e., belonging to different cloud providers. Similarly, major storage hardware vendors (IBM, EMC/Dell, NetApp, etc.) offer proprietary hybrid cloud solutions.
No widely adopted open source solution for improving dependability in the context of cloud of clouds exists at the moment. However, there are several open source projects, developed in earlier projects by SUPERCLOUD partners, that are used by many research groups around the world. These include BFT-SMaRt [41], SCFS [39], Depsky [38] and Hybris [75].
Finally, on the end user side of the spectrum, many storage front-ends that synchronize and organize multiple public cloud storage services such as MultCloud [10], Otixo [14], CloudCube Android app and others exist, yet these have no goals of boosting dependability nor are they oriented towards business users.

Chapter 4 High-level architecture of the data management layer

At the high level, the SUPERCLOUD data management architecture comprises a mixture of aggregation of resources at multiple clouds (private and/or public) and value-added functionalities for increasing dependability and security. As SUPERCLOUD data management adds many different and often complementary functionalities, we aim at an understandable logical architecture that will accommodate different value-added security and dependability features.
In the following, we first present the logical architecture of data management in SUPERCLOUD (Sec. 4.1) which aims at unifying value-added features. We then outline the specific security (Sec. 4.2) and dependability (Sec. 4.3) features and discuss how they fit into the logical architecture. Specific architectures of dependability and security features that comply with the high level logical architecture are postponed to Chapters 5 and 6, respectively.

4.1 Logical architecture

4.1.1 Storage and data management entities

A simplified view of the logical architecture of the SUPERCLOUD data management layer is shown in Figure 4.1.

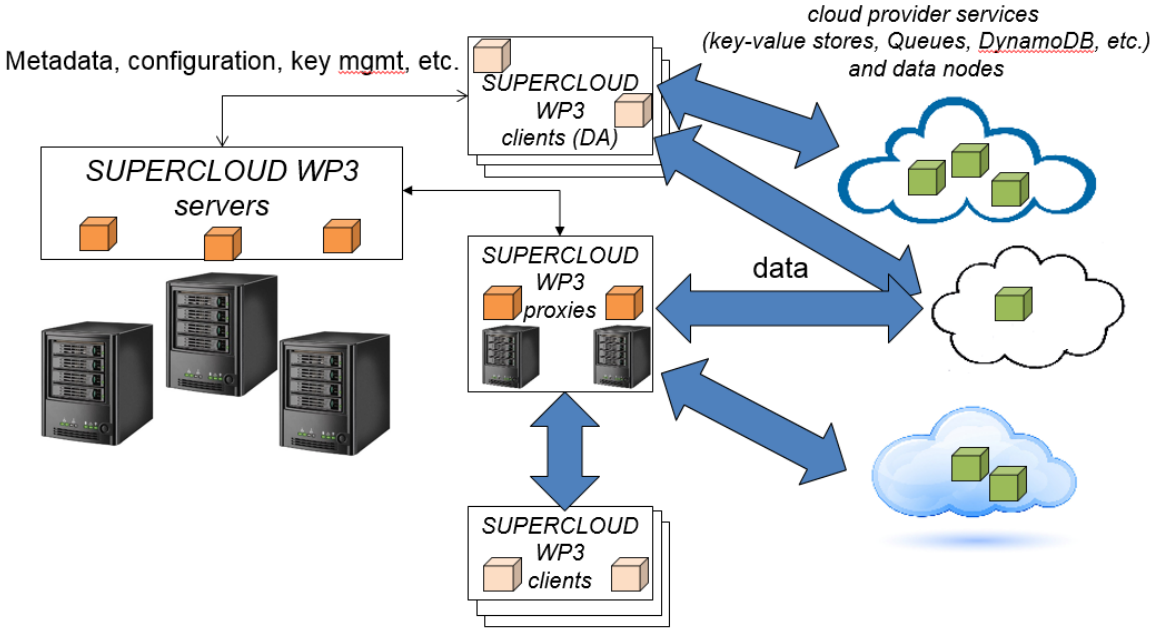


Figure 4.1: High level logical architecture of the SUPERCLOUD data management layer.

Our logical architecture is comprised of five classes of entities:

- *Storage clients* (or simply *clients*), users of SUPERCLOUD storage infrastructure. These are all L2 virtual machines (or containers) deployed in the User clouds. Other storage entities may or may not act as storage clients depending on whether they are deployed as L1 virtual machines, in which case they use cloud provider storage or L2 virtual machines in which case they act as storage clients.

  Clients are divided into two subcategories: *direct accessors (DA)* and ordinary clients. Ordinary clients interact with the SUPERCLOUD data management layer via *storage proxy* and are entirely oblivious the SUPERCLOUD data management layer. This requires minimum changes to clients without installing additional libraries. In contrast, DA clients run SUPERCLOUD specific logic as a client library and can interact and access directly *storage servers* and L1 *cloud provider services*. DA clients can also have certain functionality of storage servers built-in, making them also possibly independent of storage servers.

- *Storage proxies* (or simply *proxies*), are L2 virtual machines (or containers).Proxies are in principle stateless, and can be easily added dynamically to the system. Their primary goal is facilitating clients' access to SUPERCLOUD storage and data management offerings. As we will detail later, examples of proxies include encryption proxies, secure deduplication proxies, etc.

- *Storage servers* (or simply *servers*) are stateful L1 or L2 virtual machines (or containers). The main role of servers is housekeeping of critical portions of metadata vital to operation of the SUPERCLOUD data management layer. Examples of SUPERCLOUD storage servers includes storage metadata servers, data integrity servers, configuration management servers, etc.

- *Cloud provider services* (CPS) are L1 cloud storage services that DA clients or proxies can directly access. They expose different APIs notably object storage and block storage. Examples include Openstack Swift, Amazon S3 and EBS, etc.

- *Cloud provider data nodes* (or simply data nodes) are L1 virtual machines or containers that reside on L1 public or private clouds comprising SUPERCLOUD infrastructure. Complementing CPS, data nodes can perform computation and have locally mounted L1 block storage that ends up storing SUPERCLOUD user data.

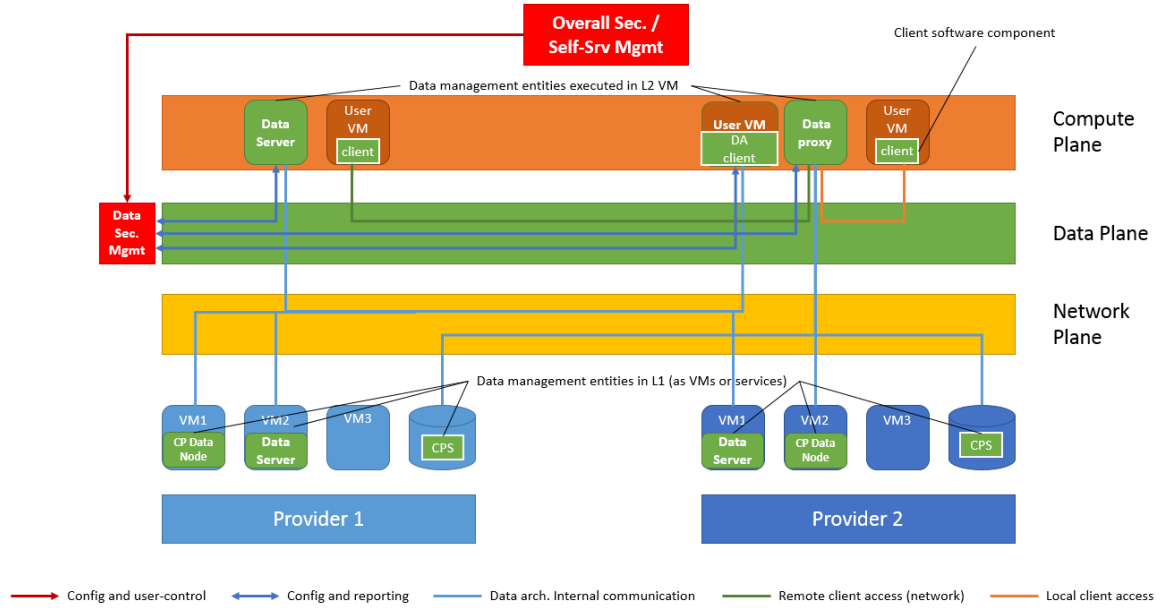


Figure 4.2: Mapping of SUPERCLOUD data management entities to L1 and L2 virtualization layers.

As we can see, storage entities are distributed across L1 and L2 virtualization layers, to facilitate the understanding of this distribution, we depict a possible mapping of SUPERCLOUD storage and data management entities to virtualization layers in Figure 4.2.

4.1.2 Network and trust model

In SUPERCLOUD, L1 entities such as storage servers, CSPs, data nodes and possibly proxies are not necessarily trusted. Sometimes even L2 entities such as clients might not be entirely trusted for different security reasons.
The main goal of SUPERCLOUD is precisely to develop protocols to maximize dependability and security of storage exposed to L2 machines and containers, despite lack of trust in different L1 entities or even misbehaving L2 entities. Depending on the nature of L1 clouds as well as User and use case preferences and configuration parameters, SUPERCLOUD data management layer will adapt to leverage trust in certain L1 entities, in case such entities exist.
That said, any component or entity in the above described architecture is assumed to be at least crash-fault prone. SUPERCLOUD data management protocols will tolerate misbehaving and crash faulty nodes with the goal of maximizing such resilience.
Furthermore, the network (whether L2 or L1) is not assumed to be reliable. Put differently, asynchronous communication and network faults/partitions are a reality for SUPERCLOUD and our protocols shall take this into account. Whereas SUPERCLOUD data management layer shall tolerate both untrusted (Byzantine) entities and network faults the specific nature of SUPERCLOUD deployment allows revisiting the correlation between Byzantine entity faults and network faults. Whereas traditional adversary models take for granted that adversary can control both the network and the Byzantine nodes, in our case this might not be entirely true, due to the diversity of the networks interconnecting different cloud providers that are difficult to control.

4.2 Security features

In this section we describe briefly the security features we will develop in WP3 and explain how this maps to the logical architecture introduced in Section 4.1. Architectural details are postponed to Chapter 5.
Our principal goal is to protect the data related to end-users. The general way to protect the data is to encrypt it, so that only authorized entities can access it. But, as shown by the requirements, in Chapter 2, the data also needs to be manipulated. We therefore need to adapt encryption techniques to these requirements. In fact, the security of clients' data can be divided into two kinds of features. Informally, when the data are stored and used by some entities different from the end-user himself: we talk about "privacy". Moreover, when the user himself stores and makes use of some of his data: we then talk about "security of storage". We define these concept more precisely in the following.

4.2.1 Privacy Mechanisms

We first propose to use several technical tools to protect the privacy of individuals, such as the ones sketched in Chapter 3. In this case, some data related to individuals are stored on one or several cloud storage service providers and a third party wants to make use of these data for some allowed purposes. We then consider several cases:

- the data is protected using encryption by the client itself, and the third party is able to do some "search" on these encrypted data. The suitable tool for that purpose is *searchable encryption*, which requires SUPERCLOUD clients[1] to be able to encrypt (and sometimes decrypt) the data. We then consider that the SUPERCLOUD WP3 proxy is the entity performing the actual search,

[1] In this case, the data are not necessarily related to this client. The client can *e.g.* be a hospital and the data are related to patients.

based on the data stored on either SUPERCLOUD or on external storage service providers, as well as on some meta-data stored by the clients (through the proxy) on the SUPERCLOUD WP3 servers;

- the input data is kept secret by SUPERCLOUD servers and/or external storage service providers and the client requires secure computation on this dataset. In this case, *secure multi party computation* is a suitable cryptographic tool. After a request by a SUPERCLOUD client, the proxy executes the multi-party computation, based on all the secret inputs. The client finally obtains the result and the inputs are totally protected. This technique of multi-party computation is also applied to data anonymization, with two different kinds of techniques, that is k-anonymity and differential privacy;

- the data access patterns can also be protected from untrusted cloud storage providers by using the techniques of *oblivious storage*. Part of the client work is here delegated to the proxies, while the servers are responsible for the storage of the index of stored files.

4.2.2 Security of Sharing

We also make use of advanced cryptographic techniques to protect the mechanisms for sharing data stored by SUPERCLOUD clients. In this case, clients stores and then share (or use) the encrypted data, while requiring data security with respect to cloud storage providers, but also internal SUPERCLOUD components.
We study several cases:

- the data is shared by one SUPERCLOUD client with another, designated SUPERCLOUD client. To this end, we will use *proxy re-encryption*. This implies, in the architecture, adding the re-encryption functionality to the SUPERCLOUD proxy component, while the data itself is sent to the underlying untrusted cloud storage provider. Some meta-data are also stored on the SUPERCLOUD servers, together with the cryptographic key allowing sharing (namely the re-encryption key);

- the data is shared by one SUPERCLOUD client with some SUPERCLOUD clients that have some specific attributes, by the means of *attribute-based encryption*. For this purpose, each client has a decryption key, based on its attributes, that are computed by a key generator functionality embedded into the proxy, with a master secret key securely stored on one or several servers (to avoid the reliability and security problems related to centralization of keys). Additionally, some meta-data are stored on the servers;

- for deduplication of encrypted data, we make use of cryptographic techniques that: (i) permit to detect that two different encrypted data correspond to the same plain data and (ii) permit two clients to be able to obtain the data in plain. With SUPERCLOUD, the detection of such duplication is done by the proxy, having access to some meta-data related to stored data that are stored on servers. Again, only the encrypted data are sent to underlying untrusted cloud storage providers.

4.3 Dependability features

In this section we describe briefly the dependability features we will develop in WP3 and explain how this maps to the logical architecture introduced in Section 4.1. Architectural details are postponed to Chapter 6.
The dependability architecture focuses on two major components, separating the control/metadata plane from the data plane:

- *Geo-replication of storage servers.* In the SUPERCLOUD architecture storage servers are the key component that store integrity and replication metadata, cryptographic keys, and other critical information. Therefore, it is essential to enable efficient geo-replication across multiple clouds of such servers. Geo-replication of storage servers is the key feature for boosting dependability in SUPERCLOUD. In SUPERCLOUD we will tackle the classical problem of state machine replication from novel angles.

- *Object/block store and file system architecture.* Besides novel techniques for replicating storage (metadata) servers, SUPERCLOUD builds innovative techniques for dealing efficiently with data scattered across clouds. In this context, we will develop novel data replication techniques to be implemented as improvements on proven technologies such as Ceph RADOS block device [4] and infrastructure developed in previous FP7 projects TClouds and BiobankCloud. The features on which SUPERCLOUD would specifically improve on are geo-replication and enabling user-defined storage across cloud of clouds.

We outline the specific features below. The reader should refer to Chapter 6 for more details.

4.3.1 Storage server geo-replication

Enabling efficient metadata replication across multiple clouds is a major focus of SUPERCLOUD. In this context we will enable the following features:

- *XFT state machine replication.* In SUPERCLOUD we will focus on a novel approach for storage server replication. Besides supporting traditional crash-fault tolerant (CFT) and Byzantine fault tolerant (BFT) solutions we will target protocols in the *XFT* (short for *cross fault tolerance*), model. The XFT model, that we introduced conceptually and recently in [123], decouples the fault space *across* machine and network faults dimensions, treating machine and network faults independently. In a nutshell, XFT models Byzantine and network faults as independent events, which is reasonable for many practical applications including the use cases and deployment models targeted by SUPERCLOUD. With such an approach, XFT offers protocols roughly at the cost of CFT with very storn reliability guarantees, sometimes even better than those of BFT itself.

- *Weight assignment for geo-replication.* Another approach we will take is to make geo-replication protocols *rely primarily on the fastest replicas present in the system*, and, at the same time *preserve its original safety and liveness properties*. To this end, we will explore the distribution of weights across replicas in such way that a majority is not always required to (correctly) make progress. This would involve using weighted quorums [172] in a novel, geo-replication context.

- *Dynamically reconfiguring the leader set.* In this context, we plan to explore the space between single-leader and multi-leader ("all-leader") for state machine replication (SMR), by dynamically reconfiguring the set of leaders. Each set of leaders is selected based on previous workload and network condition. Our preliminary results in this direction [124] are encouraging and show that, under typical imbalanced workloads that more often than not characterize real-world applications, our approach applied to state of the art geo-replication protocols efficiently reduces latency compared to native protocols. This optimization targets a sweet performance spot for both balanced and imbalanced workloads.

- *Elastic state machine replication.* Finally, in SUPERCLOUD we want to develop a generic *partition transfer* primitive (and protocol) that enables SMR-based services to be *elastic*, being thus more appropriate for implementing adaptive and self-managed services in the cloud. The idea is to design a protocol to perform transfers efficiently and with minimal perturbations on the client-perceived performance, on top of any existing SMR protocol.

4.3.2 Object/block stores and file systems

On the data plane, we will enable novel dependability features building existing storage technologies.

- *Enabling geo-replication for popular cloud block storage systems.* Currently, popular cloud block storage solutions, e.g., Ceph's RADOS block device (RBD) [4], are not optimized for geo-replication. Whereas adding geo-replicated storage data nodes to RBD is possible, the system does not distinguish remote data nodes from local data nodes. For instance, this may impact performance for a SUPERCLOUD user that wishes to use remote cloud services together with his local storage resources in the hybrid cloud setting. To rectify this, we will implement mechanisms for volume synchronization between clusters of multi-cloud block storage, with concrete focus on Ceph. The idea here is to start from implementing replication from a block storage volume in the source cluster to its replicated volume in the destination cluster.

- *User-defined Cloud-backed Storage.* In SUPERCLOUD, we want to build a user-centric/software-defined storage infrastructure that distributes its stored data by different cloud storage services, using different replication, coding and encryption techniques in accordance with the specifications provided during the virtual volume creation. The *management service* will allow users to define the storage policies they want, such as expected latency, storage overhead, security requirements, replication factor and location of data and the system will generate a virtual storage configuration defining how such requirements can be satisfied. This configuration plan will be used by *volume accessors* that can be implemented either as accessors (deployed in the machines using the virtual storage) or proxies (that export an NFS server interface).

4.4 Identity and Key Management

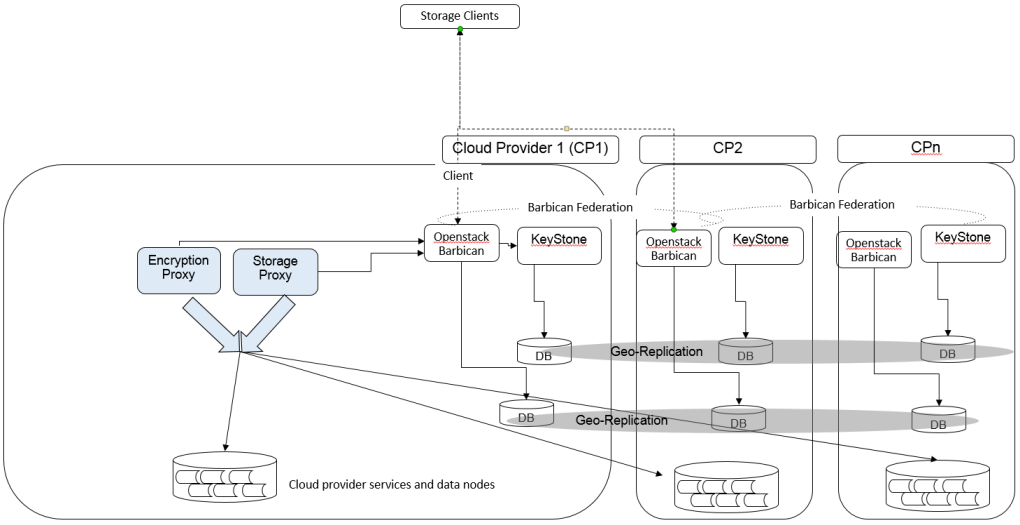


Figure 4.3: Federation of identity and key management.

SUPERCLOUD will build on proven cloud technologies for identity and key management. Namely, SUPERCLOUD will focus on levering Openstack Keystone for identity management and Openstack Barbican for key management.
These technologies, in particular Barbican, have a number of deficiencies when it comes to an enterprise environment and cloud-of-clouds in particular. These deficiencies impact the performance, scalability and security of cloud offerings based on Barbican. In the context of SUPERCLOUD we will address these problems by developing advanced features for OpenStack Barbican key management. In doing so we will make Barbican more suitable for enterprise deployment. We will focus on features such as

multi-tenancy/federation, fine-grained access control and Key Management Interoperability Protocol (KMIP) integration.
Barbican multi-tenancy and federation are in particular critical to cloud-of-clouds. Figure 4.3 illustrates a preliminary architecture of federated Barbican and its integration in the rest of the data management architecture. Specifically, the figure depicts how federated instances of Barbican and Keystone can be orchestrated. This approach will further be developed to align SUPERCLOUD efforts with Barbican community efforts [1].

Chapter 5 Data security architecture

In this Chapter we give details behind the SUPERCLOUD architecture of security features, that maps to the high-level logical architecture, given in Chapter 4. The features covered in this chapter shall enable privacy protection of SUPERCLOUD end-users and data sharing between SUPERCLOUD clients. More precisely, we here review some technical tools (mostly coming from cryptography) and show how they can be plugged into the general WP3 architecture. The main overview of our architecture, dedicated to data security, is given by Figure 5.1. It includes most of the functionalities that are detailed in this chapter.

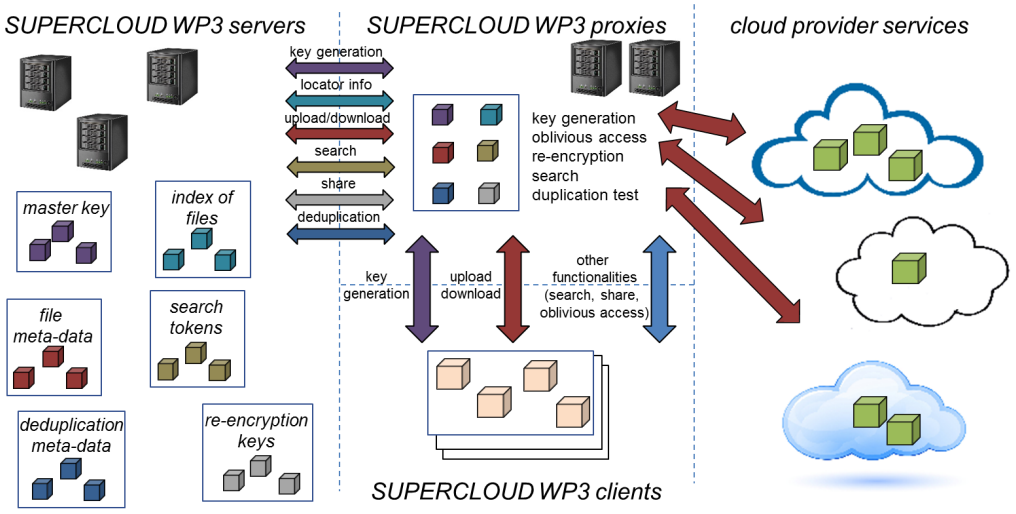


Figure 5.1: Overall architecture for data security.

5.1 Privacy Mechanisms

We first discuss privacy tools that have been introduced in Chapter 3. For each of them, we explain what kind of components should be added to SUPERCLOUD WP3 clients, proxies and servers, and how they interact among each other.

5.1.1 Searchable encryption (SE) architecture

As described in Chapter 3, searchable encryption exists in two settings: the symmetric key setting and the public key setting.

Symmetric case. We first describe the symmetric searchable encryption case. In this case, a client encrypts her data in such a way that she is able to later perform a search on them. This scheme consists in two protocols, $\mathsf{EDBSetup}$ and Search.

- $\mathsf{EDBSetup}(\mathsf{DB}, \mathsf{sk}) \rightarrow \mathsf{EDB}$. This setup procedure is executed by the client. It produces the encrypted database.

- Search. The client takes as inputs a query $\phi(\overline{w})$ and a secret key sk and the server takes as input an encrypted database EDB. After an interactive protocol, the client returns the result of her query.

The client is responsible for the $\mathsf{EDBSetup}$ procedure, and next interacts with the proxy to perform the search. In fact, the encrypted database is in this case divided into two parts: one corresponding to the actual encrypted database EDB as well as a meta-data database $\mathsf{MD_{DB}}$, later used to perform the search.
In SUPERCLOUD, we then propose the following architecture related to symmetric search using the main architecture given in Chapter 4. Within the clients, we add the $\mathsf{EDBSetup}$ functionality. Such client part is also responsible for generating and storing her own secret key sk. Within the proxy, we integrate the search functionality. FInally, within the servers, we store the storage meta-data $\mathsf{MD_{DB}}$ for each database DB. We have then the following interactions during the main procedures.

- **Upload**. After the $\mathsf{EDBSetup}$ procedure, the client sends the whole encrypted database $(\mathsf{EDB}, \mathsf{MD_{DB}})$ to the SUPERCLOUD proxy which sends EDB to the relevant cloud storage provider and stores the meta-data $\mathsf{MD_{DB}}$ in the SUPERCLOUD WP3 server.
- **Search**. Upon reception of a query, the SUPERCLOUD WP3 proxy interacts with the cloud storage providers and the SUPERCLOUD WP3 server to retrieve the database and give the relevant information to the client to perform the search.

Asymmetric case. We now describe the public key encryption with keyword search (PEKS) architecture. It allows to publicly search encrypted data for keywords from a given set W. A PEKS scheme consists of the following four procedures.

- $\mathsf{KeyGen}(\lambda) \rightarrow (\mathsf{pk}, \mathsf{sk})$. This procedure is executed by clients and outputs a pair of secret and public keys.
- $\mathsf{Enc}(\mathsf{pk}, M) \rightarrow C$: produces an encryption of the plaintext M.
- $\mathsf{TrapDoor}(\mathsf{sk}, w) \rightarrow T_w$. For every word $w \in W$, the client generates a trapdoor T_w with her secret key.
- $\mathsf{Test}(\mathsf{pk}, C, T_w) \rightarrow \{0, 1\}$. Given the public key, a ciphertext and a trapdoor T_w, returns 1 if $M = w$ and 0 otherwise.

The main problem we have now to deal with is that data stored in a cloud storage provider is *a priori* not suitable for public key cryptography, essentially because of its size. But the way to manage big-sized data with public key cryptography is well known in the literature and it is most of the time enough to use encapsulation techniques. The following steps should then be executed at each encryption procedure:

1. generate *e.g.* an AES key K_f;
2. encrypt the file f using the AES algorithm and the key K_f as $C_f = \mathsf{AES}(f, \mathsf{K}_f)$;
3. encrypt the secret key K_f using the Enc encryption procedure $\mathsf{MD}_f = \mathsf{Enc}(\mathsf{pk}, \mathsf{K}_f)$.

The ciphertext C_f is then the true encrypted file, while MD_f corresponds to some meta-data associated to the encrypted file. We consider that these meta-data also contain the identity of the client uploading the file. In SUPERCLOUD, we then propose the following architecture related to public key search using the main architecture given in Chapter 4. The overview is sketched in Figure 5.1. Within the SUPERCLOUD WP3 clients, we add the KeyGen, Enc and $\mathsf{TrapDoor}$ functionalities. Such client part is also responsible for generating and storing her key pair $(\mathsf{pk}, \mathsf{sk})$. We integrate the testing functionality Test in the proxy and the servers embed the search tokens, *i.e.* the trapdoors T_w, as well as the storage metadata MD. We then have the following interactions during the main procedures.

- **Upload**. During the upload, the SUPERCLOUD WP3 client executes the encryption functionality Enc. It outputs both C_f and MD_f. They are then sent to the SUPERCLOUD WP3 proxy which then sends C_f to the relevant cloud storage provider and MD_f to the SUPERCLOUD WP3 server for storage.
- **Trapdoors generation**. When a client generates a trapdoor T_w for a word w, she sends it to the SUPERCLOUD WP3 server for storage.
- **Search**. The SUPERCLOUD WP3 proxy can upload a search token and a file to perform a keyword search on the file. According to the result, the file can for example be rerouted to a different cloud storage provider.

There are in fact, in the literature, several types of properties related to the kind of search one wants to perform. In SUPERCLOUD, we will try to match the most relevant searchable encryption scheme to the real needs provided by use cases, keeping in mind that the whole system should match the requested performances.

5.1.2 Secure multi-party computation (SMC)

Most practical MPC approaches are based on secret sharing and basically consist of two phases: the *splitting* phase of the input (secrets) and the *reconstruction* phase of the secret, based on the given shares.
The main process for secret-sharing based MPC is given in Figure 5.2. There are then mainly two principle phases.

- The **computation phase** in which each participant of the computation of the function f receives an output (the output represents a share respective to the provided input). Within MPC, the given secret will be split by means of Adi Shamir's polynomial so the key advantage over secret sharing is the number of honest parties. There are at least $\lceil n/2 \rceil$ honest parties needed to reconstruct the whole secret. Such phase is mainly divided into the following steps:
  1. during a Setup, the parties agree on the set of participants as well as the key/prime necessary for the computation of the function;
  2. after the Setup, each party P_i provides the shares associated to its input (secret) (for this purpose, each party has to choose random values uniformly);
  3. then, the parties P_i participate in interactive protocols to evaluate the function f on the shared values;
  4. afterwards, each party/participant distributes his own shares to the other parties (sharing).
- The **reconstruction phase** is handled by the Lagrange interpolating polynomial which makes the computation of the secret possible. The most important result of the Lagrange interpolation is the recombination vector. This vector is necessary for a correct re-computation of the secret. Such phase is mainly divided into the following steps:
  1. provide shares of at least $\lceil n/2 \rceil$ honest parties/participants;
  2. compute the recombination vector based on the Lagrange interpolating polynomial;
  3. compute with the help of the recombination vector and Shamir's polynomial the secret, which belongs to the shares.

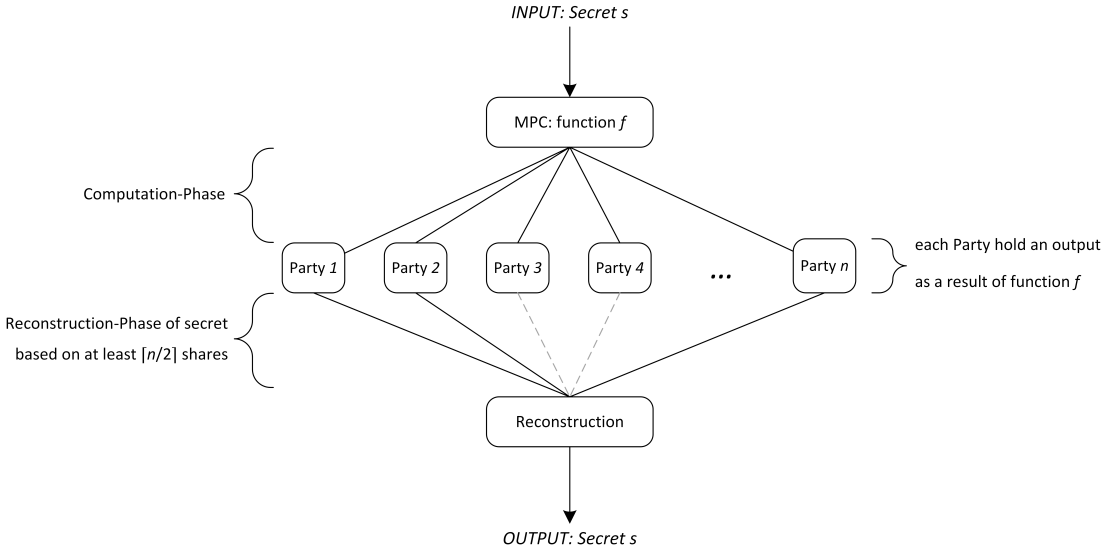


Figure 5.2: Graphical representation of the workflow of secure multi-party computation

5.1.2.1 Architecture Approach

In practical cases, different data sources are available, stored at different (trusted or untrusted) cloud provider services. If data has to be processed within the SUPERCLOUD, the data inputter/accessor uploads the data over the SUPERCLOUD proxies. The main advantage of these proxies is that it becomes possible to make use of the available cryptographic tools, such as the secure multi-party computation. As opposed to the other security techniques discussed in this section, secure multiparty computation employs multiple communicating proxies to enable processing of sensitive data without any individual proxy having to see the data. We then obtain the general architecture for SMC that is sketched in Figure 5.3. In this architecture, MPC is used for anonymizing data with k-anonymity (Section 5.1.3.1); or for answering queries from multiple datasets, possibly with verifiable computation (Section 5.1.2.2) or differential privacy (Section 5.1.3.2). Notice that, apart from the multiple communicating proxies, this architecture coincides with the general WP3 architecture. (The verification key and signatures shown in the architecture are used in verifiable computation, as discussed next.)

5.1.2.2 Verifiable Computation

Multi-party computation addresses the problem that, when querying multiple datasets, it is often not possible to collect all data in a single location – whether it is the data provider or the client. This problem is addressed by outsourcing the computation to separate proxies. However, by outsourcing the computation to an external party, a new problem is introduced, which is how to guarantee that the query result is correct.

Our solution for querying multiple datasets based on multi-party computation (MPC) and verifiable computation addresses the correctness issue by letting the proxies generate a cryptographic proof of correctness of the computation result. Specifically, we let every data provider provide a "signature" over the data available for querying (shown as "signatures" in the figure) that is stored by a trusted SUPERCLOUD server (Figure 5.3).

To implement this architecture, most of state-of-the-art cryptographic constructions from the area of "verifiable computation" can be used, such as replication (the client sends identical information to multiple independent servers) or the use of proofs (performed by the servers to convince the client that they have correctly behaved). Apart from the data providers providing signatures, these constructions also typically assume a trusted third party that performs a one-time setup of a verification key specific to the function f to be calculated. Therefore, our architecture includes this trusted third party as an

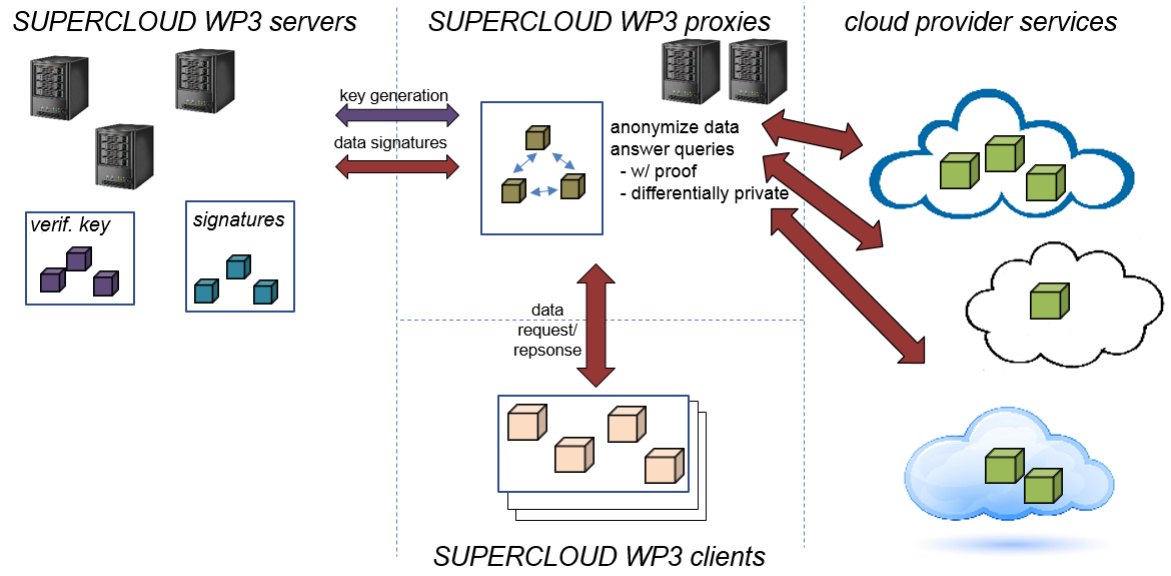


Figure 5.3: Multiparty computation-based security in the WP3 architecture

additional server in the system (shown in Figure 5.3 as "verif. key").
Beyond verifiable computation, we also use MPC in context of data anonymization, described in the following subsection.

5.1.3 Anonymization

We now focus on the way anonymization can be treated inside the SUPERCLOUD architecture. As in Chapter 3, we distinguish the way to treat k-anonymity and differential privacy, that will be the two kinds of anonymization techniques we will use.

5.1.3.1 k-Anonymity

We first focus on k-anonymity. In our WP3 architecture proposal, k-anonymity is used for the evaluation of medical data which are used in the different use cases in the SUPERCLOUD project. Today, there are mainly two ways to store data: databases stored locally and data sources stored in one or more trusted clouds on which client provider observes who accesses the saved information. In most cases, a client accesses a web application to download an anonymized medical data for statistics. Databases with quasi identifiers and data rows may lead to performance issues, specifically because a client can select the degree of anonymization. The lower the granularity stage is chosen, the more computation power is needed. A way to improve this is to outsource the computation part to the SUPERCLOUD servers. One of the biggest disadvantages regarding this architecture proposal is that databases need to remain in plain in the cloud or the local storage. That makes it easy for an adversary to attack a cloud and gain sensitive information. Multi-Party Computation (see Section 5.1.2) provides a remedy.
In fact, the combination of MPC with k-anonymity prevents from storing data in plaintext inside one of the SUPERCLOUD servers. The data has to be computed by means of the function of secure multi-party computation before being stored within the SUPERCLOUD. If any data are needed, such as for statistical purposes (for a read-only recipient), a computation over the data can be done as well as the data anonymization (k-anonymity). More precisely, when the shares of the participants are re-computed again, *e.g.* for a medical statistic, the anonymization of the records will take directly part after to avoid that an adversary sees a plain record. This anonymization is performed at the proxies, as shown in Figure 5.3.

5.1.3.2 Differential Privacy

The SUPERCLOUD architecture also enables combining multi-party techniques for hiding sensitive query responses from proxies with differential privacy techniques for hiding sensitive query responses from the client. Above, we how to use techniques based on k-anonymity to remove identifiers from disclosed dataset. Here, we address the issue that the answer to particular queries on data may leak too much information about an individual.
As discussed in Section 3.1.4.2, differential privacy addresses leakage from query answers by adding enough noise to prevent conclusions about any individual contribution to the dataset. Hence, the workflow in a traditional architecture with differential privacy (without taking privacy in to account), is to first perform a query and calculate the result, and then, depending on the type of query sample randomness according to a particular statistical distribution, add this to the result, and return it. Translated this workflow to the architecture for answering queries from multiple datasets using multi-party computation (Figure 5.3), we now not only evaluate the query answering algorithm using a distributed algorithm; but also perform the noise sampling and addition algorithm in a distributed way. Hence, Figure 5.3 applies, where the proxy apart from answering the query additionally performs this "anonymization".

5.1.4 Oblivious storage

The SUPERCLOUD architecture also providing the option of storage without revealing access patterns. As discussed, the contents of files can be hidden with respect to the storage provider using encryption techniques. Specifically, each file can be encrypted using an ephemeral encryption key, to which a key server maintains access using policies. (Alternatively, attribute-based encryption can be used to cryptographically enforce these access control policies.) However, the storage provider can still observe exactly which files are accessed by which party. In case these access patterns are sensitive, we also provide storage which hides this information.
To hide access patterns, we employ oblivious RAM [88] techniques. At a high level, these techniques hide the actual access pattern of a client by touching multiple files in each access in an elaborate way to completely hide which file is actually accessed. In particular, the "Path ORAM" technique from [161] gives a simple and efficient algorithm that achieves this.
In the most basic form of oblivious storage, a single client employs a single untrusted storage facility. This scenario can be implemented without changing the SUPERCLOUD architecture or the functionality of the storage component. Indeed, as shown in [161], in this scenario the client downloads a number of files, re-encrypts them, and uploads them again, using the normal storage functionality. However, the SUPERCLOUD architecture may be leveraged for performance improvements in two different ways. First, since oblivious access requires significant computational resources at the side of the client, the client (especially when resource-constrained) may outsource this work to a SUPERCLOUD proxy. Second, the system of [161] requires the client to store a "location index" of the files at the server, and keep a small cache of files still to be uploaded. By moving these tasks to a trusted SUPERCLOUD server (the "index of files" in the architecture), the same client can use several client devices to access the data.
When multiple clients need access to the same set of files, several solutions are available. Solutions in the literature [91, 132] allow hiding access patterns from eavesdropping and malicious storage facilities, respectively; but assume all clients have access to all files. A recent work [108] proposed to encrypt files inside the oblivious storage using attribute-based encryption. Note that all of these works aim to hide the access pattern of the clients from the storage provider, but not from other clients. However, hiding access patterns from other clients may actually be a bigger privacy concern. If we instead assume that decryption keys for the individual files are managed by a trusted SUPERCLOUD key server (the "index of files", as discussed above and shown in Figure 5.1), then there is an obvious way of hiding access patterns both from the storage and from other clients. Namely, the key server can perform the ORAM access with the storage, and provides the results to clients. This hides the

files and access patterns with respect to the storage component and other clients, but places a heavy burden on the key server in terms of computation and communication. It remains an open question if this burden on the key server can be reduced while still hiding access patterns from storage and other clients.
Finally, we mention that multi-cloud ORAM systems exist in the literature (e.g., [159]). In such systems, the amount of work performed by the client is reduced by letting two (or more) non-colluding storage providers apply techniques to hide information from the other. Note that, in this case, the storage servers need to support specific ORAM functionality (in the basic set-up above, they do not need to be aware of ORAM); and it is an open question how these techniques can be combined with the multi-client techniques discussed in the previous paragraph. Moreover, since we can already outsource the heavy client work to proxies, we do not consider these multi-cloud systems in the SUPERCLOUD architecture.

5.2 Security of sharing

We now study the cryptographic tools, introduced in Chapter 3, permitting to share stored data in a secure way. Again, we explain what kind of components should be added to SUPERCLOUD WP3 clients, proxies and servers, and how they interact all each other.

5.2.1 Proxy re-encryption (PRE)

A proxy re-encryption scheme can be either unidirectional or bidirectional and either single-hop or multi-hop, as explained in Chapter 3. We here focus on *single-hop unidirectional* proxy re-encryption (UPRE) as it is the most relevant one in our context. Moreover, the architecture that is discussed in this case in this chapter is also relevant for other kinds of proxy re-encryption.
Formally speaking, there are two kinds of actors in a UPRE scheme: clients and a third party (usually called a "proxy"). Moreover, such tool is composed of the following procedures.

- $\mathsf{Setup}(\kappa) \rightarrow \mathsf{param}$: this setup algorithm is executed by any trusted entity. It takes a security parameter $\kappa \in \mathbb{N}$ as input and produces a set of public parameters param shared by all parties.
- $\mathsf{KeyGen}(\mathsf{param}) \rightarrow (\mathsf{sk}, \mathsf{pk})$: this key generation algorithm, whose inputs are the public parameters, outputs a pair of secret and public keys $(\mathsf{sk}, \mathsf{pk})$, and is executed by every client.
- $\mathsf{ReKeyGen}(\mathsf{param}, \mathsf{sk}_A, \mathsf{pk}_B) \rightarrow R_{A \rightarrow B}$: given the public parameters, the secret key of the client A, the public key of the client B, this algorithm, executed by clients, produces a re-encryption key $\mathsf{rk}_{A \rightarrow B}$ which allows to transform second level ciphertexts intended to A into first level ciphertexts for B.
- $\mathsf{Enc}(\mathsf{param}, \mathsf{pk}, m) \rightarrow C$: this second level encryption algorithm, executed by any client, takes param, a public key, a message m and produces a second level ciphertext C that can be re-encrypted.
- $\mathsf{ReEnc}(\mathsf{param}, \mathsf{rk}_{A \rightarrow B}, C) \rightarrow C' / \perp$: this algorithm, executed by the third party, takes as input the public parameters, a re-encryption key $\mathsf{rk}_{A \rightarrow B}$ and a second level ciphertext intended to client A. The output is a first level ciphertext C' re-encrypted for client B that cannot be re-encrypted or an invalid message $\perp$.
- $\mathsf{Dec}_1(\mathsf{param}, \mathsf{sk}, C) \rightarrow m / \perp$: this first level decryption algorithm takes as input param, a client secret key and a first level ciphertext and outputs a plaintext m or an invalid message $\perp$.
- $\mathsf{Dec}_2(\mathsf{param}, \mathsf{sk}, C) \rightarrow m / \perp$: this second level decryption algorithm takes as input the public parameters, a client secret key and a second level ciphertext and outputs a plaintext m (or $\perp$) for the client.

To summarize, most of the procedures are executed by clients, except the Setup one which can be executed by any trusted entity at the creation of the system, and the ReEnc procedure which is executed by the "proxy". Regarding the parameters and keys that are manipulated, most of the keys are only manipulated by clients, except the re-encryption keys $\mathsf{rk}_{A\to B}$ that is generated by the client, but used by the "proxy".
Regarding the client, the key generation procedure is executed at the client and device registration. Each file upload is associated to the executed of the encryption algorithm Enc. At each new file sharing, the client should create a new re-encryption key, using ReKeyGen. Finally, a file download should use one of the two decryption procedures Dec_1 or Dec_2.
As for public key encryption with keyword search, we have to deal with files with potentially big sizes. Again, we can use the encapsulation techniques with the same steps as previously, which output both the ciphertext C_f corresponding to the true encrypted file, and the meta-data MD_f data associated to the encrypted file.
In SUPERCLOUD, we then propose the following architecture related to data sharing by using proxy re-encryption, using the main architecture given in Chapter 4. The main view is sketched in Figure 5.1. We add to the client the KeyGen, ReKeyGen, (AES,Enc), Dec_1 and Dec_2 functionalities. Such client part also stores the client's dedicated key pair $(\mathsf{sk}_A, \mathsf{pk}_A)$. We integrate to the proxy the re-encryption functionality ReEnc. The servers finally store three kinds of data: (i) the storage meta-data MD_f for each uploaded file f, (ii) the re-encryption keys $\mathsf{rk}_{A\to B}$ for each existing sharing between two clients and (iii) optionally all client's public keys that can be requested during a sharing procedure.
We have then the following interactions during the main procedures.

- **Upload**. During the upload, the SUPERCLOUD WP3 client executes the encryption functionality (AES,Enc) which outputs both C_f and MD_f. Both are then sent to the SUPERCLOUD WP3 proxy which then sends C_f to the relevant cloud storage provider and MD_f to the SUPERCLOUD WP3 server for storage.

- **Sharing**. During a sharing from one client A to another client B, client A may first request the SUPERCLOUD WP3 server, throw the SUPERCLOUD WP3 proxy, to obtain client B's public key. It then executes the re-encryption key generation functionality ReKeyGen and the result $\mathsf{rk}_{A\to B}$, together with the new sharing information, is sent to the SUPERCLOUD WP proxy. The new sharing information is sent to the relevant cloud storage provider and $\mathsf{rk}_{A\to B}$ is sent to the SUPERCLOUD WP3 server for storage.

- **Download**. During a download, the client B first requests the encrypted file to the SUPERCLOUD WP3 proxy who keep back (i) from the cloud storage provider the encrypted file C_f and (ii) from the SUPERCLOUD WP3 server the meta-data MD_f associated to the file f. The SUPERCLOUD WP3 proxy first checks whether the requesting client B corresponds or not to the identity A embedded into the meta-data MD_f. If this is the case, both C_f and MD_f are directly sent to the client, who can execute the decryption procedure (using Dec_2). If not, the SUPERCLOUD WP3 proxy requests the SUPERCLOUD WP3 server for the existence of a re-encryption key $\mathsf{rk}_{A\to B}$. If not, the process is stopped. Otherwise, the SUPERCLOUD WP3 proxy executes the re-encryption procedure ReEnc on input MD_f and $\mathsf{rk}_{A\to B}$ and sends the result to the SUPERCLOUD WP3 client, together with C_f, who can execute the decryption procedure (using Dec_1).

In SUPERCLOUD, we will try to manage all kinds of data storage with such cryptographic tool: object storage, block storage, but also file system. We argue that the way to share data in each case can be different, and that we need to adapt PRE to each of them.

5.2.2 Attribute-based encryption (ABE)

As said in Chapter 3, an attribute based encryption (ABE) scheme can be *key-policy* or *ciphertext-policy*. We here focus on the second kind of ABE since, as shown in Chapter 2 on use case requirements,

we should focus on role-based access control, that is the case where clients are given attributes that may permit them to obtain access to protected data. As for proxy re-encryption, the architecture we give below is most of the time also relevant for other kinds of ABE.
Formally speaking, there are two kinds of actors in a ABE scheme: a key manager and clients. Moreover, such tool is composed of the following procedures.

- $\mathsf{Setup}(\kappa) \rightarrow (\mathsf{param}, \mathsf{msk}, \mathsf{ek}, \mathcal{B})$: it takes as input the security parameter κ and generates the global parameters param of the system, a master secret key msk, an encryption key ek (that can be either private or public) and the set $\mathcal{B}$ of all possible client attributes. It is executed by the key manager.
- $\mathsf{KeyGen}(\mathsf{param}, \mathsf{msk}, u) \rightarrow (\mathsf{dk}_u)$: it takes as input a client u together with his set of attributes $\mathcal{B}(u) \subset \mathcal{B}$. It also takes param and msk, and outputs the client's decryption key dk_u. Such procedure is executed for each client, by the key manager.
- $\mathsf{Enc}(\mathsf{param}, \mathsf{ek}, m, \mathbb{A}) \rightarrow C$: it takes as input the encryption key ek, a message m and an access policy $\mathbb{A}$. It outputs the ciphertext C related to m, and a header Hdr which includes the access policy $\mathbb{A}$. It is executed by client.
- $\mathsf{Dec}(\mathsf{param}, \mathsf{Hdr}, \mathsf{dk}_u, \mathcal{B}(u)) \rightarrow m/\perp$: such procedure, executed by clients, takes as input the header Hdr, the decryption key dk_u of a client u with attributes $\mathcal{B}(u)$ and outputs the plaintext m if and only if $\mathcal{B}(u)$ satisfies $\mathbb{A}$. Otherwise, it outputs $\perp$.

The key manager is responsible for the Setup and the KeyGen procedures, and should manage msk, while clients execute the encryption Enc (during upload/sharing) and decryption Dec (during download) procedures, and manipulate the attributes in $\mathcal{B}$, the access policies $\mathbb{A}$, and their own private key dk_u. We note here that the upload and sharing functionalities are here put in one action (contrary to proxy re-encryption). Each upload is again associated to a ciphertext C_f and some related meta-data MD_f. The main problem we have to face off is that, using the above description, the key manager knowing the master secret key msk obtain the decryption key of all clients. To mitigate this risk, we add the possibility to split the role of the key manager into several distinct parts, such that (i) the master secret key msk is better protected and (ii) no part can obtain the exact client decryption key dk_u. This gives us the following distribution between architecture components (see Figure 5.1 for a graphical view). Within the clients, we add the Enc and Dec functionalities. Such client part also stores the client's private key dk_u and the set $\mathcal{B}(u)$ of attributes. The proxyis responsible for the interaction between the clients, the servers and the cloud storage provider. Finally, the master secret key can be distributed between multiple servers. Within these servers, we store three kinds of data: (i) the set $\mathcal{B}$ of all possible attributes, (ii) the storage meta-data MD_f for each uploaded file f and (iii) the master secret key msk, possibly split into several independent parts.
The main interaction, in this case, are as follows.

- **Attribute addition**. When a new instance of a SUPERCLOUD WP3 client is created, or when the attributes related to an existing instance have to be modified (added or deleted), a request is sent by the SUPERCLOUD WP3 client to several different instances of the SUPERCLOUD WP3 proxy. The SUPERCLOUD WP3 proxy instances then request the SUPERCLOUD WP3 servers to obtain each an independent part of the master secret key. Each instance then executes the KeyGen procedure with part of msk and all the results are sent to the SUPERCLOUD WP3 client who can compute, with each share, his client secret key dk_u.
- **Upload**. During the upload, the SUPERCLOUD WP3 client executes the encryption functionality Enc by choosing an access policy $\mathbb{A}$ of the file he wants to upload. Such functionality outputs both C_f and MD_f. Both are then sent to the SUPERCLOUD WP3 proxy which then sends C_f to the relevant cloud storage provider and MD_f to the SUPERCLOUD WP3 server for storage.

- **Download**. During a download, the SUPERCLOUD WP3 client requests the encrypted file to the SUPERCLOUD WP3 proxy who keeps back (i) from the cloud storage provider the encrypted file C_f and (ii) from the SUPERCLOUD WP3 server the meta-data MD_f associated to the file f. Both are sent to the SUPERCLOUD WP3 client who executes the decryption procedure, using Dec and the client private key dk_u.

Within SUPERCLOUD, additionally to find a secure and efficient way to manage multiple SUPERCLOUD WP3 servers, we will study the way to apply such cryptographic tool to real role-based access control defined by use cases. The resulting performances will be here an important challenge.

5.2.2.1 Extension to multi-authority attribute-based encryption

As discussed in Section 3.2.2, so-called "multi-authority" attribute-based encryption [62, 63] allow to delegate the issuing of particular attributes to so-called "attribute authorities" in a secure way. Our architecture can support the multi-authority case by, in addition to SUPERCLOUD WP3 server(s) storing the master secret key (cf. Figure 5.1), also having SUPERCLOUD WP3 server(s) storing "attribute keys", that can then in turn be distributed to further reduce risks in key management. Note that typically different attribute authorities would be operated by different organisations, aligning well with other multi-organization parts of the SUPERCLOUD architecture such as application-level authorization.

5.2.3 Secure deduplication

We here focus on the convergent encryption case as it is the only cryptographic protocol for deduplication which is used in practice. A convergent encryption scheme consists in the following protocols.

- $\mathsf{KeyGen}(\lambda, m) \rightarrow \mathsf{sk}_M = H(m)$. This procedure is executed by clients and outputs the secret key which is a deterministic hash of the message.
- $\mathsf{Enc}(M, \mathsf{sk}_M) \rightarrow C = \mathrm{AES}_{ENC}(M, \mathsf{sk}_M))$. Users encrypt their messages.
- $\mathsf{Test}(C_1, C_2) \rightarrow \{0, 1\}$. Returns 1 if $C_1 = C_2$ and 0 otherwise.
- $\mathsf{Dec}(C, \mathsf{sk}_M) \rightarrow C = \mathrm{AES}_{DEC}(C, \mathsf{sk}_M)$. Decryption is performed with the AES decryption procedure.

In SUPERCLOUD, we then propose the following architecture related to deduplication, using the main architecture given in Chapter 4. The overview is also sketched in Figure 5.1. Within the SUPERCLOUD WP3 clients, we add the KeyGen, Enc and Dec functionalities. Such client part also stores the client's private key sk_M in relation with each message M stored in the cloud provider. An encryption of each key can also be sent into the server. The testing functionality is integrated into the proxy. We finally store the deduplication meta-data, namely a tag for each ciphertext, in the servers. In convergent encryption, such a tag is a deterministic hash of the ciphertext.
We then have the following interactions during the main procedures.

- **Upload**. During the upload, the SUPERCLOUD WP3 client executes the encryption functionality Enc. Such functionality outputs C and the deduplication meta-data. Both are then sent to the SUPERCLOUD WP3 proxy which then sends C to the relevant cloud storage provider and the deduplication meta-data to the SUPERCLOUD WP3 server for storage.
- **Test**. Upon reception of a new ciphertext with deduplication meta-data, the SUPERCLOUD WP3 proxy requests the deduplication meta-data to the SUPERCLOUD WP3 server and performs the Test procedure. If the procedure returns 1, the SUPERCLOUD proxy adds the information about the new client to the storage meta-data and interacts with cloud service providers for the deletion of the new file. If the procedure returns 0, the SUPERCLOUD proxy proceeds with the file storage.

In SUPERCLOUD, we will try to work on the way to provide at the same time confidential data sharing, replication (to fight against fault attacks), and deduplication. The way the cryptographic tools dedicated to each need can interoperate is here an interesting problem we will have to face to.

Conclusion. Regarding data security, our main purpose is, in conclusion, to protect individuals' data while still permitting their manipulation. This is quite important when looking at use case requirements, where client's data should be protected but attainable, where the privacy of medical data is a top priority, and where special permissions to access data outside their context should be feasible.
The main problems we will manage are (i) to obtain interoperable solutions (how to have at the same time *e.g.* search, sharing and deduplication in a single system) and (ii) to take care of the resulting performances, so as to match requirements on real-time data access.

Chapter 6 Dependability and Data Integrity Architecture

In this Chapter we detail the SUPERCLOUD architecture of storage and data management dependability features, that maps to the high-level logical architecture, given in Section 4.1.
The features covered in this Chapter shall enable SUPERCLOUD high-availability, fault and disaster tolerance, data integrity and consistency and geo-replication capabilities. To this end, some of the key techniques we will use are quorum-based replication, erasure coding, secret-sharing and state machine replication (SMR). We approach these classical techniques in a novel way, necessary for our use cases and requirements, and re-implement them in an efficient, secure and scalable way.

6.1 Storage server geo-replication

Tolerating any kind of service disruption, whether caused by a simple hardware fault or by a large-scale disaster, is key for the survival of modern distributed systems at cloud-scale and has direct implications on the business behind them [113]. Modern systems today increase the number of *nines of reliability* (i.e., availability and data integrity) by employing sophisticated distributed protocols that tolerate machine faults as well as network faults such as *network partitions*, or *asynchrony*, which reflects the inability of *correct* machines to communicate among each other in a timely manner. At the heart of these systems typically lies a consensus-based *state machine replication* (SMR) protocol [152, 60].
In SUPERCLOUD, we will primarily use SMR to make storage servers robust. Specifically, storage servers in SUPERCLOUD shall be used to maintain a critical portion of system metadata in a consistent and highly available way, so that the resilience of the rest of the system can be based on it. SUPERCLOUD server replication using SMR shall be modular and applicable to different deployment scenarios; namely, we will support different server fault models including both crash/recovery and Byzantine faults, as well as tolerate network faults. Particular focus will be on tailoring our SMR solutions to geo-replicated context of cloud-of-clouds.
Our work in this direction will focus on four main novel approaches:

- Revisit the correlation between server faults and network faults;
- Consider the diverse performance of replicas and the clouds hosting them in the operation of the replication protocols;
- Optimize the number of leaders in a geo-replicated SMR system;
- Support elastic and self-managed SMR.

We outline these four directions in the remaining of this section.

6.1.1 Decoupling Byzantine server from network faults

Existing SMR-based systems cannot efficiently deal with Byzantine faults despite more than 30 years of intensive research in distributed computing since the seminal work of Lamport, Shostak and Pease [117], no *practical* answer to tolerating non-crash faults has emerged yet. In particular,

Byzantine fault-tolerance (BFT) has not lived up to expectations, due to its large cost compared to crash fault-tolerant (CFT) solutions. For example, in the context of asynchronous (that is, eventually synchronous [79]) BFT SMR, this overhead implies using at least $3t+1$ replicas to tolerate t non-crash faults (instead of $2t+1$ in the case of CFT).
The overhead of BFT is due to the extraordinary power given to the adversary, which may control both the faulty machines *and* the entire network in a coordinated way. In line with observations by practitioners [114], we claim that this adversary model is actually too strong for the phenomena observed in deployed systems: The Byzantine, non-crash faults experienced today occur independently of network faults (and often also independently of each other), just like a network switch failing due to a soft error has no correlation with a server that was misconfigured by an operator. The proverbial all-powerful attacker as a common source behind those faults is a popular and powerful simplification used for the design phase, but it has not seen equivalent proliferation in practice. Surprisingly, though, the reliability models that decouple network faults (i.e., asynchrony, or untimely communication) from machine faults (yet allow for both classes of faults) have not been adequately studied.
More concretely, in the context of SUPERCLOUD, we do not necessarily expect an untrusted cloud provider to be able to corrupt communication of correct users with honest clouds, or the communication among honest clouds themselves. Whereas this communication can fail and network faults can occur, these are not realistically under control of an untrusted cloud provider.
In our novel approach that we will pursue in SUPERCLOUD for storage server replication (besides supporting traditional CFT and BFT solutions) we will target protocols in the *XFT* (short for *cross fault tolerance*), model. The XFT model, that we introduced conceptually and recently in [123], decouples the fault space *across* machine and network faults dimensions, treating machine and network faults independently. In a nutshell, XFT models Byzantine and network faults as independent events, which is reasonable for many practical applications including the use cases and deployment models targeted by SUPERCLOUD.
As an illustration, the XFT approach allows to build systems tolerate a certain class of faults (e.g., crash faults) regardless of network faults (asynchrony), and another class of faults (e.g., non-crash faults) only when there are no network faults (i.e., when the system is synchronous). This allows for the development of reliable systems that have the cost of asynchronous CFT (already paid for in practice), yet with decisively better reliability than CFT and sometimes even better reliability than fully asynchronous BFT itself.
Preliminary results on XFT SMR in the geo-replicated context are encouraging and show performance and cost similar to CFT SMR for roughly the same cost with considerably more nines of reliability and the ability to tolerate Byzantine server faults. Surprisingly, our preliminary analysis shows that XFT sometimes (depending on the system environment) offers even *strictly stronger* reliability guarantees than state-of-the-art BFT SMR protocols.
In SUPERCLOUD we will implement support for XFT, in addition to that for classical CFT and BFT to storage server replication in the data management layer. One of the first candidate deployments we will tackle is the metadata server replication in the Ceph distributed file system [3]. Ceph is particularly interesting as a deployment technology as it is the most popular reliable and fault tolerant block storage system backing Openstack cloud deployments.[1] However, today's Ceph is neither optimized for geo-replication nor capable of tolerating Byzantine faults, and hence is not suitable for SUPERCLOUD use cases. That said, our XFT designs will in no way be limited to Ceph and we will explore XFT implementations for other systems (e.g., BFT-SMaRt [41]).

6.1.2 Weight assignment for Geo-replicated State Machines

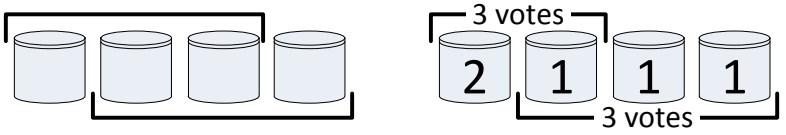
Some works shown that waiting fewer replies from replicas to make progress in distributed protocols (i.e., smaller quorums of replicas) improves the latency of such protocols when deployed in real wide-area networks [158, 105]. Popular protocols like Paxos [116] or RAFT [142] make each replica (and

[1] `http://superuser.openstack.org/articles/openstack-user-survey-insights-november-2014`

in particular, the leader process) wait for answers from a majority of the replica set, i.e., they wait for a majority quorum. However, the aforementioned studies show that waiting for 2 out-of 4 replicas makes the latency of such protocols significantly better than waiting 3 out-of 4 or even 2 out-of 3. Unfortunately, relatively smaller quorums cannot be used in standard SMR systems since they may lead to violations.

One of the improvements we want to make in SMR protocols, within the context of the SUPERCLOUD project, is to make them *rely on the fastest replicas present in the system*, and, at the same time *preserve its original safety and liveness properties*. The most important guarantees that quorum-based protocols need to preserve are (1) all possible quorums overlap in some correct replica and (2) even with up to f failed replicas, there is always some quorum available in the system. As said before, in crash fault tolerant (CFT) protocols like Paxos, quorums must overlap in at least one replica. Such intersection is enforced by accessing a simple majority of replicas during each communication step of a protocol. More specifically, protocols access $\lceil \frac{n+1}{2} \rceil$ replicas out of $n \geq 2f + 1$. Byzantine fault tolerant (BFT) protocols like PBFT [59], on the other hand, usually employ disseminating Byzantine quorums [128] with at least $f + 1$ replicas in the intersection. In this case, protocols access $\lceil \frac{n+f+1}{2} \rceil$ replicas out of $n \geq 3f + 1$. In both these strategies, adding a single extra replica to the system results in potentially higher latencies, as any possible quorum becomes larger in size.

In order to accommodate the two improvements we want to make, we have to deal with this limitation. The fundamental observation we make is that accessing a majority of replicas guarantees the aforementioned intersection, but that this is not the only way to secure such intersection. More specifically, if n is greater than $2f + 1$ (in CFT), it is possible to distribute weights across replicas in such way that a majority is not always required to (correctly) make progress. As an example, consider the quorums illustrated in Figure 6.1 (with one extra replica in the system). Whereas in Figure 6.1(a) the intersection is obtained by strictly accessing a majority of replicas, in Figure 6.1(b) we see that we can still obtain an intersection with a variable number of replicas (since we can obtain a sum of 3 votes by either accessing 2 or 3 replicas). In particular, if the replica with weight 2 is successfully probed, the protocol can finish a communication step with a quorum comprised by only half of the replicas. Otherwise, a quorum comprised by all replicas with weight 1 is necessary to make progress. Notice that for this distribution to be effective, it is necessary to attribute weight 2 to the fastest replica in the system.



(a) Classical (always a majority of replicas).

(b) Weighted (from $f + 1$ to $n - f$ replicas).

Figure 6.1: Quorum formation when $f = 1$ and $n = 4$ (CFT mode).

The weight assignment scheme described above was generalized in a recent paper [158], and will be an important asset for our multi-cloud replicated state machines for supporting data stores.

6.1.3 How many leaders in cloud of clouds?

Cloud-of-clouds systems are geographically distributed among different clouds across the globe. This poses challenges to classical SMR protocols. Take Paxos [116] for example; Paxos has a *single leader* that is responsible for sequencing and proposing clients' requests. These proposed requests are then replicated at least across a majority of replicas, executed in order of their sequence numbers, with application-level replies eventually sent to the clients. For clients residing at a remote site with respect to that of a leader, this may imply costly round-trips across the globe.

In order to reduce latency for geo-replicated applications, several *multi-leader* or *leaderless* SMR protocols have been proposed [77, 129, 135]. In multi-leader SMR [129, 77], a request can be proposed and sequenced by *any* replica, where every replica can act as a leader, typically by partitioning the sequence number space. In these protocols, a client submits its request to the nearest replica, avoiding the communication with a single (and possibly remote) leader.
However, a challenge for a multi-leader SMR is the coordination with distant or slow replicas, which can be the bottleneck, or even block the system. In some sense, the performance is determined by the 'slowest' replica: this causes what is known as a "delayed commit" problem [129]. Roughly speaking, the delayed commit problem is due to the need to confirm that all requests with an earlier sequence number are not "missed". For most typical, imbalanced workloads, e.g., if most requests originate from clients that gravitate to a given cloud S, this incurs communication with *all* replicas including remote and slow ones. In this case, multi-leader SMR may have worse performance than single-leader SMR, in which replication involves only S and the majority of sites closest to S.
On the other side, leaderless protocols (e.g., [135]) do not suffer from the delayed commit problem as they explore the commute property of concurrent requests. In these protocols, a request can be proposed by the client or any replica, and committed with a round-trip latency from a fast-path quorum, which is equal to or larger than a majority. However, if the conflict with some concurrent request is detected, one more round-trip message exchange from a majority is introduced.
Obviously, neither single-leader nor multi-leader (i.e., "all-leader") solution fits all situations. Existing work either assumes that requests are evenly distributed among all replicas (favoring multi-leader SMR), or assumes that requests are largely distributed around one replica (favoring single leader SMR). More than often, neither of these assumptions is true in the geo-replicated context: for instance, due to time zone differences, clients located at a given site may have different access patterns at different times of a day, changing the "popularity" of sites dynamically.
In this context we plan to explore the space between single-leader and multi-leader ("all-leader") SMR, by dynamically reconfiguring the set of leaders. Each set of leaders is selected based on previous workload and network condition. Our preliminary results in this direction [124] are encouraging. This is in particular true when dynamic leader election is coupled with state partitioning and coordination schemes that, in SUPERCLOUD, will be inherent with multi-tenant users. Our preliminary results show that, under typical imbalanced workloads that usually characterize real-world applications, our approach applied to Mencius and Clock-RSM efficiently reduces latency compared to native protocols, and have the similar latency as that of leaderless protocol, e.g., EPaxos.

6.1.4 Elastic State Machine Replication

A critical limitation of the state machine replication approach is its lack of scalability. This limitation comes from three main properties of the model and its practical protocols [152]: (1) the services usually need to be single-threaded to ensure replica determinism, (2) there is normally a leader replica which is the bottleneck of the SMR ordering protocol (as discussed before), and (3) adding more replicas does not improve the system performance. Different techniques have been developed to deal with these limitations. Some works propose SMR implementations that take advantage of multiple cores to execute requests in parallel [110, 94, 107], solving (1). Although effective, the improvements are limited by the number of cores available on servers. In the same way, some unorthodox protocols spread the additional load imposed to the leader among all the system replicas [129, 135]. These protocols solve (2), but scalability remains limited because every replica still needs to execute all the operations.
A recent line of work proposes partitioning of the SMR-based systems in multiple Replicated State Machines (RSMs) [86, 145, 69, 149] for addressing (3). Although partitioning solves the *scalability of SMR*, the existing solutions are quite limited in terms of *elasticity*, i.e., the capacity to dynamically increase (scale-out) and decrease (scale-in) the number of partitions at runtime. More specifically, some scalable systems consider only static partitions [149, 145], with no elasticity at all, while others [86, 69] provide dynamic partitioning through ad-hoc protocols that are executed in background to avoid

performance disruption.
In SUPERCLOUD we want to develop a generic *partition transfer* primitive (and protocol) that enables SMR-based services to be *elastic*, being thus more appropriate for implementing adaptive and self-managed services in the cloud. The idea is to design a protocol to perform transfers efficiently and with minimal perturbations on the client-perceived performance, on top of any existing SMR protocol. Figure 6.2 illustrates three situations in which such elasticity might be beneficial. A group G is used up to its maximum load capacity and, then, the state managed by this group is *split* to another group L to balance the load among the two groups and open room for the system to handle more work. When two groups are processing requests, a non-uniform access pattern might unbalance the load handled by the two groups, as shown in the second case. Here, part of the state of the overloaded group G can be transferred to the underloaded group L to *rebalance* the system. Finally, if the load decreases and one of the partitions becomes underutilized, it is possible to *merge* the state of the two groups in a single RSM, for freeing the underutilized resources.

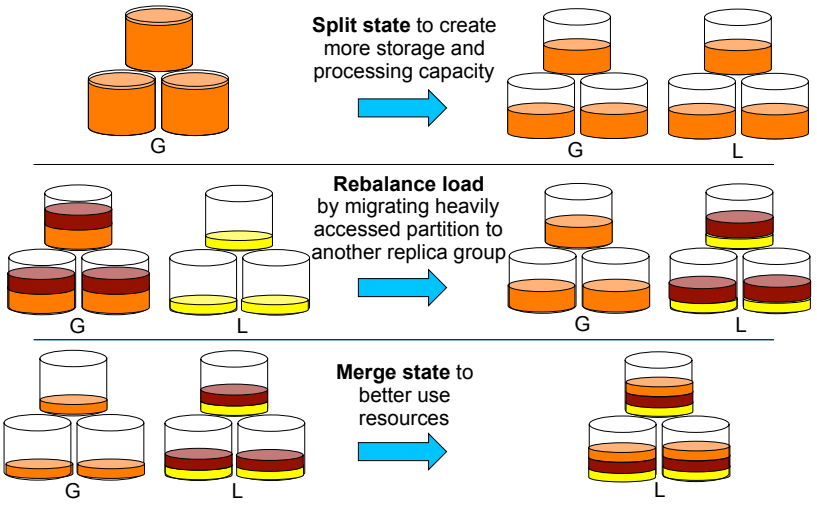


Figure 6.2: Reconfigurations of a partitionable RSM.

An important issue in elastic systems is to define when and how to perform reconfigurations like the ones shown in Figure 6.2. There are many works on dynamic resource configuration managers [112, 138, 84], which decide when and how to adapt according to specified policies. A fundamental parameter used in these systems is the duration required for deploying a new server and make it ready to process requests, i.e., the *setup cost*. A high setup cost always discourages reconfigurations, leading to over-provisioning and increased operational costs [84]. Given that, it is extremely important to reduce the setup cost.
In contrast to a stateless service, where servers start processing requests as soon as they are launched, in a stateful service such as a RSM, a partition transfer must be performed before a server starts processing requests. As a result, the setup cost increases. Therefore, tackling the cost of performing a partition transfer is crucial when stateful services are deployed on cloud systems in order to meet the requirements defined in Service Level Agreements (SLA). Usually, elastic stateful systems rely on distributed cache layers [140], write-offloading [137] and, ultimately, over-provisioning [84]. These techniques are useful to create buffers to either absorb unexpected load spikes or buy time for the stateful backend to reconfigure. We want to challenge this design in SUPERCLOUD, at least for an important class of systems – the ones that employ SMR protocols, and define a principled way for replicated stateful systems to scale-in and -out efficiently, for non-negligible partition sizes and workloads.
We are currently working on the protocol and implementing a prototype in the BFT-SMaRt state machine replication programming library [41].

6.2 Object/block stores and file systems

In complement with our work on reliability of SUPERCLOUD storage server reliability, which we will mostly use to reliably store SUPERCLOUD data management metadata, we will also design novel solutions for geo-replication and reliability of data scattered across clouds. In particular, the solutions outlined in this section describe how we will deal with reliability of *data nodes* and cloud provider services (CPS). The specific directions are outlined below.

6.2.1 Enabling geo-replication for distributed block storage

Currently, popular cloud block storage solutions, e.g., Ceph's RADOS block device (RBD) [4], are not optimized for geo-replication. Whereas adding geo-replicated storage data nodes to RBD is possible, the system does not distinguish remote data nodes from local data nodes. For instance, this may impact performance for a SUPERCLOUD user that wishes to use remote cloud services together with his local storage resources in the hybrid cloud setting.
To rectify this, we will implement mechanisms for volume synchronization between clusters of multi-cloud block storage, with concrete focus on Ceph. The idea here is to start from implementing replication from a block storage volume in the source cluster to its replicated volume in the destination cluster. This would be done at first periodically, by:

- making a snapshot of the source volume/cluster,
- creating a change set from the previously transferred snapshot,
- transferring the change set to the destination volume/cluster,
- applying the change set to previously transferred snapshot in the destination cluster,
- saving this merged snapshot in the destination cluster,
- maintaining snapshots on both clusters.

The implementation will involve:

- *Storage server* responsible for: (1) receiving/responding requests via ReST interface to the client, (2) sending/receiving messages to proxies, (3) querying/updating/maintaining the database;
- *Storage proxy* responsible for: (1) receiving/sending messages to the server, (2) starting the replication process, (3) providing the cluster credentials for the replication processes, (4) running a replication process (one per replicated volume);
- *Storage client* connector to Openstack Cinder and CLI tool.

Implementation would use a database (maintained by the server) for storing current and past proxy information, storing the timestamp to check the proxy status, credentials and statistics data. Communication layer of the components will be implemented over RabbitMQ, which would also be used for CLI tool messages to the server, server messages to proxies and listening OpenStack Cinder messages for automated volume replication start.

6.2.2 User-defined Cloud-backed Storage

In the last years there have been many projects proposing the use of cloud storage services (e.g., Amazon S3, Google Storage) as the backend for files systems [39, 170] and databases [48]. All these services are quite limited in their capability to adapt to the dependability, security and legislation requirements imposed by different applications that use cloud services.
In SUPERCLOUD, we want to build a user-centric/software-defined storage infrastructure that distributes its stored data by different cloud storage services, using different replication, coding and

encryption techniques in accordance with the specifications provided during the virtual volume creation.
This infrastructure builds upon previous works done within the context of FP7 TClouds and BiobankCloud projects [38, 33, 39], and extend it by improving aspects such as configurability, adaptiveness, performance and storage-efficiency. The basic idea is to create two main components: management service and access proxy.
The *management service* will allow users to define the storage policies they want, such as expected latency, storage overhead, security requirements, replication factor and location of data and the system will generate a virtual storage configuration defining how such requirements can be satisfied. This configuration plan will be used by *volume accessors* that can be implemented either as accessors (deployed in the machines using the virtual storage) or proxies (that export an NFS server interface). In both cases, users access the volume through a file system interface provided either by an NFS client installed in their machines (to access a proxy) or a FUSE-based file system client deployed in their machine (the accessor).
There are many opportunities and gaps that we need to solve to implement such user-defined cloud-backed storage in SUPERCLOUD. In the following we discuss the main issues we are addressing:

1. *Efficient cloud-of-clouds replication:* Current cloud-of-clouds dependable data storage algorithms are still not very efficient in terms of storage overhead. If one uses full replication for storing data of size L, the overhead will be $(n-1) \times L$. If storage optimal erasure code are used, such overhead can be as small as desired, but to make it really small one needs to spread the data among plenty of cloud services [21]. This is undesirable because clients will need to access many cloud services to read the data, which will negatively impact the latency of the storage service. We are working on a new algorithm for solving this problem. The key idea of the algorithm is to first write with a 50% storage overhead and then, a posteriori, change the storage allocation to use less storage.

2. *Mapping high-level requirements to algorithms and clouds:* Mapping high-level requirements expressed, for example as 90th latency values or storage budget, requires a deep understanding of the services being "aggregated" and how they can be mapped to the requirements. To this end, we want to design a set of heuristics during the 2nd year of the SUPERCLOUD project.

3. *Scalable proactive secret sharing:* One of the limitations of current cloud-of-clouds data replication algorithms is that large volumes of data need to be transferred for reconfiguring the set of cloud services. Such reconfigurations are required when one of the clouds is removed (due to failures or even a sudden spike in the provider prices). Currently, most algorithms require that clients read the data and then write the data back to the new set of storage clouds. In order to avoid this situation, we need to modify the secret sharing primitive employed in confidentiality-aware replication algorithms (e.g., [38]). We are currently identifying recent progresses in secret sharing and trying to integrate this on the current algorithms we implemented in the previous projects (e.g., TClouds and BiobankCloud).

4. *Support efficient database replication/disaster recovery:* A final (and fundamental) challenge we want to address in the project is how to run existing databases efficiently on top of the SUPERCLOUD cloud-backed storage service. The main advantage of doing this is to implement low-cost disaster recovery for existing applications with no modification in their architectures. Currently we are profiling some popular relational databases (e.g., PostgreSQL) to understand their I/O pattern and map it efficiently to our multi-cloud replication algorithms.

Finally, an important requirement of the system is that it must be compatible with many of the features described in the security architecture. In particular, attribute-based encryption, secure deduplication and oblivious RAMs are three techniques that we envision being used together with our system.

Chapter 7 Conclusions

Summary

This deliverable presented a preliminary architecture for SUPERCLOUD data management infrastructure enabling dependable and secure storage and data protection in cloud-of-clouds.

- We first analyzed the design requirements of the architecture. Besides functional and dependability and security requirements, particular and very important requirements to data management are those related to compliance and non-functional, performance requirements.
- We then surveyed the state of the art of key enabling technologies for security and dependability in cloud data management that provide a context to build our SUPERCLOUD architectures.
- Finally, we presented a preliminary architecture for data management infrastructure, describing the overall design, the architecture of the different components, also showing how the different techniques enable to fulfill the identified requirements.

Next steps

Next steps will be to implement and evaluate the proposed architecture and its components. This notably includes refining the relation with the use cases, and the other architectures for virtualization of computation (WP2), networking (WP4), and overall design (WP1). During this phase, the platform architecture will be continuously re-evaluated and improved, e.g., refining the description of some components of this preliminary version of the architecture.

Bibliography

[1] Barbican/discussion-federated-barbican. https://wiki.openstack.org/wiki/Barbican/Discussion-Federated-Barbican.

[2] Boxcryptor. https://www.boxcryptor.com/.

[3] Ceph. http://ceph.com/.

[4] ceph rados block device.

[5] CipherCloud. https://www.ciphercloud.com/.

[6] Data Masker. https://www.datamasker.com/.

[7] IMB Optim. https://www.ibm.com/software/data/optim/.

[8] Informatica Data Privacy. https://www.informatica.com/products/data-security/data-masking.html.

[9] MEGA. http://www.mega.com/.

[10] MultCloud. https://www.multcloud.com/.

[11] Nasuni UniFS. http://www.nasuni.com/.

[12] OpenStack Mercador project. https://wiki.openstack.org/wiki/Mercador.

[13] Oracle Data Masking Pack. http://www.oracle.com/us/products/database/data-masking-subsetting/overview/index.html.

[14] Otixo. https://www.otixo.com/.

[15] Panzura CloudFS. http://panzura.com/.

[16] Perspecsys. https://perspecsys.com/.

[17] IBM IDentity Mixer, 2015.

[18] Martín Abadi, Dan Boneh, Ilya Mironov, Ananth Raghunathan, and Gil Segev. Message-locked encryption for lock-dependent messages. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, pages 374–391, 2013.

[19] Michel Abdalla, Mihir Bellare, Dario Catalano, Eike Kiltz, Tadayoshi Kohno, Tanja Lange, John Malone-Lee, Gregory Neven, Pascal Paillier, and Haixia Shi. Searchable encryption revisited: Consistency properties, relation to anonymous ibe, and extensions. In *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, pages 205–222, 2005.

[20] Ittai Abraham, Gregory Chockler, Idit Keidar, and Dahlia Malkhi. Byzantine disk paxos: optimal resilience with byzantine shared memory. *Distributed Computing*, 18(5):387–408, 2006.

[21] Hussam Abu-Libdeh, Lonnie Princehouse, and Hakim Weatherspoon. RACS: A case for cloud storage diversity. *SoCC*, 2010.

[22] James Alderman, Christian Janson, Carlos Cid, and Jason Crampton. Access Control in Publicly Verifiable Outsourced Computation. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security, ASIA CCS '15, Singapore, April 14-17, 2015*, pages 657–662, 2015.

[23] Y. Amir, C. Danilov, D. Dolev, J. Kirsch, J. Lane, C. Nita-Rotaru, J. Olsen, and D. Zage. Steward: Scaling Byzantine fault-tolerant replication to wide area networks. *IEEE Transactions on Dependable and Secure Computing*, 7(1):80–93, 2010.

[24] Paul Anderson and Le Zhang. Fast and secure laptop backups with encrypted de-duplication. In *Proceedings of the 24th International Conference on Large Installation System Administration*, LISA'10, pages 1–8, Berkeley, CA, USA, 2010. USENIX Association.

[25] Giuseppe Ateniese, Karyn Benson, and Susan Hohenberger. Key-private proxy re-encryption. In *CT-RSA 2009*, volume 5473 of *Lecture Notes in Computer Science*, pages 279–294. Springer, 2009.

[26] Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary Peterson, and Dawn Song. Provable data possession at untrusted stores. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 598–609. Acm, 2007.

[27] Giuseppe Ateniese, Kevin Fu, Matthew Green, and Susan Hohenberger. Improved proxy re-encryption schemes with applications to secure distributed storage. *ACM Trans. Inf. Syst. Secur. 2006*, 9(1):1–30, 2006.

[28] Nuttapong Attrapadung. Dual system encryption via doubly selective security: Framework, fully secure functional encryption for regular languages, and more. pages 557–577, 2014.

[29] Nuttapong Attrapadung, Benoît Libert, and Elie de Panafieu. Expressive key-policy attribute-based encryption with constant-size ciphertexts. pages 90–108, 2011.

[30] Michael Backes, Manuel Barbosa, Dario Fiore, and Raphael M. Reischuk. Adsnark: Nearly practical and privacy-preserving proofs on authenticated data. Cryptology ePrint Archive, Report 2014/617, 2014. `http://eprint.iacr.org/`.

[31] Lucas Ballard, Seny Kamara, and Fabian Monrose. Achieving efficient conjunctive keyword searches over encrypted data. In *Information and Communications Security, 7th International Conference, ICICS 2005, Beijing, China, December 10-13, 2005, Proceedings*, pages 414–426, 2005.

[32] Nathalie Baracaldo, Elli Androulaki, Joseph Glider, and Alessandro Sorniotti. Reconciling end-to-end confidentiality and data reduction in cloud storage. In *Proceedings of the 6th edition of the ACM Workshop on Cloud Computing Security*, pages 21–32. ACM, 2014.

[33] C. Basescu, C. Cachin, I. Eyal, R. Haas, A. Sorniotti, M. Vukolic, and I. Zachevsky. Robust data sharing with key-value stores. In *DSN*, 2012.

[34] Mihir Bellare, Alexandra Boldyreva, and Adam O'Neill. Deterministic and efficiently searchable encryption. In *Advances in Cryptology - CRYPTO 2007, 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007, Proceedings*, pages 535–552, 2007.

[35] Mihir Bellare and Sriram Keelveedhi. Interactive message-locked encryption and secure deduplication. In Jonathan Katz, editor, *Public-Key Cryptography – PKC 2015*, volume 9020 of *Lecture Notes in Computer Science*, pages 516–538. Springer Berlin Heidelberg, 2015.

[36] Mihir Bellare, Sriram Keelveedhi, and Thomas Ristenpart. Message-locked encryption and secure deduplication. In *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, pages 296–312, 2013.

[37] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. SNARKs for C: Verifying Program Executions Succinctly and in Zero Knowledge. In Ran Canetti and JuanA. Garay, editors, *Advances in Cryptology – CRYPTO 2013*, volume 8043 of *Lecture Notes in Computer Science*, pages 90–108. Springer Berlin Heidelberg, 2013.

[38] Alysson Bessani, Miguel Correia, Bruno Quaresma, Fernando Andre, and Paulo Sousa. DepSky: Dependable and secure storage in cloud-of-clouds. *ACM Transactions on Storage*, 9(4), 2013.

[39] Alysson Bessani, Ricardo Mendes, Tiago Oliveira, Nuno Neves, Miguel Correia, Marcelo Pasin, and Paulo Verissimo. SCFS: a shared cloud-backed file system. In *Proc. of the 2014 USENIX ATC*, 2014.

[40] Alysson Bessani, Marcel Santos, Joo Felix, Nuno Neves, and Miguel Correia. On the efficiency of durable state machine replication. In *Proc. of the USENIX Annual Technical Conference – USENIX ATC 2013*, June 2013.

[41] Alysson Neves Bessani, João Sousa, and Eduardo Adílio Pelinson Alchieri. State machine replication for the masses with BFT-SMART. In *44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2014, Atlanta, GA, USA, June 23-26, 2014*, pages 355–362, 2014.

[42] Jorge Blasco, Roberto Di Pietro, Agustin Orfila, and Alessandro Sorniotti. A tunable proof of ownership scheme for deduplication using bloom filters. In *Communications and Network Security (CNS), 2014 IEEE Conference on*, pages 481–489. IEEE, 2014.

[43] Matt Blaze, Gerrit Bleumer, and Martin Strauss. Divertible protocols and atomic proxy cryptography. In *EUROCRYPT'98*, volume 1403 of *Lecture Notes in Computer Science*, pages 127–144. Springer, 1998.

[44] Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas P. Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, Michael I. Schwartzbach, and Tomas Toft. Secure Multiparty Computation Goes Live. In *Financial Cryptography and Data Security, 13th International Conference, FC 2009, Accra Beach, Barbados, February 23-26, 2009. Revised Selected Papers*, pages 325–343, 2009.

[45] William J Bolosky, Dexter Bradshaw, Randolph B Haagens, Norbert P Kusters, and Peng Li. Paxos replicated state machines as the basis of a high-performance data store. In *Symposium on Networked Systems Design and Implementation (NSDI)*, pages 141–154, 2011.

[46] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public key encryption with keyword search. In *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, pages 506–522, 2004.

[47] Dan Boneh and Brent Waters. Conjunctive, subset, and range queries on encrypted data. In Vadhan [167], pages 535–554.

[48] Matthias Brantner, Daniela Florescu, David Graf, Donald Kossmann, and Tim Kraska. Building a database on S3. In *Proc. of the 2008 ACM SIGMOD Int'l Conference on Management of Data*, pages 251–264, 2008.

[49] Mike Burrows. The chubby lock service for loosely-coupled distributed systems. In *Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI)*. USENIX Association, 2006.

[50] JinWook Byun, DongHoon Lee, and Jongin Lim. Efficient conjunctive keyword search on encrypted data storage system. In AndreaS. Atzeni and Antonio Lioy, editors, *Public Key Infrastructure*, volume 4043 of *Lecture Notes in Computer Science*, pages 184–196. Springer Berlin Heidelberg, 2006.

[51] Christian Cachin, Idit Keidar, and Alexander Shraer. Fork Sequential Consistency is Blocking. *Inf. Process. Lett.*, 109(7):360–364, 2009.

[52] Christian Cachin, Idit Keidar, and Alexander Shraer. Trusting the cloud. *SIGACT News*, 40(2):81–86, 2009.

[53] Christian Cachin, Idit Keidar, and Alexander Shraer. Fail-aware untrusted storage. *SIAM J. Comput.*, 40(2):493–533, 2011.

[54] Christian Cachin and Matthias Schunter. A cloud you can trust. *Spectrum, IEEE*, 48(12):28–51, 2011.

[55] Sébastien Canard and Julien Devigne. Combined proxy re-encryption. In *Information Security and Cryptology - ICISC 2013 - 16th International Conference, Seoul, Korea, November 27-29, 2013, Revised Selected Papers*, volume 8565, pages 49–66. Springer, 2013.

[56] Sébastien Canard, Julien Devigne, and Fabien Laguillaumie. Improving the security of an efficient unidirectional proxy re-encryption scheme. *Journal of Internet Services and Information Security (JISIS)*, 1(2/3):140–160, 8 2011.

[57] Ran Canetti and Susan Hohenberger. Chosen-ciphertext secure proxy re-encryption. In *ACM Conference on Computer and Communications Security 2007*, pages 185–194. ACM, 2007.

[58] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. Dynamic searchable encryption in very-large databases: Data structures and implementation. In *21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014*, 2014.

[59] Miguel Castro and Barbara Liskov. Practical Byzantine fault tolerance and proactive recovery. *ACM Transactions Computer Systems*, 20(4):398–461, 2002.

[60] Tushar D. Chandra, Robert Griesemer, and Joshua Redstone. Paxos made live: an engineering perspective. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*. ACM, 2007.

[61] Yan-Cheng Chang and Michael Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In *Applied Cryptography and Network Security, Third International Conference, ACNS 2005, New York, NY, USA, June 7-10, 2005, Proceedings*, pages 442–455, 2005.

[62] Melissa Chase. Multi-authority attribute based encryption. In Vadhan [167], pages 515–534.

[63] Melissa Chase and Sherman S. M. Chow. Improving privacy and security in multi-authority attribute-based encryption. In Ehab Al-Shaer, Somesh Jha, and Angelos D. Keromytis, editors, *Proceedings of the 2009 ACM Conference on Computer and Communications Security, CCS 2009, Chicago, Illinois, USA, November 9-13, 2009*, pages 121–130. ACM, 2009.

[64] Melissa Chase and Seny Kamara. Structured encryption and controlled disclosure. In *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings*, pages 577–594, 2010.

[65] Cheng Chen, Jie Chen, Hoon Wei Lim, Zhenfeng Zhang, Dengguo Feng, San Ling, and Huaxiong Wang. Fully secure attribute-based systems with short ciphertexts/signatures and threshold access structures. pages 50–67, 2013.

[66] Jie Chen, Romain Gay, and Hoeteck Wee. Improved dual system abe in prime-order groups via predicate encodings. In E. Oswald and M. Fischlin, editors, *Proceedings of EUROCRYPT*, volume 9057 of *LNCS*, pages 595–624. Springer, 2015.

[67] Sherman S. M. Chow, Jian Weng, Yanjiang Yang, and Robert H. Deng. Efficient unidirectional proxy re-encryption. In *AFRICACRYPT 2010*, volume 6055 of *Lecture Notes in Computer Science*, pages 316–332. Springer, 2010.

[68] Jae Yoon Chung, Carlee Joe-Wong, Sangtae Ha, James Won-Ki Hong, and Mung Chiang. CYRUS: Towards client-defined cloud storage. In *Proc. of the 10th ACM European Systems Conference – EuroSys'15*, 2015.

[69] James Corbet et. al. Spanner: Google's globally distributed database. *ACM Transactions on Computer Systems*, 31(3):8:1–8:22, August 2013.

[70] Ronald Cramer, Ivan Bjerre Damgrd, and Jesper Buus Nielsen. *Secure Multiparty Computation and Secret Sharing*. Cambridge University Press, 2015.

[71] Reza Curtmola, Juan A. Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, Alexandria, VA, USA, Ioctober 30 - November 3, 2006*, pages 79–88, 2006.

[72] Cyphertite. Cyphertite data backup. Technical report.

[73] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. Practical Covertly Secure MPC for Dishonest Majority - Or: Breaking the SPDZ Limits. In *Computer Security - ESORICS 2013 - 18th European Symposium on Research in Computer Security, Egham, UK, September 9-13, 2013. Proceedings*, pages 1–18, 2013.

[74] Sebastiaan de Hoogh, Berry Schoenmakers, and Meilof Veeningen. Universally verifiable outsourcing and application to linear programming. In P. Laud and L. Kamm, editors, *Applications of Secure Multiparty Computation*, volume 13. IOS Press, 2015.

[75] D. Dobre, P. Viotti, and M. Vukolic. Hybris: Robust hybrid cloud storage. *SoCC*, 2014.

[76] J.R. Douceur, A. Adya, W.J. Bolosky, P. Simon, and M. Theimer. Reclaiming space from duplicate files in a serverless distributed file system. In *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on*, pages 617–624, 2002.

[77] Jiaqing Du, Daniele Sciascia, Sameh Elnikety, Willy Zwaenepoel, and Fernando Pedone. Clock-RSM: Low-latency inter-datacenter state machine replication using loosely synchronized physical clocks. In *DSN*, 2014.

[78] Lo Ducas and Daniele Micciancio. FHEW: Bootstrapping Homomorphic Encryption in less than a second. Cryptology ePrint Archive, Report 2014/816, 2014. `http://eprint.iacr.org/`.

[79] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35, April 1988.

[80] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating Noise to Sensitivity in Private Data Analysis. In *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006, Proceedings*, pages 265–284, 2006.

[81] Keita Emura, Atsuko Miyaji, Akito Nomura, Kazumasa Omote, and Masakazu Soshi. A ciphertext-policy attribute-based encryption scheme with constant ciphertext length. In Bao F, Li H, and Wang G, editors, *Proceedings of ISPEC*, volume 5451 of *LNCS*, pages 13–23. Springer, 2009.

[82] A. Machanavajjhala et al. L-diversity: privacy beyond k-anonymity. In *Proceedings of the 22nd International Conference on Data Engineering*, page 24, 2006.

[83] Flud. The flud backup system. Technical report.

[84] Anshul Gandhi, Mor Harchol-Balter, Ram Raghunathan, and Michael A. Kozuch. Autoscale: Dynamic, robust capacity management for multi-tier data centers. *ACM Trans. Comput. Syst.*, 30(4):14:1–14:26, November 2012.

[85] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic Span Programs and Succinct NIZKs without PCPs. In Thomas Johansson and PhongQ. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 626–645. Springer Berlin Heidelberg, 2013.

[86] Lisa Glendenning, Ivan Beschastnikh, Arvind Krishnamurthy, and Thomas Anderson. Scalable consistency in scatter. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, pages 15–28, New York, NY, USA, 2011. ACM.

[87] Eu-Jin Goh. Secure indexes. *IACR Cryptology ePrint Archive*, 2003:216, 2003.

[88] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. *J. ACM*, 43(3):431–473, 1996.

[89] Philippe Golle, Jessica Staddon, and Brent R. Waters. Secure conjunctive keyword search over encrypted data. In *Applied Cryptography and Network Security, Second International Conference, ACNS 2004, Yellow Mountain, China, June 8-11, 2004, Proceedings*, pages 31–45, 2004.

[90] Lorena González-Manzano and Agustin Orfila. An efficient confidentiality-preserving proof of ownership for deduplication. *Journal of Network and Computer Applications*, 2015.

[91] Michael T. Goodrich, Michael Mitzenmacher, Olga Ohrimenko, and Roberto Tamassia. Privacy-preserving group data access via stateless oblivious RAM simulation. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 157–167, 2012.

[92] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. pages 89–98, 2006. Available as Cryptology ePrint Archive Report 2006/309.

[93] Jens Groth. Short Pairing-Based Non-interactive Zero-Knowledge Arguments. In *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings*, pages 321–340, 2010.

[94] Zhenyu Guo, Chuntao Hong, Mao Yang, Dong Zhou, Lidong Zhou, and Li Zhuang. Rex: Replication at the speed of multi-core. In *Proc. of the 9th European Conference on Computer Systems – EuroSys '14*, 2014.

[95] Shai Halevi, Danny Harnik, Benny Pinkas, and Alexandra Shulman-Peleg. Proofs of ownership in remote storage systems. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 491–500. ACM, 2011.

[96] Seungyeop Han, Haichen Shen, Taesoo Kim, Arvind Krishnamurthy, Thomas Anderson, and David Wetherall. MetaSync: File synchronization across multiple untrusted storage services. In *Proc. of the 2015 USENIX ATC*, 2015.

[97] Danny Harnik, Benny Pinkas, and Alexandra Shulman-Peleg. Side channels in cloud services, the case of deduplication in cloud storage. *IEEE Security & Privacy*, 8(6):40–47, 2010.

[98] Maurice Herlihy and Jeannette M. Wing. Linearizability: A correctness condition for concurrent objects. *ACM Transactions on Programing Languages and Systems*, 12(3):463–492, 1990.

[99] Javier Herranz, Fabien Laguillaumie, and Carla Ràfols. Constant size ciphertexts in threshold attribute-based encryption. pages 19–34, 2010.

[100] Patrick Hunt, Mahadev Konar, Flavio P. Junqueira, and Benjamin Reed. ZooKeeper: wait-free coordination for internet-scale systems. In *Proceedings of the 2010 USENIX conference on USENIX annual technical conference*, USENIX ATC'10, pages 11–11, 2010.

[101] Mohammad Saiful Islam, Mehmet Kuzu, and Murat Kantarcioglu. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *19th Annual Network and Distributed System Security Symposium, NDSS 2012, San Diego, California, USA, February 5-8, 2012*, 2012.

[102] Zhang Jinsheng, Zhang Wensheng, and Daji Qiao. A multi-user oblivious ram for outsourced data. Technical report, Iowa State University, 2014.

[103] Ari Juels and Burton S Kaliski Jr. Pors: Proofs of retrievability for large files. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 584–597. Acm, 2007.

[104] Pascal Junod and Alexandre Karlov. An efficient public-key attribute-based broadcast encryption scheme allowing arbitrary access policies. In *ACM Workshop on Digital Rights Management*, pages 13–24. ACM Press, 2010.

[105] Flavio Junqueira, Yanhua Mao, and Keith Marzullo. Classic Paxos vs Fast Paxos: Caveat emptor. In *Proc. of the Workshop on Hot Topics in System Dependability*, 2007.

[106] Seny Kamara, Charalampos Papamanthou, and Tom Roeder. Dynamic searchable symmetric encryption. In *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*, pages 965–976, 2012.

[107] Manos Kapritsos, Yang Wang, Vivien Quema, Allen Clement, Lorenzo Alvisi, and Mike Dahlin. All about Eve: execute-verify replication for multi-core servers. In *Proceedings of the 10th USENIX conference on Operating Systems Design and Implementation*, OSDI'12, pages 237–250, Berkeley, CA, USA, 2012. USENIX Association.

[108] Nikolaos P. Karvelas, Andreas Peter, Stefan Katzenbeisser, Erik Tews, and Kay Hamacher. Privacy-preserving whole genome sequence processing through proxy-aided ORAM. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society, WPES 2014, Scottsdale, AZ, USA, November 3, 2014*, pages 1–10, 2014.

[109] Ramakrishna Kotla, Lorenzo Alvisi, and Mike Dahlin. SafeStore: A durable and practical storage system. 2007.

[110] Ramakrishna Kotla and Mike Dahlin. High throughput byzantine fault tolerance. In *Proc. of the 2004 International Conference on Dependable Systems and Networks – DSN'04*, 2004.

[111] Tim Kraska, Gene Pang, Michael J. Franklin, Samuel Madden, and Alan Fekete. MDCC: multi-data center consistency. In *Eighth Eurosys Conference 2013, EuroSys '13, Prague, Czech Republic, April 14-17, 2013*, pages 113–126, 2013.

[112] Andrew Krioukov, Prashanth Mohan, Sara Alspaugh, Laura Keys, David Culler, and Randy Katz. Napsac: Design and implementation of a power-proportional web cluster. In *Proc. of the 1st ACM Workshop on Green Networking*, 2010.

[113] Kripa Krishnan. Weathering the unexpected. *Commun. ACM*, 55(11):48–52, November 2012.

[114] Petr Kuznetsov and Rodrigo Rodrigues. BFTW3: Why? When? Where? Workshop on the theory and practice of Byzantine fault tolerance. *SIGACT News*, 40(4):82–86, January 2010.

[115] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.

[116] Leslie Lamport. The part-time parliament. *ACM Trans. Computer Systems*, 16(2):133–169, May 1998.

[117] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

[118] Allison B. Lewko, Tatsuaki Okamoto, Amit Sahai, Katsuyuki Takashima, and Brent Waters. Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. pages 62–91, 2010.

[119] Allison B. Lewko and Brent Waters. Unbounded HIBE and attribute-based encryption. pages 547–567, 2011.

[120] Jinyuan Li and David Mazières. Beyond one-third faulty replicas in byzantine fault tolerant systems. In *NSDI*, 2007.

[121] Mingqiang Li, Chuan Qin, and Patrick P. C. Lee. Cdstore: Toward reliable, secure, and cost-efficient cloud storage via convergent dispersal. In *2015 USENIX Annual Technical Conference (USENIX ATC 15)*, pages 111–124, Santa Clara, CA, July 2015. USENIX Association.

[122] Benoît Libert and Damien Vergnaud. Unidirectional chosen-ciphertext secure proxy re-encryption. In *Public Key Cryptography 2008*, volume 4939 of *Lecture Notes in Computer Science*, pages 360–379. Springer, 2008.

[123] Shengyun Liu, Christian Cachin, Vivien Quéma, and Marko Vukolic. XFT: practical fault tolerance beyond crashes. *CoRR*, abs/1502.05831, 2015.

[124] Shengyun Liu and Marko Vukolić. How many planet-wide leaders should there be? In *Operating Systems Review, Proceedings of the 3rd Workshop on Distributed Cloud Computing*, 2015.

[125] Jacob R. Lorch, James W. Mickens, Bryan Parno, Mariana Raykova, and Joshua Schiffman. Toward practical private access to data centers via parallel ORAM. *IACR Cryptology ePrint Archive*, 2012:133, 2012.

[126] Prince Mahajan, Lorenzo Alvisi, and Mike Dahlin. Consistency, availability, and convergence. Technical Report TR-11-22, Computer Science Department, University of Texas at Austin, 2011.

[127] Prince Mahajan, Srinath T. V. Setty, Sangmin Lee, Allen Clement, Lorenzo Alvisi, Michael Dahlin, and Michael Walfish. Depot: Cloud storage with minimal trust. In *OSDI*, pages 307–322, 2010.

[128] Dahlia Malkhi and Michael Reiter. Byzantine quorum systems. *Distributed Computing*, 11(4):203–213, 1998.

[129] Yanhua Mao, Flavio P. Junqueira, and Keith Marzullo. Mencius: building efficient replicated state machines for WANs. In *Proc. of the 8th USENIX Conference on Operating Systems Design and Implementation*, 2008.

[130] Luis Marques and Carlos J. Costa. Secure deduplication on mobile devices. In *Proceedings of the 2011 Workshop on Open Source and Design of Communication*, OSDOC '11, pages 19–26, New York, NY, USA, 2011. ACM.

[131] Toshihide Matsuda, Ryo Nishimaki, and Keisuke Tanaka. Cca proxy re-encryption without bilinear maps in the standard model. In *Public Key Cryptography 2010*, volume 6056 of *Lecture Notes in Computer Science*, pages 261–278. Springer, 2010.

[132] Travis Mayberry, Erik-Oliver Blass, and Guevara Noubir. Multi-user oblivious RAM secure against malicious servers. *IACR Cryptology ePrint Archive*, 2015:121, 2015.

[133] David Mazières and Dennis Shasha. Building Secure File Systems out of Byzantine Storage. In *Proceedings of PODC*, pages 108–117, 2002.

[134] Medical Imaging & Technology Alliance. DICOM PS3.2, 2015.

[135] Iulian Moraru, David G. Andersen, and Michael Kaminsky. There is more consensus in egalitarian parliaments. In *Proc. of 24th ACM Symposium on Operating Systems Principles*, 2013.

[136] Martin Mulazzani, Sebastian Schrittwieser, Manuel Leithner, Markus Huber, and Edgar Weippl. Dark clouds on the horizon: Using cloud storage as attack vector and online slack space. In *USENIX Security Symposium*, 2011.

[137] Dushyanth Narayanan, Austin Donnelly, Eno Thereska, Sameh Elnikety, and Antony Rowstron. Everest: Scaling down peak loads through i/o off-loading. In *Proc. of the 8th USENIX Conference on Operating Systems Design and Implementation – OSDI'08*, 2008.

[138] Ripal Nathuji, Aman Kansal, and Alireza Ghaffarkhah. Q-clouds: Managing performance interference effects for QoS-aware clouds. In *Proceedings of the 5th European Conference on Computer Systems – EuroSys '10*, pages 237–250, 2010.

[139] Tiancheng Li Ninghui Li and S. Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and l-diversity. In *IEEE 23rd International Conference on Data Engineering*, pages 106–115, 2007.

[140] Rajesh Nishtala, Hans Fugal, Steven Grimm, Marc Kwiatkowski, Herman Lee, Harry C. Li, Ryan McElroy, Mike Paleczny, Daniel Peek, Paul Saab, David Stafford, Tony Tung, and Venkateshwaran Venkataramani. Scaling memcache at Facebook. April 2013.

[141] Tatsuaki Okamoto and Katsuyuki Takashima. Fully secure unbounded inner-product and attribute-based encryption. pages 349–366, 2012.

[142] Diego Ongaro and John K. Ousterhout. In search of an understandable consensus algorithm. In *2014 USENIX Annual Technical Conference, USENIX ATC '14, Philadelphia, PA, USA, June 19-20, 2014.*, pages 305–319, 2014.

[143] Alina Oprea and Michael K. Reiter. On consistency of encrypted files. In *DISC*, pages 254–268, 2006.

[144] Rafail Ostrovsky, Amit Sahai, and Brent Waters. Attribute-based encryption with non-monotonic access structures. pages 195–203, 2007.

[145] Ricardo Padilha and Fernando Pedone. Augustus: Scalable and robust storage for cloud applications. In *Proceedings of the eigth conference on Computer systems*, EuroSys '13, 2013.

[146] B. Parno, J. Howell, C. Gentry, and M. Raykova. Pinocchio: Nearly Practical Verifiable Computation. In *Security and Privacy, 2013 IEEE Symposium on*, pages 238–252, 2013.

[147] Patel, P. et al. Ananta: Cloud scale load balancing. In *ACM SIGCOMM'13*, 2013.

[148] João Paulo and José Pereira. A survey and classification of storage deduplication systems. *ACM Computing Surveys (CSUR)*, 47(1):11, 2014.

[149] F. Pedone, C. E. Bezerra, and R. van Renesse. Scalable state-machine replication. In *44th International Conference on Dependable Systems and Networks (DSN 2014)*, 2014.

[150] Yannis Rouselakis and Brent Waters. Practical constructions and new proof methods for large universe attribute-based encryption. pages 463–474, 2013.

[151] Amit Sahai and Brent R. Waters. Fuzzy identity-based encryption. pages 457–473, 2005.

[152] Fred Schneider. Implementing fault-tolerant service using the state machine aproach: A tutorial. *ACM Computing Surveys*, 22(4):299–319, 1990.

[153] Berry Schoenmakers and Meilof Veeningen. Universally Verifiable Multiparty Computation from Threshold Homomorphic Cryptosystems. In *Applied Cryptography and Network Security (ACNS)*, 2015. `http://eprint.iacr.org/2015/058`.

[154] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, November 1979.

[155] Jun Shao and Zhenfu Cao. Cca-secure proxy re-encryption without pairings. In *Public Key Cryptography 2009*, volume 5443 of *Lecture Notes in Computer Science*, pages 357–376. Springer, 2009.

[156] Elaine Shi, John Bethencourt, Hubert T.-H. Chan, Dawn Xiaodong Song, and Adrian Perrig. Multi-dimensional range query over encrypted data. In *2007 IEEE Symposium on Security and Privacy (S&P 2007), 20-23 May 2007, Oakland, California, USA*, pages 350–364, 2007.

[157] Dawn Xiaodong Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *2000 IEEE Symposium on Security and Privacy, Berkeley, California, USA, May 14-17, 2000*, pages 44–55, 2000.

[158] J. Sousa and A. Bessani. Separating the WHEAT from the chaff: An empirical design for geo-replicated state machines. In *Proc. of the 34th Symposium on Reliable Distributed Systems*, September 2015.

[159] Emil Stefanov and Elaine Shi. Multi-cloud oblivious storage. In *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 247–258, 2013.

[160] Emil Stefanov, Marten van Dijk, Ari Juels, and Alina Oprea. Iris: a scalable cloud file system with efficient integrity checks. In *ACSAC*, 2012.

[161] Emil Stefanov, Marten van Dijk, Elaine Shi, Christopher W. Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path ORAM: an extremely simple oblivious RAM protocol. In *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 299–310, 2013.

[162] Mark W. Storer, Kevin Greenan, Darrell D.E. Long, and Ethan L. Miller. Secure data deduplication. In *Proceedings of the 4th ACM International Workshop on Storage Security and Survivability*, StorageSS '08, pages 1–10, New York, NY, USA, 2008. ACM.

[163] Latanya Sweeney. k-Anonymity: A Model for Protecting Privacy. In *School of Computer Science*, pages 1–14, 2002.

[164] Haowen Tang, Fangming Liu, Guobin Shen, Yuchen Jin, and Chuanxiong Guo. UniDrive: Synergize multiple consumer cloud storage services. In *Proc. of the ACM/IFIP/USENIX Middleware'15*, 2015.

[165] Douglas B. Terry, Vijayan Prabhakaran, Ramakrishna Kotla, Mahesh Balakrishnan, Marcos K. Aguilera, and Hussam Abu-Libdeh. Consistency-based service level agreements for cloud storage. In *Proc. of the 24th ACM Symposium on Operating Systems Principles – SOSP'13*, pages 309–324, 2013.

[166] Piotr K. Tysowski and M. Anwarul Hasan. Re-encryption-based key management towards secure and scalable mobile applications in clouds. *IACR Cryptology ePrint Archive*, 2011:668, 2011.

[167] Salil P. Vadhan, editor. *Theory of Cryptography, 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, February 21-24, 2007, Proceedings*, volume 4392 of *Lecture Notes in Computer Science*. Springer, 2007.

[168] G. Veronese, M. Correia, A.N. Bessani, and Lau Cheuk Lung. EBAWA: Efficient Byzantine agreement for wide-area networks. In *Proc. of the 12th IEEE Int. High Assurance Systems Engineering Symposium*, 2010.

[169] Werner Vogels. Eventually consistent. *ACM Queue*, 6(6):14–19, 2008.

[170] Michael Vrable, Stefan Savage, and Geoffrey M. Voelker. BlueSky: A cloud-backed file system for the enterprise. In *FAST*, 2012.

[171] Marko Vukolić. The byzantine empire in the intercloud. *SIGACT News*, 41(3):105–111, September 2010.

[172] Marko Vukolic. *Quorum Systems: With Applications to Storage and Consensus*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2012.

[173] Brent R. Waters, Dirk Balfanz, Glenn Durfee, and Diana K. Smetters. Building an encrypted and searchable audit log. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2004, San Diego, California, USA*, 2004.

[174] Hoeteck Wee. Dual system encryption via predicate encodings. pages 616–637, 2014.

[175] Jian Weng, MinRong Chen, YanJiang Yang, Robert Deng, KeFei Chen, and Feng Bao. Cca-secure unidirectional proxy re-encryption in the adaptive corruption model without random oracles. *SCIENCE CHINA Information Sciences*, 53:593–606, 2010.

[176] Zhe Wu, Michael Butkiewicz, Dorian Perkins, Ethan Katz-Bassett, and Harsha V Madhyastha. SPANStore: Cost-effective geo-replicated storage spanning multiple cloud services. In *Proc. of the 24th ACM Symposium on Operating Systems Principles – SOSP'13*, pages 292–308, 2013.

[177] Jia Xu and Jianying Zhou. Leakage resilient proofs of ownership in cloud storage, revisited. In *Applied Cryptography and Network Security*, pages 97–115. Springer, 2014.

[178] Shota Yamada, Nuttapong Attrapadung, Goichiro Hanaoka, and Noboru Kunihiro. A framework and compact constructions for non-monotonic attribute-based encryption. pages 275–292, 2014.