THÈSE DE DOCTORAT CONJOINT TELECOM SUDPARIS et L'UNIVERSITE PIERRE ET MARIE CURIE

Spécialité : Informatique et Réseaux

École doctorale : Informatique, Télécommunications et Electronique de Paris

Présentée par

Nesrine KAANICHE

Pour obtenir le grade de
DOCTEUR DE TELECOM SUDPARIS

La sécurité des données stockées dans un environnement Cloud, basée sur des mécanismes cryptographiques

Soutenue le 15 Décembre 2014, devant le jury composé de :

Christophe BIDAN
Professeur, Supélec, France – *Rapporteur*

Sébastien CANARD
Ingénieur de recherche, Orange Labs, France – *Rapporteur*

Georg CARLE
Professeur, Technische Universität München, Allemagne – *Examinateur*

Thierry COUPAYE
Directeur de recherche, Orange Labs, France – *Examinateur*

Yves ROUDIER
Maître de conférences, EURECOM, France – *Examinateur*

Pierre SENS
Professeur, Université Pierre et Marie Curie, France – *Examinateur*

Maryline LAURENT
Professeur, Télécom SudParis, France – *Directrice de thèse*

Thèse No : 2014TELE0033

PHD THESIS TELECOM SUDPARIS IN PARTNERSHIP WITH PIERRE ET MARIE CURIE UNIVERSITY

Speciality : Informatics and Networks

Doctoral School : Informatique, Télécommunications et Électronique de Paris

Presented by

Nesrine KAANICHE

To obtain the degree of
DOCTOR OF TELECOM SUDPARIS

Cloud Data Storage Security based on Cryptographic Mechanisms

---

Presented on December 15th, 2014 with the Jury composed by :

Thesis No : 2014TELE0033

Dedicated to Chrif for 1001 *reasons.*

Abstract

Recent technological advances have given rise to the popularity and success of cloud. This new paradigm is gaining an expanding interest, since it provides cost efficient architectures that support the transmission, storage, and intensive computing of data. However, these promising storage services bring many challenging design issues, considerably due to the loss of data control. These challenges, namely data confidentiality and data integrity, have significant influence on the security and performances of the cloud system. Some threat models assume that the cloud service provider cannot be trusted, and therefore security designers propose a high level security assurance, such as storing encrypted data in cloud servers. Others suppose that cloud providers can be trusted, and that potential threats come primarily from outside attackers and other malicious cloud users. Furthermore, a cloud user can never deny a potential server breakdown. Therefore, there are several challenges that need to be addressed with respect to security and privacy in a cloud context.

This thesis aims at overcoming this trade-off, while considering two data security concerns.

On one hand, we focus on data confidentiality preservation which becomes more complex with flexible data sharing among a dynamic group of users. It requires the secrecy of outsourced data and an efficient sharing of decrypting keys between different authorized users.
For this purpose, we, first, proposed a new method relying on the use of ID-Based Cryptography (IBC), where each client acts as a Private Key Generator (PKG). That is, he generates his own public elements and derives his corresponding private key using a secret. Thanks to IBC properties, this contribution is shown to support data privacy and confidentiality, and to be resistant to unauthorized access to data during the sharing process, while considering two realistic threat models, namely an honest but curious server and a malicious user adversary.
Second, we define CloudaSec, a public key based solution, which proposes the separation of subscription-based key management and confidentiality-oriented asymmetric encryption policies. That is, CloudaSec enables flexible and scalable deployment of the solution as well as strong security guarantees for outsourced data in cloud servers. Experimental results, under OpenStack Swift, have proven the efficiency of CloudaSec in scalable data sharing,

while considering the impact of the cryptographic operations at the client side.

On the other hand, we address the Proof of Data Possession (PDP) concern. In fact, the cloud customer should have an efficient way to perform periodical remote integrity verifications, without keeping the data locally, following three substantial aspects : *security level, public verifiability,* and *performance.* This concern is magnified by the client's constrained storage and computation capabilities and the large size of outsourced data.
In order to fulfill this security requirement, we first define a new zero-knowledge PDP protocol that provides deterministic integrity verification guarantees, relying on the uniqueness of the Euclidean Division. These guarantees are considered as interesting, compared to several proposed schemes, presenting probabilistic approaches.
Then, we propose SHoPS, a Set-Homomorphic Proof of Data Possession scheme, supporting the 3 levels of data verification. SHoPS enables the cloud client not only to obtain a proof of possession from the remote server, but also to verify that a given data file is distributed across multiple storage devices to achieve a certain desired level of fault tolerance. Indeed, we present the set homomorphism property, which extends malleability to set operations properties, such as union, intersection and inclusion. SHoPS presents high security level and low processing complexity. For instance, SHoPS saves energy within the cloud provider by distributing the computation over multiple nodes. Each node provides proofs of local data block sets. This is to make applicable, a resulting proof over sets of data blocks, satisfying several needs, such as, proofs aggregation.

Résumé

Au cours de la dernière décennie, avec la standardisation d'Internet, le développement des réseaux à haut débit, le paiement à l'usage et la quête sociétale de la mobilité, le monde informatique a vu se populariser un nouveau paradigme, le *Cloud*. Le recours au *cloud* est de plus en plus remarquable compte tenu de plusieurs facteurs, notamment ses architectures rentables, prenant en charge la transmission, le stockage et le calcul intensif de données. Cependant, ces services de stockage prometteurs soulèvent la question de la protection des données et de la conformité aux réglementations, considérablement due à la perte de maîtrise et de gouvernance.

Cette dissertation vise à surmonter ce dilemme, tout en tenant compte de deux préoccupations de sécurité des données, à savoir la confidentialité des données et l'intégrité des données.

En premier lieu, nous nous concentrons sur la confidentialité des données, un enjeu assez considérable étant donné le partage de données flexible au sein d'un groupe dynamique d'utilisateurs. Cet enjeu exige, par conséquence, un partage efficace des clés entre les membres du groupe.
Pour répondre à cette préoccupation, nous avons, d'une part, proposé une nouvelle méthode reposant sur l'utilisation de la cryptographie basée sur l'identité (IBC), où chaque client agit comme une entité génératrice de clés privées. Ainsi, il génère ses propres éléments publics et s'en sert pour le calcul de sa clé privée correspondante. Grâce aux propriétés d'IBC, cette contribution a démontré sa résistance face aux accès non autorisés aux données au cours du processus de partage, tout en tenant compte de deux modèles de sécurité, à savoir un serveur de stockage honnête mais curieux et un utilisateur malveillant.
D'autre part, nous définissons CloudaSec, une solution à base de clé publique, qui propose la séparation de la gestion des clés et les techniques de chiffrement, sur deux couches. En effet, CloudaSec permet un déploiement flexible d'un scénario de partage de données ainsi que des garanties de sécurité solides pour les données externalisées sur les serveurs du cloud. Les résultats expérimentaux, sous OpenStack Swift, ont prouvé l'efficacité de CloudaSec, en tenant compte de l'impact des opérations cryptographiques sur le terminal du client.

En deuxième lieu, nous abordons la problématique de la preuve de possession de données (PDP). En fait, le client du cloud doit avoir un moyen efficace lui permettant d'effectuer

des vérifications périodiques d'intégrité à distance, sans garder les données localement. La preuve de possession se base sur trois aspects : le niveau de sécurité, la vérification publique, et les performances. Cet enjeu est amplifié par des contraintes de stockage et de calcul du terminal client et de la taille des données externalisées.
Afin de satisfaire à cette exigence de sécurité, nous définissons d'abord un nouveau protocole PDP, sans apport de connaissance, qui fournit des garanties déterministes de vérification d'intégrité, en s'appuyant sur l'unicité de la division euclidienne. Ces garanties sont considérées comme intéressantes par rapport à plusieurs schémas proposés, présentant des approches probabilistes.
Ensuite, nous proposons SHoPS, un protocole de preuve de possession de données capable de traiter les trois relations d'ensembles homomorphiques. SHoPS permet ainsi au client non seulement d'obtenir une preuve de la possession du serveur distant, mais aussi de vérifier que le fichier, en question, est bien réparti sur plusieurs périphériques de stockage permettant d'atteindre un certain niveau de la tolérance aux pannes. En effet, nous présentons l'ensemble des propriétés homomorphiques, qui étend la malléabilité du procédé aux propriétés d'union, intersection et inclusion.

Acknowledgement

Most of the important things in the world have been accomplished by people who have kept on trying when there seemed to be no hope at all.

Dale Carnegie.
-

THere are quite a few people that have contributed in one way or another to the accomplishment of this work. Some of these people even come unexpectedly to our lives to give us a word of courage or just to listen to us when we are down, or when we do not find an answer to our multiple questions. I would like to thank all of you from very deep inside.

I would like to express my sincerest gratitude and thanks to **Maryline LAURENT**, my thesis director, for her support, her dedication, her trust and her advices throughout the three years of my thesis. It is absolutely difficult to succeed in the process of finding and developing an idea without the help of a specialist in the domain. I found in my director not only the source of wonderful ideas to develop, but also the support that a PhD student needs. Without any doubt, the influence of Maryline in my life has largely contribute to what I have accomplished today.

I am very much thankful to **Aymen BOUDGUIGA**, my senior project supervisor, for his guidance and valuable advice during the first year of my PhD. He always helps me improve new ideas and dedicates part of his time to discuss and find an interesting solution to most of the problems I faced. Thanks to **Ethmane EL MOUSTAINE** for his time, his suggestions and the attention he put to my work.

A special thank to the Project **"ODISEA"** (Open DIstributed and networked StoragE Architecture) and all the partners, that contributed with this dissertation in providing a variety of real case studies. Thanks to **Mohammed El Barbori** for his implementation of CloudaSec, and above all, for being always willing to help in searching the best solution to the different technical problems faced during the integration.

I would also like to thank **Prof. Christophe BIDAN** and **Dr. Sébastien CANARD** who, as reporters and members of the jury, had the hard task of reading my thesis and giving their advice in order to improve its content. Thanks a lot to **Prof. Georg CARLE**, **Prof. Thierry COUPAYE**, **Prof. Yves ROUDIER**, and **Prof. Pierre SENS** for their interest, involvement and for being part of the jury of my thesis.

I am eternally indebted to my loving parents, my husband **Chrif** and all my family members especially my father **Nejib**, my father in law **Hassan**, my mother **Habiba**, my mother in law **Leila**, my crazy and lovely sisters **Najla, Imen, Nedia** and **Nada**. They readily and selflessly tried to provide the best conditions, generous care, dedication and support to achieve my thesis. Thanks for always showing the pride in their faces while referring to me and my achievements.

A special thank to **Hanen, Nahed** and **Wijdene** for their love, patience and encouragement whenever I was in need. My very sincere thanks to **Nadia & Rachid BOUTAR** for their parent-like support, generous care and the home feeling whenever I was in need.

A big big thank to all my friends, **Majdi, Haytham, Walid** and especially **Moutie**, who always finds 1001 ways to disturb me, but who supported me so much. You guys have all been my family in this country, thanks for being white share of this adventure.

Thanks to my colleagues for all the time spent together, the team meetings, and discussions. It has been a great experience having you all around.

I can not conclude this acknowledgement without thanking **Francoise ABAD** for her love and her effort in making sure that our missions were treated properly, and for always helping us in finding a solution to our flights and hotel problems.

Thanks a lot, Merci beaucoup, Muchas Gracias to everybody that contributed directly and indirectly to the realization of this dissertation.

Enjoy your reading!!!

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

There is nothing more difficult to take in hand, more perilous to conduct, or more uncertain in its success, than to take the lead in the introduction of a new order of things

The Prince
N. Machiavelli - 1513

RECENT technological advances relieve an explosive growth of digital contents. The US International Data Corporation (IDC) proclaims that the digital universe will grow by a factor of 300, up to 40 trillion gigabytes of replicated data by 2020 [GR12]. This proliferation of digital universe continues to rise the demand for new storage and network utilities, along with an increasing need for more cost-effective usage of storage capacities and network bandwidth for data transfer. As such, the use of remote storage systems is gaining an expanding interest, namely the *Cloud storage based services*, since they provide profitable architectures. These architectures support the transmission, storage, and intensive computation of outsourced data in a pay per use business model.
This widespread interest in cloud storage services mainly emanates from business organizations and government agencies seeking for more resilient and cost-effective systems. That is, the benefits of cloud adoption are very tangible in a new era of responsiveness, effectiveness and efficiency in *Information Technology* service delivery. Hence, there is no longer need to spend large amounts of capital on buying expensive application software or sophisticated hardware that they might never need again. These economical benefits present the main essential motivations for cloud adoption as they help enterprises reducing the Capital Expenditure (CapEx), reserved to buy fixed assets and the Operational Expenditure (OpEx) which is an ongoing cost for running a product, business, or a system.

However, despite all these attractive features, this new paradigm brings several cloud-specific security issues, especially due to its outsourcing and multi-tenancy characteristics.

Dealing with these issues more closely, we perceive that many of the cloud security concerns are essentially old problems in a new setting. For example, corporate partnerships and offshore outsourcing involve similar trust and regulatory issues. Similarly, open source software enables IT departments to quickly build and deploy applications, but at the cost of control and governance. Moreover, virtual machine attacks and Web service vulnerabilities existed long before cloud computing became fashionable. Thus, the diversity of these services delivered through cloud infrastructures increases their vulnerability to security incidents and attacks. Therefore, these challenges need to be addressed with respect to security and privacy in a cloud context.

1.1 Cloud Storage Basics & Challenges

Many people are confused about what cloud computing is, especially as the term is overused. Roughly, it describes highly scalable resources provided as an external service via the Internet on a pay per use basis. Cloud computing can be defined as a specialized distributed computing model, which is dynamically configured and delivered on demand. This new massively scalable paradigm is different from traditional networks. It is highly abstract to deliver three levels of services.
Economically, the main attractiveness of cloud computing is that users only use what they need, and only pay for what they actually use. Resources are available to be accessed from the cloud at any time, and from any location through networks. There is no need to worry about how things are being maintained. The US National Institute of Standards and Technology (NIST) [PT09] provides a formal definition of the cloud computing as follows:

"Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."

In this sense, cloud users are raised to a level of abstraction that hides the details of hardware or software infrastructures, deployed for supporting intensive computation and data storage. From this definition, three main key points have to be considered. First, NIST outlines the development of technologies that support an omnipresent, universal and appropriate new business model. Second, it involves the importance of network access techniques to shared resources that assure a fluid interaction between the cloud providers and their clients. Third, this definition focuses on the associated pricing model of cloud which allows users to pay only for consumed resources.
To better understand the core concepts and technologies in the cloud, we extract from the NIST definition document [PT09] five attributes. These attributes describe a cloud based system as a general model providing metered on demand services to his clients. These characteristics are presented as follows:

- *on-demand self-service*– cloud users may obtain extra resources, such as the usage of storage capacities and computing performances, without any human intervention.

Similar to the principle of autonomic computing, this cloud property refers to the self managing characteristics of distributed computing resources, adapting to unpredictable changes while hiding intrinsic complexity to operators and users, in order to overcome the growing complexity of computing systems management, and to reduce the barrier that complexity raises.

- *broad network access*– the large variety of heterogeneous devices, such as mobile phones, PCs, tablets, and all hand-held and static equipments have to be able to access to cloud services through standard mechanisms. This cloud ubiquitous network access characteristic is usually supported using standard protocols via Internet. Nevertheless, public networks are considered as untrusted, and therefore, several attacks may be relevant to the cloud context, like Man In The Middle attacks (MITMs).

- *shared resources*– based on a multi-tenant model, cloud resources are shared among several users. We must note that there are no resources dedicated to a specific client. These shared capabilities are assigned, allocated and reassigned as needed to the requesting entities.
  The *shared resources* property is almost supported by several providers based on virtualization techniques, where multiple Operating Systems (OSs) co-reside on the same physical machine. However, Virtual Machines (VMs) co-residence has raised certain security requirements, namely data and process isolation.
  Moreover, in multi-tenant environments, the loss of physical control and even the lack of trustful relations between the cloud client and his service provider show a great need to traditional security concerns with a new framework, such as ensuring confidentiality and integrity of outsourced data on remote servers.

- *elasticity*– along with self provisioning resources, cloud is characterized with the major capability to efficiently locate and release resources. This property demonstrates a scalability of greater resources. In fact, these resources are abstracted to cloud users in order to appear as unlimited and suitable.
  One key vulnerability that must be considered is the bandwidth under provisioning, of which malicious users can take advantage to target a service or an application availability, through Denial Of Service (DOS) attacks. Therefore, the scalability and network reliability remain important key factors to guarantee the elasticity characteristic of a cloud model.

- *metered service*– this property refers to the business model adopted by cloud based services, where users pay on a consumption basis, enabling major cost reductions. By this way, the authentication and the accountability requirements have to be considered as significant needs. Moreover, the provision of metered services is supported by several monitoring tools, in order to ensure business continuity and data investigation needs.

These attributes illustrate cloud based characteristics compared to traditional computing models, supporting more efficient and scalable services.
Cloud systems can be classified based on their deployment as private, public or hybrid infrastructures. As a combination of hardware and software resources, cloud infrastructures

provide the aforementioned characteristics, to support three cloud service models [PT09]: Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS). Building upon hardware facilities, these service models are offered in various forms:

- *Infrastructure as a Service (IaaS)*– in this service model, resources are managed, aggregated and delivered to users as storage capacities (e.g., Amazon Simple Storage Service (S3) [Ama]), network mediums, or computing capabilities (e.g., Amazon Elastic Compute Cloud (EC2) [Inc08]). Users can then run any operating systems and software that best meet their requirements. Merely, they cannot manage or control the underlying cloud infrastructure.

- *Platform as a Service (PaaS)*– this service model provides a development environment or platform, on which users execute their applications. That is, users can customize applications that target a specific platform, with the tools offered by their Cloud Service Provider (CSP). They also have full control over their deployed applications and the associated configuration. Benefits of Paas are more prominent for startup enterprises. In fact, these small companies can develop and deploy their own applications and business software with no need for procuring servers and working teams to manage them. Google App Engine [Goo11] and Microsoft Azure Platform [Cha10] are good examples of PaaS service models.

- *Software as a Service (SaaS)*– in this model, the cloud provider offers software applications as a service. For instance, instead of buying and installing software on individual systems, clients use the proposed applications, in a pay per use basis. Access to these applications can be performed from various devices through either a client interface or a program interface. Along with provisioning services, the CSP engagement is to maintain a set of facilities, for ensuring software updating and managing a large scale cloud system. The proliferation of SaaS is illustrated by the increasing number of applications that are proposed, like Google Docs [AGJ^{+}08], Salesforce [chaf] and Dropbox [chad].

Along with these different service models, the portfolio of the proposed cloud services is continuously enriched. The proliferation of these services and the diversity of the cloud components and the complexity of architectures are raising new security and privacy challenges issues (e.g., virtualization). Threats, vulnerabilities and risks are usually associated to the technologies adopted in a specific environment. In cloud service models, vulnerabilities are associated to technologies, as well as to the main attributes of this environment. According to [XX12], there are three main challenges that are closely related to the cloud characteristics:

- *outsourcing* – outsourcing is delegating the responsibility for performing data storage or business functions to a third party. By outsourcing data, users remove the burden of establishing and maintaining a local storage infrastructure. However, outsourcing also means that users partially loose control on their data and tasks. Many cloud providers are not up to the level they should be in order to effectively guarantee a trustworthy security architecture. In fact, data may be read, altered or deleted, when outsourced. In the sequel, owners have to be aware of these confidentiality, integrity and privacy challenges. They also must worry about the availability of services, the error recovery of data and the business continuity. Thus, the control loss concern has become one of the root causes of cloud security challenges. Consequently, data and process security remains a dominating barrier to the development and widespread use of cloud storage. To deal with outsourcing security issues, the cloud provider has to provide trust and secure data storage. Moreover, outsourced data have to be protected, controlled and verified to ensure confidentiality, integrity and other security services. In case of external incident, outsourced data may incur privacy violations which should be assured by providers.

- *multi-tenancy* – multi-tenancy means that the cloud infrastructure is shared and used by multiple users. As such, in a virtual environment, data belonging to different users may be placed on the same physical machine, based on a certain resource allocation policy. Although multi-tenancy is an essential choice of cloud vendors due to its economic efficiency, it provides new vulnerabilities to the cloud platform. That is, malicious users may exploit this co-residence issue to perform flooding attacks [Zun12].

- *massive data* – the scale of data and applications grows exponentially and brings new challenges of dynamic data monitoring and security protection, such as image processing and data mining in the cloud context. That is, traditional security mechanisms are insufficient and inefficient, due to heavy computation and communication overhead. For example, in a cloud storage security setting, we cite the integrity verification of outsourced data. Thus, current technologies of privacy preservation are mainly based on static data sets, while data are always dynamically changed, including data patterns and variation of attributes and access rights. As such, new strategies and protocols are expected.

Various security challenges are widespread due to the joint emergence of several technologies and new concepts in order to provide new cloud objectives, such as virtualization [BSPN04, Cla05, CA10, AF12, MRSP12].

Table 1.1 summarizes the relation between security requirements, vulnerabilities and threats, in cloud environment. As with any remote storage system, there are principal security properties that are highly recommended in cloud storage, namely, confidentiality, integrity and freshness. These properties ensure that client data are secure and cannot be modified by unauthorized users. Moreover, data need to be protected when transferred and stored in cloud storage servers. As such, service providers have to ensure fine grained access, data availability claims, and effective data and process isolation which remains a major issue in cloud systems. We have also to underline the importance of the regulation and the legislation compliance, when outlining the different security requirements. That is, when stored, data may be transmitted through different cloud architectures, and then they might fall under different regulatory compliance restrictions which can give rise to Service Level Agreement (SLA) or privacy violations.

In Table 1.1, we consider five security attributes namely confidentiality, integrity, availability, accountability and privacy.

When dealing with clouds, confidentiality implies that client's data and computation tasks have to be kept secret from cloud providers and other unauthorized users. Confidentiality remains as one of the greatest concerns with regards to clouds, largely due to the loss of physical control. Similar to confidentiality, the notion of integrity in clouds concerns both data and process integrity. Data integrity implies that data should be honestly stored on cloud servers, and any violations (e.g. data are lost, altered or compromised) have to be detected. Computation integrity implies that programs are executed without being distorted by malware, cloud providers or other malicious users, and that any incorrect computation have to be detected.
Privacy is yet another critical concern with regards to cloud environments, due to the fact that clients' data reside among remote distributed servers, maintained by potentially untrusted cloud providers. Therefore, there are potential risks that the confidential data or personal information are disclosed to unauthorized entities. Obviously, in order to guarantee privacy preservation, confidentiality and integrity become essential, ensuring that data and computation are kept secret and uncorrupted. Contrary, accountability may undermine privacy since these two security attributes usually conflict. That is, accountability implies the capability to of identifying a party, with undeniable evidence. In fact, a fine-grained identity may be employed to identify a specific entity or even a malicious program.

TABLE 1.1 - Summary of Cloud security and privacy challenges

Requirements	Vulnerabilities	Threats
Confidentiality	VM co-residence	Cross-VM attacks [RTSS09, AHFG10]
	Loss of physical control	Data manipulation [ABC+07, ABC+11]
	Cloud user management	Vertical/Horizontal privilege attacks
Integrity	Loss of physical control	Data loss
		Dishonest computation [WRW11]
		SLA violation
Availability	Bandwidth under provisioning	Flooding attack [Zun12]
	Cloud Pricing Model	Fraudulent Resource Consumption attack
Accountability	Cloud Pricing Model	Inaccurate billing of resource consumption
	Loss of physical control	Hidden identity adversaries
Privacy	Loss of physical control	Privacy breach

1.2 Problem Statement and Objectives

Cloud data storage services bring many challenging design issues, considerably due to the loss of physical control. These challenges have significant influence on the data security and performances of cloud systems. That is, cloud data are often subject to a large number of attack vectors, as depicted in Table 1.1.

On one side, providing **data confidentiality**, in multi-tenant environments, becomes more challenging and conflicting. This is largely due to the fact that users outsource their data on remote servers, which are controlled and managed by possible untrusted Cloud Service Providers (CSPs). It is commonly agreed that data encryption at the client side is a good alternative to mitigate such concerns of data confidentiality [KL10,CGJ$^+$09]. Thus, the client preserves the decrypting keys out of reach of the cloud provider. Nonetheless, this approach gives rise to several key management concerns, such as, storing and maintaining keys' availability at the client side. In addition, the confidentiality preservation becomes more complicated with flexible data sharing among a group of users. First, it requires efficient sharing of decrypting keys between different authorized users. The challenge is to define a smooth group revocation which does not require updating the secret keys of the remaining users. So that, the complexity of key management is minimized. Second, the access control policies should be flexible and distinguishable among users with different privileges to access data. That is, data may be shared by different users or groups, and users may belong to several groups.

On the other side, the **data integrity** is considered as a relevant concern, in cloud environments. That is, the responsibility of securely managing outsourced data is splitting across multiple storage capacities. Such distribution provides resilience against hardware. Nonetheless, in order to reduce operating costs and save storage capacities, dishonest providers might intentionally slight these replication procedures, resulting in unrecoverable data errors or even data losses. Even when cloud providers implement a fault tolerant policy, clients have no technical means for verifying that their files are not vulnerable, for instance, to drive-crashes. There might be implementations of remote data checking at the three following levels:

- Between a client and a CSP – a cloud customer should have an efficient way to perform periodical remote integrity verifications, without keeping the data locally. Additionally, the client should also detect SLA violation, with respect to the storage policy. This customer's concern is magnified by his constrained storage and computation capabilities and the large size of outsourced data.

- Within a CSP – it is important for a cloud provider to check the integrity of data blocks stored across multiple storage nodes, in order to mitigate byzantine failures and drive-crashes.

- Between two CSPs – in the case of the cloud of clouds scenarios, where data are divided on different cloud infrastructures. Therefore, a CSP, through its cloud gate, should periodically verify the authenticity of data blocks hosted by another cloud platform.

These security concerns are even more important, as the European regulations will be more severe and inflexible, including further derogations to effectively protect personal data which are outsourced on remote servers. The EU General Data Protection Regulation (GDPR) is expected to be passed this year and takes effect beginning of 2015 [chab]. The US Skyhigh Networks company conducted a survey of over 7000 cloud services, and shows that actually only 1 in 100 cloud providers fulfills all the security requirements outlined by the new European regulation. As such, cloud providers will have serious work to ensure compliance with these new derogations [chaa].

To meet the aforementioned challenges, we set the following objectives:

- ***Objective A*** – improving data confidentiality in cloud storage environments while enhancing dynamic sharing between users. Indeed, the proposed security mechanisms should ensure both robustness and efficiency, namely the support of flexible access control, efficient user revocation and performances.
- ***Objective B*** – addressing the issue of provable data possession in cloud storage environments for data integrity verification support, following three substantial aspects: *security level, public verifiability,* and *performance*, and considering the limited storage and processing capacities of user devices.
- ***Objective C*** – implementing the proposed techniques using standards and widely deployed schemes, and validating their feasibility and impact on real hardware.
- ***Objective D*** – providing mathematical proofs of soundness and correctness of the proposed schemes.

1.3 Contributions

When defining the solutions for confidentiality and integrity of outsourced data, we take into consideration the following aspects: ease of deployment, robustness, supported flexibility when activating/deactivating security services and performances. The contributions of this dissertation are summarized as follows:

- *Contribution 1* – proposition of a cryptographic scheme for cloud storage, based on an original usage of ID-Based Cryptography (IBC) [KBL13]. First, the proposed scheme ensures better **data confidentiality**. That is, every client acts as a Private Key Generator (PKG) by computing an ID-based pair of keys to encrypt the data that he intends to store in the cloud. As such, data access is managed by the data owner. Second, by using a per data ID-based key, we provide a flexible sharing approach. Indeed, the distribution of decrypting keys between the client and the authorized users, does not reveal any information about the client's secret (*Objective A, Objective C*).

In order to alleviate the computation complexity at the client side, a possible refinement to our first contribution is introduced [KL14]. Indeed, [KL14] details a client-side deduplication scheme for cloud applications, based on a content hash keying approach.

- *Contribution 2* – definition of CloudaSec, a public key based solution for improving **data confidentiality** in cloud storage environments and enhancing dynamic sharing between users [KLEB14]. CloudaSec applies the convergent encryption mechanism on data contents. That is, the data owner uploads encrypted content to the cloud and seamlessly integrates the deciphering key encrypted into the metadata to ensure data confidentiality. In addition, CloudaSec integrates a conference key distribution scheme, based on parallel Diffie Hellman exchanges, in order to guarantee backward and forward secrecy. That is, only authorized users can access metadata and decipher the decrypting data keys. As such, user revocation is achieved without updating the private keys of the remaining users (*Objective A, Objective C*).

- *Contribution 3* – proposition of an efficient remote **data integrity** verification framework based on a fundamental arithmetic Euclidean Division (ED), adapted to limited storage capacities [KEML14]. The framework is demonstrated to be resistant against data privacy leakage within a Zero-Knowledge Proof System, in the key role of public verifiability and the privacy preservation support (*Objective B, Objective D*).

- *Contribution 4* – presentation of SHoPS, a novel Set-Homomorphic Proof of data possession Scheme, supporting the 3 levels of **data integrity** verification. SHoPS enables a verifier not only to obtain a proof of possession from the remote server, but also to verify that a given data file is distributed across multiple storage devices to achieve a certain desired level of fault tolerance. Indeed, we introduce the set homomorphism property, which extends malleability to set operations, such as union, intersection and inclusion (*Objective B, Objective D*).

1.4 Thesis Organization

This dissertation is divided into 2 parts described below.

Part I – Cloud Data Storage Confidentiality – this part focuses on data confidentiality preservation which becomes more complicated with flexible data sharing among a dynamic group of users. It requires the secrecy of outsourced data and an efficient sharing of decrypting keys between different authorized users. Besides, access control policies should be distinguishable among users with different granted privileges.

Chapter 2 – Cryptography in Cloud Data Storage Environments – we discuss research directions and technology trends to mitigate cloud data confidentiality issue. That is, we mainly focus on public key algorithms and several cryptographic primitives and we discuss the potential use of certain techniques in cloud environments. Then, we introduce practical techniques to validate proposed protocols, as cryptographic protocols can be vulnerable to attacks outside the scope of the existing formal analyses.

Chapter 3 – ID-Based Cryptography for Secure Cloud Data Storage – this chapter describes Contribution 1. The proposed prototype relies on the use of ID-Based Cryptography (IBC), where each client (data owner) acts as a Public Key Generator (PKG). That is, he generates his own public elements and derives his corresponding private key using a secret.

Chapter 4 – CloudaSec: A Public Key based Framework to handle Data Sharing Security in Clouds – this chapter presents Contribution 2. A Swift-based CloudaSec framework is introduced, where data sharing and deduplication are the main objectives of the solution. CloudaSec is a public key based solution applying the convergent encryption on the data file. That is, the data owner uploads encrypted content to the cloud, using a symmetric cryptographic algorithm. Then, he integrates the encrypted key, relying on the public key of the recipient(s), in metadata in order to preserve data confidentiality.

Part II – Cloud Data Storage Integrity – this part addresses the Proof of Data Possession, by cloud servers. In fact, cloud customer should have an efficient way to perform periodical remote integrity verifications, without keeping the data locally. This concern is magnified by the client's constrained storage and computation capabilities and the large size of outsourced data.

Chapter 5 – Remote Data Checking in Clouds – in practice, many sophisticated protocols have been proposed in the literature to address remote data checking concerns. This Chapter highlights the requirements that should be fulfilled by a PDP scheme and provides an overview on the approaches that have been proposed in the literature to address the verification of the authenticity of data stored on untrusted servers.

Chapter 6 – A Zero-Knowledge Scheme for proof of Data Possession in Cloud Storage Applications – this chapter details Contribution 3. We present a novel PDP model based on the well-known GPS scheme proposed by Girault et al. in [GPS06]. Hence, we extend the GPS scheme to the verification of the authenticity of files stored on untrusted servers in cloud platforms.

Chapter 7 – SHoPS: Set Homomorphic Proof of Data Possession Scheme in Cloud Storage Applications – this chapter presents Contribution 4. It addresses the PDP concern, while supporting the verification of several data blocks outsourced across multiple storing nodes. We propose a new set homomorphic proof of data possession, called SHoPS, which ensures the verification of aggregated proofs, under an interactive proof system.

Chapter 8 – Conclusions and Perspectives – this Chapter concludes the dissertation with a summary of contributions and presents the perspectives for future work.

Part I

Cloud Data Storage Confidentiality

Table of Contents

Chapter 2

Cryptography in Cloud Data Storage Environments

It is possible to build a cabin with no foundations, but not a lasting building

Isidor Goldreich - 1980

Contents

2.1 Introduction

Cloud Storage is an evolving paradigm, shifting the computing and storage capabilities to external service providers. Especially due to this loss of direct control on outsourced data, users are reluctant for adopting cloud services. The data security and privacy concerns are quite legitimate, given the latest mediated revelations. That is, in November 2013, the Washington Post points more indiscriminate data capture, by the US National Security Agency (NSA), than even the PRISM revelations suggest. This collection is done by intercepting private links that connect Google and Yahoo data centers around the world – and decrypting the traffic that should be protected in transit [1] [nsaa]. In addition, in December 2013, a PriceWaterhouseCoopers (PwC) survey revealed that 54 percent of German companies find the cloud risky after learning of NSA spying [nsab]. Therefore, several security measures have to be set up, in order to cope with the emerged cloud concerns, namely outsourcing encrypted data and periodically checking data integrity and availability. However, the choice of effective security mechanisms has to take into consideration peripheral challenges. For example, storing encrypted data yields to a cumbersome key management and access control, and regularly checking huge amounts of data tightens the bandwidth consumption.

In this chapter, we provide an introductory compendium to some cryptographic techniques for ensuring security and privacy in clouds. Cloud security concerns have emerged to be of an increasing interest and importance, in the applied cryptography and computer research community, while demanding adequate measures for cloud challenges. For this purpose, we consider a storage scenario, where the client outsources data in remote servers. The cloud servers act as distributed black boxes, from the client's perspective. The techniques presented in this chapter are partially to be used at the service provider's side or the client's side, but in any case should protect the interests of both, to establish a successful and trustworthy service.
Next, we give an overview of cryptography fundamentals in Section 2.2. Then, we investigate the use of cryptographic mechanisms in cloud storage environments in Section 2.3, before presenting formal security models (Section 2.4).

2.2 Fundamentals on Cryptography

For many years, cryptography was the exclusive domain of military, diplomatic and governmental secret services, and has been used to mainly provide security properties, such as data confidentiality, data integrity and data origin authentication [ISO89]. Presented as the art of coding information into secrets, cryptography enables the intended receivers to recover the original content of messages. During the second part of the twentieth century, the field of cryptography has expanded due to the proliferation of computers and networks and the appearance of new cryptographic systems. In 1976, Diffie Hellman conducted radical changes in cryptography, presenting the first asymmetric cryptographic

[1]Data in transit is commonly delineated into two primary categories – data that is moving across public or untrusted networks such as the Internet, and data that is moving within the confines of private networks such as corporate Local Area Networks (LANs).

algorithm [DH76]. In 1978, Rivest, Shamir and Adelman defined their well-known RSA algorithm [RSA78]. Then, Shamir continued publishing revolutionizing ideas, namely, threshold schemes, ID-based cryptographic systems and privacy homomorphisms. Concurrently, Koblitz and Miller independently proposed novel cryptographic schemes based on elliptic curve structures [Kob87, Mil86]. Recently, quantum cryptography appears as the cryptography of the future, as it does not rely on abstract algebra and groups theory, but on optic and light theories, where every bit is represented by the polarization of a photon. In this section, we first present the general concepts of symmetric cryptography (Section 2.2.1). Then, we describe in depth public key cryptography (Section 2.2.2).

2.2.1 Symmetric Cryptography

A fundamental distinction between cryptographic schemes refers to the relation between the pair of keys, involved in message encryption and decryption algorithms. Symmetric or conventional cryptography relies on the share of a secret key between two communicating entities Alice and Bob. Symmetric cryptography, as well as asymmetric cryptography, is based on using two related algorithms for message encryption and decryption.
Let $\mathcal{C}$ be the ciphertext message space, $\mathcal{M}$ the plaintext message space and $\mathcal{K}$ the key space. We denote the encryption algorithm by E, and the decryption algorithm by D, defined as follows:

- The encryption algorithm $E : \mathcal{M} \times \mathcal{K} \longrightarrow \mathcal{C}$ takes as input the plaintext message m, and the secret key k, and returns the ciphertext c. E is often randomized.
- The decryption algorithm $D : \mathcal{C} \times \mathcal{K} \longrightarrow \mathcal{M}$ takes as input the ciphertext message c, and k, and returns m. D is always deterministic.

We say that a symmetric cryptographic scheme is well defined, if it fulfills Equation 2.1:

$$\forall m \in \mathcal{M}, k \in \mathcal{K}, \qquad D(E(m, k), k) = m \tag{2.1}$$

The Vernam one-time pad [Ver19] is one of the well known symmetric algorithms. It was proposed by Gilbert Vernam, in 1917. Vernam supposes that the secret key k is a random bit string as long the message m. As depicted in Figure 2.1, when Alice wants to encrypt a message m, using the shared secret key k, with Bob, she computes $c = E(m, k) = m \oplus k$. Thus, Bob uses the same key k to decrypt the received ciphertext c as $m = D(c, k) = c \oplus k$.

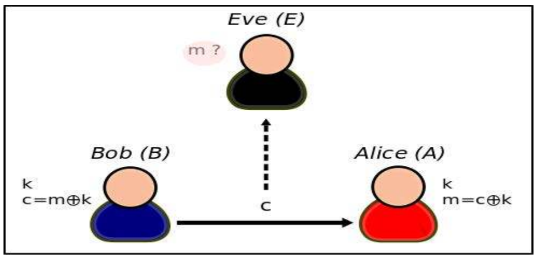

FIGURE 2.1 - Vernam encryption scheme

The Vernam's encryption scheme is called one-time pad as the secret key k is used once for enciphering a unique message. Therefore, the key has to be renewed for every message. One-time pads are *perfectly secure* in that the encrypted message provides no information about the original message to a cryptanalyst. This is a very strong notion of security first developed by Claude Shannon and proved, mathematically, to be true for the one-time pad by Shannon about the same time.

Shannon first introduces the perfect secrecy definition of a cipher, in 1949, as detailed in Definition 2.2.1 [Sha49].

Definition 2.2.1 ***Perfect Secrecy*** –
Let (E, D) be a cipher defined over $(\mathcal{M}, \mathcal{C}, \mathcal{K})$. We say that (E, D) has a perfect secrecy if:

$$\forall m_0, m_1 \in \mathcal{M}, (|m_0| = |m_1|), \textit{ and } \forall c \in \mathcal{C}, \qquad Pr[E(m_0, k) = c] = Pr[E(m_1, k) = c]$$

where k is uniformly chosen in $\mathcal{K}$ ($k \xleftarrow{R} \mathcal{K}$).

The perfect secrecy definition is twofold. First, it induces the indistinguishably property of an encryption scheme. In fact, an eavesdropper Eve, intercepting the communication channel between Alice and Bob, cannot distinguish between the encryption of m_0 from the encryption of m_1. Second, Definition 2.2.1 states that the most powerful adversaries cannot recover information from the enciphered message.
Shannon proved that the Vernam one-time pad algorithm ensures the perfect secrecy as follows.

Proof.
On the one side, we know that $\forall m \in \mathcal{M}, c \in \mathcal{C}, Pr_k[E(m, k) = c] = \frac{\sharp\{k \in \mathcal{K}, E(m,k) = c\}}{|\mathcal{K}|}$, where k is uniformly chosen in $\mathcal{K}$. As such, if the set $\sharp\{k \in \mathcal{K}, E(m, k) = c\}$ is constant, we say that the cipher has a perfect secrecy.
On the other side, for the one-time pad algorithm, $\forall m \in \mathcal{M}, c \in \mathcal{C}, E(m, k) = m \oplus k = c$. As such, $k = m \oplus c$. In the sequel, the set $\sharp\{k \in \mathcal{K}, E(m, k) = c\}$ is constant and it is equal to 1.
Therefore, the one-time pad is perfectly secure. □

Despite the different strengths of the Vernam algorithm, it reveals many drawbacks. That is, in order to support strong secrecy, Shannon proved that the length of the secret key $|k|$ has to be equal or greater than that of the message $|m|$ [Sha49]. In addition, we recall that the key has to be renewed for every message. As such, Alice and Bob must maintain a secure communication channel to exchange a new secret key for every transmitted message. This is obviously not feasible in practice because Alice and Bob will be wasting half of their communication time in exchanging keys.
In order to settle the problems of the Vernam algorithm, new types of symmetric encryption schemes appeared, referred to as block ciphers. These algorithms encrypt blocks of data, using small keys of pre-fixed bit lengths. Block ciphers mainly rely on permutations. The most famous algorithms are the Data Encryption Standard (DES) [NIS99], and the Advanced Encryption Standard (AES) [FIP01]. This latter is widely deployed by several

cloud service providers, such as the US provider Amazon Simple Storage Service S3 [Ama].

Cloud storage researchers are giving more focus on data security, while considering the impact of the proposed algorithms on cloud performances. Thus, modern symmetric encryption algorithms join several cloud security requirements, namely, cloud availability and compliance. In fact, these conventional schemes are typically fast and computationally less intense than asymmetric algorithms. Therefore, they are suitable for processing large streams of outsourced data.
However, symmetric encryption algorithms presume that Alice and Bob are able to exchange the key in a secure manner prior to each communication. As such, the key management is a significant challenge in a multi-tenant environment, especially for Security as a Service models (SecaaS). So that, secret key schemes are usually mixed with public key algorithms to obtain a blend of security and speed.

2.2.2 Public Key Cryptography

Public Key Cryptography (PKC) ensures several security properties, namely data confidentiality, non repudiation, and authentication, while exchanging information over an insecure channel. Contrary to symmetric cryptography where two communicating entities have to share the same secret key, public key cryptography relies on two related keys to secure the exchanged information. The pair of keys consists of a *public* key and a *private* key, where each entity shares its public key with its peers. However, the private key is kept secret, as depicted in Figure 2.2.

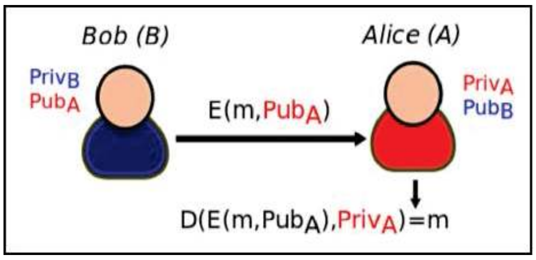

Figure 2.2 - Public Key Cryptography (PKC)

This pair of keys is defined over a mathematical relation. Solving this mathematical equation comes to breaking a hard computational problem. In fact, the field of public key cryptography has motivated a number of complex computational problems, such as the integer factorization related to the well-known RSA cryptosystem [RSA78]. The supposition that these problems are in general computationally difficult to resolve in polynomial time defines the most contemporary cryptographic systems. By computationally infeasible, we mean that a computationally bounded adversary $\mathcal{A}$ has only a *negligible* chance succeeding in calculating D given E, with respect to some pre-defined security parameter ξ . In several works, ξ is selected to be the bit-length of the public parameters of a given cryptographic scheme (Definition 2.2.2).

Definition 2.2.2 *A function $\tau : \mathbb{N} \rightarrow \mathbb{R}$ is said to be negligible if for any non zero polynomial $p \in \mathbb{R}[x]$, there exists $m \in \mathbb{N}$ such that:*

$$\forall n > m, |\tau(n)| < \frac{1}{|p(n)|}$$

In practice, a Public Key Infrastructure (PKI) is deployed and a Certification Authority (CA) is used to securely bind public keys to their related entities. The CA is a trusted third party which signs the certificate containing the public key and the identification information of a user.
We next present the first public key scheme, appeared in the Diffie Hellman seminal paper in 1976 [DH76], based on the multiplicative groups' theory (Section 2.2.2.1). Then, we introduce the Elliptic Curve Cryptography (ECC) [HMV03] which relies on additive groups derived from elliptic curve structures (Section 2.2.2.2). Finally, we enumerate some functional challenges, due to the application of traditional cryptographic mechanisms in cloud storage environments (Section 2.2.2.3).

2.2.2.1 Diffie Hellman Algorithms

Whitfield Diffie and Martin Hellman proposed the Diffie Hellman algorithm (DH), in 1976. This algorithm is now celebrating its 38^{th} anniversary while it is still playing an active role in Internet protocols today.
DH is a mechanism for securely exchanging a shared secret between two communicating entities, Alice and Bob, in real time over an untrusted network. A shared secret is important between two parties that may not have ever communicated previously. The public elements provided to each party is a large prime p and a nonzero generator g of $\mathbb{Z}^*{}_p$.

First, Alice and Bob generate, respectively, their public values Pub_A and Pub_B from their secret keys $Priv_A$ and $Priv_B$. The private keys are randomly chosen in $\mathbb{Z}^*{}_{p-1}$. And, the public values satisfy the following relation:

$$\forall i \in \{A, B\}, \qquad Pub_i \equiv g^{Priv_i}(mod[p]) \tag{2.2}$$

After exchanging these public values, Alice and Bob use their private keys, again to calculate $Pub_A{}' \equiv Pub_B{}^{Priv_A}(mod[p])$ and $Pub_B{}' \equiv Pub_A{}^{Priv_B}(mod[p])$, respectively. These computed values, $Pub_A{}'$ and $Pub_B{}'$, are the same, since:

$$Pub_B{}^{Priv_A}(mod[p]) \equiv g^{Priv_B{}^{Priv_A}}(mod[p]) \equiv g^{Priv_A{}^{Priv_B}}(mod[p]) \equiv Pub_A{}^{Priv_B}(mod[p]) \tag{2.3}$$

The main weakness of the DH algorithm is the Man In the Middle attack (MIM). That is, Eve can create a shared secret with Alice and Bob by impersonating as Bob to Alice side and as Alice to Bob side. Mitigation of MIM attacks is possible by making Alice and Bob signing their selected public elements.
Although the Diffie Hellman key exchange algorithm provides a method for publicly sharing a random secret key, it does not achieve the full goal of being a public key cryptographic system, as a cryptosystem permits exchanging messages, not just a random string of bits.

The first public key cryptographic scheme following the Diffie Hellman construction is a system proposed by Taher ElGamal in 1985 [Elg85].
The DH security is based on the definition of two computational problems, namely the Diffie Hellman Problem (DHP) and the Discrete Logarithm Problem (DLP), detailed in Section 2.4.1.

2.2.2.2 Elliptic Curve Cryptography (ECC)

Elliptic Curve Cryptography (ECC) was discovered in 1985 by Victor Miller (IBM) and Neil Koblitz (University of Washington) as an alternative mechanism for implementing public key cryptography. ECC algorithms rely on the algebraic structure of elliptic curves over finite fields.
Elliptic curves (EC) are cubic forms that are defined over finite fields, generally a prime or a binary field denoted $\mathbb{F}_p$ or $\mathbb{F}_{2^p}$, where p and 2^p represent the order of the field respectively. By order, we mean the number of elements of the finite field. In this thesis, we only consider elliptic curves which are defined over finite prime fields, fulfilling the Weistrass equation [HMV03].
The set of points of an elliptic curve $E(\mathbb{F}_p)$, with the binary operation $+ : E(\mathbb{F}_p) \times E(\mathbb{F}_p) \rightarrow E(\mathbb{F}_p)$ forms an additive abelian group $(E(\mathbb{F}_p), +)$. That is, the binary operation of the group is the *addition* of two points of the curve. The elliptic curve $E(\mathbb{F}_p)$ is said to be well defined (smooth) if its discriminant $\triangle$ is different from 0. This latter ensures that the elliptic curve does not contain singular points for whom the addition cannot be defined. This abelian group has to fulfill four properties, namely the associativity, the commutativity, the existence of inverses and the identity element. The identity element is a rational element called the *point at infinity* P_∞ [HMV03].

The Diffie-Hellman algorithm is widely adapted to elliptic curve groups. As explained in Section 2.2.2.1, Alice and Bob have to exchange their public elements Pub_A and Pub_B, respectively, to compute the shared key.
In fact, let $\mathbb{G}$ be a subgroup of $E(\mathbb{F}_p)$. The additive group $\mathbb{G}$ is generated by the point P of prime order n. Alice and Bob choose random integers $Priv_A \in \mathbb{Z}^*{}_n$ and $Priv_B \in \mathbb{Z}^*{}_n$, as their private keys. Then, each communicating entity computes the related public key as follows:

$$\forall i \in \{A, B\}, \qquad Pub_i \equiv [Priv_i]P$$

That is, the problem of finding the private key given the generator P and the published public key denotes the *Elliptic Curve Discrete Logarithm Problem* (ECDLP), presented in Section 2.4.1.

Then, after exchanging the public elements, Alice and Bob calculate the shared secret as follows:

$$[Priv_A]Pub_B = [Priv_A.Priv_B]P = [Priv_B.Priv_A]P = [Priv_B]Pub_A \tag{2.4}$$

The security level is a recurrent concept in cryptography. It permits to evaluate the hardness of breaking an encryption or a signature algorithm. That is, the longer the level

of security is, the harder the cryptanalysis of the algorithm becomes. The security level of a symmetric encryption scheme is defined as the number of operations needed to crack an algorithm using a key of k-bits length. In fact, the number of elementary operations needed to break a symmetric encryption algorithm is 2^k. As such, the current key size recommendation for symmetric schemes is equal to 112 bits [chae].
Regarding the asymmetric cryptography, the security level of an algorithm sets the length of RSA and EC keys with respect to the hardness of breaking a mathematical computational problem, such as, factoring integers in the case of RSA or solving the DLP and DH problems in the case of DH algorithms. Table 2.1 depicts the equivalence between the lengths of RSA and EC keys respectively to the security level l_k, where l_k presents the length of a symmetric key k [MVO96].

TABLE 2.1 - RSA and EC key sizes for equivalent security level

Security level l_k	80	112	128	192	256
EC key length (bits)	160	224	256	384	512
RSA key length (bits)	1024	2048	3072	7680	15360

Table 2.1 shows that the use of EC keys is more interesting than RSA keys, in public key cryptography. That is, for the same security level, the 2048 bits current key-size recommendation is offered by a hugely smaller 224 bits EC keys. In favor of ECC, this advantage significantly increases with the security level. The use of elliptic curve cryptographic schemes becomes fascinating as stronger security mechanisms become mandated and devices get smaller. In addition, the usage of ECC takes an expanding interest as the implementation of elliptic curves requires less storage and computation capacities.

2.2.2.3 Limits of Traditional Cryptographic Systems in Clouds

Despite traditional cryptographic systems provide strong security guarantees, they may be inadequate for modern storage systems. In fact, several limitations reduce the account of these traditional schemes, especially due to the huge amounts of outsourced data.
In cloud storage environments, bandwidth, memory and power consumptions are a big concern, as they impact the availability and performances of delivered services. Consequently, the selection of adequate cryptographic tools for security support is accurate.

First, under an untrusted service provider security model, the client generally chooses to encipher data before outsourcing to remote servers. Thus, the usage of traditional asymmetric algorithms is overly cumbersome for large amounts of data, and computation capacities at the client side are significantly reduced, even by using elliptic curves.

Second, classical asymmetric algorithms require deploying PKI and certificate management functions for the generation and delivery of certificates to authenticated entities. In addition, the periodic downloading of revocation lists by the clients from the Certification Authority (CA) is necessary to verify the validity of certificates. Thus, the bandwidth consumption, and then, availability requirements are deteriorated.

Third, while using symmetric cryptographic schemes to encipher data at the client side, this latter preserves the decrypting keys out of reach of the service provider. However, the confidentiality provision becomes more complex with flexible data sharing among a group of users. That is, it requires efficient sharing of decrypting keys between different authorized users. The challenge is to define a smooth group revocation which does not require updating the secret keys of the remaining group members. Therefore, the complexity of key management is minimized.

Finally, as these traditional cryptographic tools are mostly deterministic, they are not malleable and do not allow operations over encrypted data, such as the search over enciphered texts. Search is a convenient method for retrieving outsourced data information on remote servers. Hence, several applications, that index data, have emerged to allow quick search, namely, *Apple Spotlight* and *Google Desktop*.

2.3 Cryptographic Mechanisms in Clouds

Modern cryptography provide much more flexible decryption mechanisms, and explicitly allow malleability on ciphertexts, namely search over encrypted data, Proofs of Data Possession (PDP) and Proofs of data Retrievability (PoR). These promoting approaches are greatly interesting in a multi-tenant cloud environment. PDP and PoR concepts will be investigated, in depth, in Chapter 5.
In the next section, we introduce emerging asymmetric schemes. First, we introduce the Identity Based Cryptography (IBC), in section 2.3.1. In IBC, the public key of an entity is directly derived from its identity, with no need for certificates. Then, we briefly present the Attribute Based Cryptography (ABC), in section 2.3.2 and the homomorphic cryptography, in section 2.3.3, allowing several mathematical operations on enciphered data.

2.3.1 Identity Based Cryptography

In 1984, ID-Based Cryptography (IBC) was introduced by Shamir [Sha85] with the original idea to provide public and private key pairs with no need for certificates and CA deployment. Shamir assumes that each entity uses one of its identifiers as its public key. These identifiers have to be unique. In addition, he assigns the private key generation function to a special entity called the Private Key Generator (PKG). That is, before accessing the network, every entity has to contact the PKG to get its private key. This private key is computed so as to be bound to the public key of the entity. During the last decade, IBC has been enhanced by the use of the Elliptic Curve Cryptography (ECC) [HMV03]. As a consequence, new ID-based encryption and signature schemes emerged. These new schemes differ from Shamir approach relying on smart cards to store the private keys of users and the ciphering information.
In 2001, Boneh and Franklin [BF01] proposed the first ID-based encryption scheme, relying on the use of bilinear pairing functions to bind elliptic curve points to a number of a multiplicative group.
We note that certificates may be considered as an identity based feature, as they map the user's public key to his identity. In this dissertation, we focus on identity based schemes

where the public key is computationally derived from the user identity. This public key is generally considered as the output of hash function that takes as input the identity of the user.

In the following sections, we first present the pairing functions that were widely used in modern cryptographic systems, thanks to its interesting properties in section 2.3.1.1. Then, we introduce the key generation process for ID-based schemes, which are based on pairing functions.

2.3.1.1 Prerequisites on Pairing Functions

The pairing function $\hat{e}$ is a bilinear map, non degenerate and efficiently computable. That is, the pairing map has to verify the following properties:

- *Bilinearity* – the pairing function $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ is linear with respect to each of its inputs as follows:

$$\forall P \in \mathbb{G}_1, \forall Q \in \mathbb{G}_2 \text{ and } \{a, b\} \in \mathbb{Z}^2, \qquad \hat{e}(a.P + b.P, Q) = \hat{e}(P, Q)^a \hat{e}(P, Q)^b$$

$$\forall P \in \mathbb{G}_1, \forall Q \in \mathbb{G}_2 \text{ and } \{a, b\} \in \mathbb{Z}^2, \qquad \hat{e}(P, a.Q + b.Q) = \hat{e}(P, Q)^a \hat{e}(P, Q)^b$$

- *Non degeneracy* – this property defines two relations as follows:

$$\forall P \in \mathbb{G}_1, \hat{e}(P, Q_\infty) = 1_{\mathbb{G}_T}$$

$$\forall Q \in \mathbb{G}_2, \hat{e}(P_\infty, Q) = 1_{\mathbb{G}_T}$$

  Let us consider a generator P of $\mathbb{G}_1$ and a generator Q of $\mathbb{G}_2$, so that, the value $\hat{e}(P, Q)$ is equal to the generator of $\mathbb{G}_T$.

- *Efficiency* – the efficiency property means that there is an algorithm that computes the pairing function.

Pairing functions can be divided into symmetric and asymmetric functions. The symmetric pairings require the same input group $\mathbb{G}_1 = \mathbb{G}_2$, while the asymmetric functions verify $\mathbb{G}_1 \neq \mathbb{G}_2$ [HMV03].
In practice, bilinear maps are generally derived from the well known Weil or Tate pairing [BSSC05].

2.3.1.2 ID-Based Key Construction

In order to be able to derive a client's private key, the PKG first defines a set of *ID-based public elements* (IBC–PE). The PKG generates the groups $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ and the pairing function $\hat{e}$ from $\mathbb{G}_1 \times \mathbb{G}_2$ in $\mathbb{G}_T$. $\mathbb{G}_1$ and $\mathbb{G}_2$ are additive subgroups of the group of points of an Elliptic Curve (EC). However, $\mathbb{G}_T$ is a multiplicative subgroup of a finite field. $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ have the same order q. In addition, $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ are generated by P, Q and the generator $g = \hat{e}(P, Q)$, respectively. The bilinear function $\hat{e}$ is generally derived from the Weil or Tate pairing [BSSC05, BF01].

After the specification of the groups, the PKG defines a set of hash functions in accordance to the ID-based encryption and signature schemes in use [RRP04]. For example, the PKG defines a hash function $Hash_{pub}()$ to transform the client's identity (ID) into a public key as follows:

$$Pub_{ID} = Hash_{pub}(ID) \tag{2.5}$$

Generally, the public key of a client is computed as a hash of one of his identities and it is either a point of an elliptic curve [BF01] or a positive integer [SK03].

The PKG generates the private key of an entity using a local secret $s_{PKG} \in \mathbb{Z}_q^*$ and a private key generation function $PrivGen()$. Note that the private key is computed as:

$$Priv_{ID} = PrivGen(s_{PKG}, Pub_{ID}) \tag{2.6}$$

For example, Boneh and Franklin [BF01] compute the private key as $Priv_{ID} = s_{PKG}.Pub_{ID}$, where Pub_{ID} is a point $\in \mathbb{G}_1$. However, Sakai and Kasahara [SK03] generate the private key as $Priv_{ID} = [1/(Pub_{ID} + s_{PKG})].P$, where Pub_{ID} is an integer. Differently, Boneh and Boyen define a new key derivation function [BB04]. That is, they first compute three public points as $P_1 = \alpha.P$, $P_2 = \beta.P$ and $P_3 = \gamma.P$, where α, β and γ are secrets selected by the PKG. The Boneh and Boyen key derivation scheme computes the user private key as a couple of elliptic curve points $Priv_{ID} = (Priv_1, Priv_2) = (Pub_{ID}.r.P_1 + \alpha.P_2 + r.P_3, r.P)$, where the random $r \in \mathbb{Z}^*$.

The groups $\mathbb{G}_1$ and $\mathbb{G}_2$, the pairing $\hat{e}$, the points P, Q and $Q_{pub} = s_{PKG}.Q$, and the hash functions $H_1(), ..., H_k()$ form the *ID-based public elements*, as follows:

$$IBC - PE = \{\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, \hat{e}, g, P, Q, Q_{pub}, Hash_{pub}(), H_1(), ..., H_k()\} \tag{2.7}$$

After generating a private key, the PKG has to secure its transmission to the related owner either using cryptography or directly to the physical person (using a secure transportation device). In the aforementioned key derivation schemes, the PKG computes the private keys of entities. Thus, the PKG is able to impersonate as any of them by illegally deciphering encrypted data files. This attack is known as the *Key Escrow Attack* (KEA). In order to mitigate that KEA, a strong assumption is usually made necessary that the PKG is a trustworthy entity.

2.3.1.3 Examples of ID-Based Encryption Schemes

In this section, we describe three ID-Based encryption schemes, relying on different key derivation algorithms. First, we introduce Boneh and Franklin encryption algorithm [BF01]. Then, we present Boneh and Boyen encryption scheme [BB04]. Finally, we describe Chen et al. scheme [CCMLS06] which relies on Sakai-Kasahara key construction [SK03].
The security of these schemes is based on the hardness of the *Bilinear Diffie-Hellman* (BDH) problem which consists on calculating $\hat{e}(P, P)^{abc}$, given the public points $P, a.P, b.P$ and $c.P$ and the symmetric pairing function $\hat{e}$.

- ***Boneh and Franklin encryption scheme*** – the Boneh and Franklin IBE scheme uses a symmetric pairing function. Thus, the public elements are defined as IBC–PE=$\{\mathbb{G}_1, \mathbb{G}_T, q, \hat{e}, g, P, Q, Q_{pub}, Hash_{pub}(), H_1()\}$, where $H_1 : \mathbb{G}_T \rightarrow \{0,1\}^n$. The PKG computes the user's public key, using a hash-to-point function, as $Pub_{ID} = Hash_{pub}(ID)$. Then, it generates the related private key as $Priv_{ID} = s_{PKG}.Pub_{ID}$. To encrypt a message $M \in \{0,1\}^n$ using the public key Pub_{ID}, the sender generates a random r and computes C such that $C = (U, V) = (r.P, M \oplus H_1(\hat{e}(Pub_{ID}, Q_{pub})^r))$. The recipient user decrypts the message as $M = V \oplus H_1(\hat{e}(Priv_{ID}, U))$.

- ***Boneh and Boyen encryption scheme*** – relying on a symmetric pairing, Boneh and Boyen define two hash functions H_1 and H_2 as $H_1 = \{0,1\}^* \rightarrow \mathbb{Z}_q^*$ and $H_2 = \mathbb{G}_T \rightarrow \{0,1\}^n$. So that, the Boneh and Boyen public elements are $\{\mathbb{G}_1, \mathbb{G}_T, q, \hat{e}, g, P, u, P_1, P_2, P_3, H_1(), H_2()\}$, where $u = \hat{e}(P_1, P_2)$ computed by the PKG after the generation of the public points (Section 2.3.1.2).
  To encrypt a message $M \in \{0,1\}^n$ using the public key Pub_{ID}, the sender generates a random r and computes C such that $C = (U, V, W)$, where $U = M \oplus H_2(u^r)$, $V = r.P$ and $W = Pub_{ID}.r.P + r.P_3$. The recipient user first computes $k = \frac{\hat{e}(V, Priv_1)}{\hat{e}(V, Priv_2)}$, where $Priv_{ID} = (Priv_1, Priv_2)$ (Section 2.3.1.2). Then, he recovers the message M as $M = U \oplus H_2(u^k)$.

- ***Chen et al. encryption scheme*** – Chen et al. presented an IBE scheme using a symmetric pairing function. They define two hash functions H_1 and H_2 as $H_1 = \{0,1\}^* \rightarrow \mathbb{Z}_q^*$ and $H_2 = \mathbb{G}_T \rightarrow \{0,1\}^n$. The key derivation procedure follows the Sakai-Kasahara algorithm (Section 2.3.1.2).
  To encrypt a message $M \in \{0,1\}^n$ using the public key Pub_{ID}, the sender generates a random r and computes $C = (U, V)$, where $U = r.(Q_{pub} + Pub_{ID}.P)$ and $V = M \oplus H_2(g^r)$. The recipient user recovers the encrypted message as $M = V \oplus H_2(\hat{e}(U, Priv_{ID}))$.

2.3.1.4 IBC in Cloud Data Storage Security

The application of ID-Based Cryptography, in a distributed environment, is an emerging and interesting area, which has been partially investigated in the literature. IBC was first adapted to grid networks. Ian Foster [Fos02] defines the grid as a flexible system that coordinates resource sharing among individuals and institutions, using standard and open protocols, in order to achieve a common goal. Recently, Ian Foster et al. [Fos09] show that grids tightly join clouds in technology and architecture, but they differ in several aspects such as security models. The idea of applying IBC to grid security was explored by Lim and Robshaw in 2004 [LR04]. In their proposal, each virtual organization has its own PKG, and all of its users share the same IBC–PE certified by a grid certification authority. Their scheme offers to the encrypting entity more flexibility during the key generation process, and permits to add granularity to the ID-based public key. In fact, Lim and Robshaw propose to include the security policy into the identifier used as input for the public key computation algorithm. However, their proposal has two drawbacks. First, the user needs to maintain an independent secure channel with the PKG for the retrieval of his private key. Second, the PKG is able to perform a key escrow attack, due to its knowledge of the clients' private keys.

Then, Lim and Robshaw [LR05] introduced a new concept of dynamic key infrastructure for grid, to simplify the key management issues listed in [LR04]. That is, Lim and Robshaw proposed a hybrid approach combining identity based mechanisms at the client level, and traditional PKI to support key management above the client level. In [LR05], each user distributes a fixed parameter set through a $X.509$ certificate. This parameter allows the other users to act as their own trusted authorities for the purposes of delegation and single sign-on. Therefore, they remove the need for a proxy certification. On one hand, this technique avoids the key escrow attack and the need for a secure channel for private key distribution in an ID-based system. Unfortunately, users have to support the cumbersome task of verifying the parameter sets of other entities. In addition, this paper does not address the arising risk of Man In the Middle attacks [SDJ+10].

In 2005, Lim and Paterson [LP11] proposed to use IBC in order to secure a grid environment. They describe several scenarios in which IBC simplifies the current grid solutions, like the elimination of the use of certificates, simple proxy generation, easy revocation of proxy certificates and the savings of bandwidth by using the pairing based approach proposed by Boneh and Franklin in [BF01].
In the same way, Li et al. [LDTY09] propose to use IBC as an alternative to the SSL authentication protocol in a cloud environment. That is, [LDTY09] introduces an identity based hierarchical model relying on three levels. The top level is the root PKG and it corresponds to the cloud administrator. The second level presents a sub-PKG corresponding to a cloud data center, while the third level is presented by any cloud client. As such, each client public key is derived from the concatenation of a set of identities in the hierarchical model. Obviously, this scheme suffers from the needed trust hierarchy to ensure a secure working system.

Recently, Schridde et al. [SDJ+10] presented a novel security infrastructure, using IBC, for service-oriented cloud applications to overcome the problems of certificate based solutions. In their proposal, each client has to be registered at a corresponding authority server. The registration includes specifying how to pay for the desired service and retrieving login credentials for the corresponding account. During the registration, each client has a unique private identity key for the chosen account. This key serves as a mapping between the client identity and the allowed services. Although [SDJ+10] relieves the burden of maintaining a trusted environment, the usage of a unique identity arises the problem of sharing data among a dynamic group of users.

2.3.2 Attribute Based Cryptography

In 2005, Sahai and Waters introduced the concept of Attribute Based Cryptography (ABC) [SW05], as a new mean for encrypted access control. In ABC, ciphertexts are not necessarily encrypted to one particular user as in traditional public key cryptography. Instead both users' private keys and ciphertexts are associated with a set of attributes or a policy over attributes. The user is able to decrypt a ciphertext if there is a *match* between his private key and the ciphertext.

This section gives a brief overview of ABC, as this cryptographic system is not addressed in this dissertation. However, it was interesting to introduce ABC, given its attractive features in cloud data storage environments. Interested readers may refer to [LCH13] for more details.

In [SW05], Sahai and Waters presented a threshold Attribute Based Encryption (ABE) scheme. That is, ciphertexts are labeled with a set of attributes $\mathcal{S}$ and a user private key is associated with both a threshold parameter t and another set of attributes $\mathcal{S}'$. In order to decrypt enciphered data, at least t attributes must match between the ciphertext and the user private key. One of the primary original motivations for this work was to design an error-tolerant (*Fuzzy*) identity-based encryption scheme that could use biometric identities.
In 2006, Goyal et al. proposed a key-policy attribute based encryption (KP-ABE) scheme [GPea06] that built the access policy into the user private key. That is, ciphertexts are labeled with sets of attributes and private keys are associated with access structures that control which ciphertexts a user is able to decrypt.
KP-ABE schemes ensure flexible and fine grained access control. In fact, data are outsourced in an encrypted form, while different users are still allowed to decrypt different pieces of data per security policy. This effectively eliminates the need to rely on the storage server for preventing unauthorized data access. However, the disadvantage of KP-ABE is that the access policy is built onto one user private key. As such, the data owner cannot choose who can decrypt the data except choosing a set of attributes which can describe the outsourced data. In addition, the sender must trust that the key-issuer issues the appropriate keys to grant or deny access to the appropriate users.
In 2007, Bethencourt et al. presented the first construction of a ciphertext policy attribute based encryption (CP-ABE) scheme [BSW07]. In their scheme, the user secret key is associated with a set of attributes, and the ciphertext is associated with an access policy over attributes. The user can decrypt the ciphertext if and only if the attribute set of his secret key satisfies the access policy specified in the ciphertext. [BSW07] is conceptually closer to traditional access control methods such as *Role Based Access Control* (RBAC).

Attribute based Cryptography (ABC) is referred to as an innovative concept and one of the most attractive way to manage and control file sharing in cloud, thanks to the computation properties on attributes. In fact, traditional access control architectures generally assume that remote servers storing the data are fully trusted by their clients. So that, they are often responsible for defining and enforcing access control policies. However, this statement does not usually hold in multi-tenant cloud data storage environments, especially due to the abstract nature of this business model. Consequently, cloud clients are still reluctant, while outsourcing their data file contents.

ABC is considered as a promotive solution, to ensure fine grained access control to data, which are outsourced on untrusted storage servers. First, ABC allows searching over encrypted data. That is, the ciphertext is assigned with a set of descriptive attributes. Thus, viewing these attributes as keywords in such a system leads a keyword based search on encrypted data. Second, although data are outsourced in an encrypted form, each authorized user is allowed to decrypt different pieces of enciphered contents, thanks to the security policy included in the ciphertext and a required match with the decrypting key of this recipient. This effectively eliminates the need to rely on the cloud storage server for preventing unauthorized data access.

There are several common drawbacks of the above works. First, they usually assume the use of a single trusted authority in the system. This not only may create a load bottleneck, but also suffers from the key escrow problem. In fact, this entity can access all the encrypted files, opening the door for potential privacy exposure. In addition, it is not practical to delegate all attribute management tasks to one entity, including certifying all users attributes or roles and generating secret keys. For instance, different organizations usually form their domains. For example, a professional association would be responsible for certifying medical specialties, while a regional health provider would certify the job ranks of its staffs. Second, there still lacks an efficient and on demand user revocation mechanism for attribute based schemes with the support for dynamic policy updates and changes, which are essential components of secure sharing use cases.

Recently, several research works used attribute based schemes to ensure fine grained access control for outsourced data [IAP09,YWRL10a,IPN+09,YWRL10b,CC09]. In these schemes, there has been an increasing interest in applying ABC to secure *Electronic Health Records* (EHRs). They consider that the use and disclosure of *Protected Health Information* should meet the requirements of Health Insurance Portability and Accountability Act (HIPAA) [MD11]. So that, they propose to provide a fine grained access policy admitted by HIPAA. For example, Ibraimi et al. [IAP09] applied ciphertext policy ABE (CP-ABE) [BSW07] to manage the sharing of PHRs, and introduced the concept of social and professional domains.
In 2010, Yu et al. proposed a key-policy ABE scheme to secure outsourced data in the cloud [YWRL10a], where a single data owner can encrypt his data and share with multiple authorized users, by distributing keys to them. The distributed keys contain attribute-based access privileges. They also propose a method for the data owner to revoke a user efficiently by delegating the updates of affected ciphertexts and user secret keys to the cloud server. Since the key update operations can be aggregated over time, their scheme achieves low overhead.

2.3.3 Homomorphic Cryptography

Homomorphic cryptosystems are cryptographic schemes whose encryption function is a homomorphism, and thus preserves group operations performed on ciphertexts. Homomorphic encryption algorithms allow a third party to perform computations on ciphertexts, ensuring privacy preservation. In this section, we first introduce the concept of the homomorphic cryptography. Then, we give a review of some applications to homomorphic encryption schemes in cloud storage environments.

2.3.3.1 General Concept

The main idea of performing simple computations on encrypted messages was first introduced by Rivest, Adleman and Dertouzous [RAD78], who referred to these computations as *privacy homomorphism*. The original motivation for privacy homomorphism was the ability to store an encrypted database by an untrusted third party, while allowing to the owner to perform simple updates and queries such that nothing about the database contents is revealed.

Several cryptographic schemes are defined over algebraic groups or rings [Coh00]. Systems, defined over groups, naturally support a single operation, usually denoted by multiplication or addition for cryptographic purposes. Meanwhile, cryptographic schemes defined over a ring naturally support two operations – addition and multiplication. Thus, if the encryption algorithm E, where both the plaintext and ciphertext spaces are groups, is a homomorphism, then such a cryptosystem is referred to as an homomorphic scheme (Definition 2.3.1).

Definition 2.3.1 *Let a cryptographic system $\mathcal{S}$ defined over $(\mathcal{M}, \mathcal{C}, \mathcal{K})$, where $\mathcal{M}$ and $\mathcal{C}$ are both groups such that for any $k \in \mathcal{K}$, the two ciphertexts c_1 and c_2 are defined as: $c_1 = E(m_1, k)$ and $c_2 = E(m_2, k)$.*
$\mathcal{S}$ is said to be homomorphic if and only if the following condition holds:

$$D(c_1 \cdot c_2) = m_1 \cdot m_2$$

where $\cdot$ are the respective group operations in $\mathcal{C}$ and $\mathcal{M}$.

Several cryptographic systems, defined over groups, were proposed to allow simple computations in encrypted data and have been known for over 30 years. For example, the encryption systems of Goldwasser and Micali [MRS88], El Gamal [Elg85], RSA [RSA78] and Paillier [Pai99], support either adding or multiplying over encrypted ciphertexts, *but not both operations in the same time.*

Definition 2.3.1 obviously extends to a system defined over a ring as follows.

Definition 2.3.2 *Let a cryptographic system $\mathcal{S}$ defined over $(\mathcal{M}, \mathcal{C}, \mathcal{K})$, where $\mathcal{M}$ and $\mathcal{C}$ are both groups such that for any $k \in \mathcal{K}$, the two ciphertexts c_1 and c_2 are defined as: $c_1 = E(m_1, k)$ and $c_2 = E(m_2, k)$.*
$\mathcal{S}$ is said to be algebraically homomorphic, or ring homomorphic if and only if the following two conditions hold:

1. $D(c_1 + c_2) = D(E(m_1, k) + E(m_2, k)) = m_1 + m_2$
2. $D(c_1 \cdot c_2) = D(E(m_1, k) \cdot E(m_2, k)) = m_1 \cdot m_2$

where $+$ and $\cdot$ are the respective ring operations in $\mathcal{C}$ and $\mathcal{M}$.

Generally, a homomorphic cryptographic scheme can be considered as a black box, when given two ciphertexts and an operation, return an encryption of the result of that operation on the two corresponding plaintexts. While addition and multiplication are common operations provided by a homomorphic scheme, we consider the symbol $\boxplus$ to denote the operation on ciphertexts which produces an encryption of the sum of n messages.

Similarly, we consider the symbol $\boxtimes$ to denote the operation on ciphertexts which provides an encryption of the product of n messages as follows:

$$D(E(m_1, k) \boxplus E(m_2, k) \boxplus \cdots \boxplus E(m_n, k)) = m_1 + m_2 + \cdots m_n$$

$$D(E(m_1, k) \boxtimes E(m_2, k) \boxtimes \cdots \boxtimes) = m_1 \cdot m_2 \cdot \cdots \cdot m_n$$

In 2005, Boneh et al. [BGN05] were the first to introduce a construction of a scheme capable of performing both operations in the same time. However, their scheme does an arbitrary number of additions and just *one* multiplication.
In 2009, Gentry proposed the first Fully Homomorphic Encryption (FHE), performing an arbitrary number of additions and multiplications [Gen09]. Later, two further fully homomorphic schemes were presented [VDHV10, SV10] following Gentry's framework. The underlying tool behind all these schemes is the use of *Euclidean lattices*, which have previously proved powerful for devising many cryptographic primitives [Grä78].

2.3.3.2 Homomorphic Cryptosystems in Cloud Storage Environments

The ability to perform simple deterministic computations on encrypted data make homomorphic schemes ideal for creating privacy preserving protocols. In general, privacy preserving protocols present the following scenario. Bob has a secret function f, and Alice has a set of inputs $\{x_1, \cdots, x_n\}$, for which she wants to learn $f(x_1, \cdots, x_n)$, without revealing her inputs. If Bob's function can be designed as an homomorphic function, then, Alice can submit encrypted inputs to Bob. This latter performs the necessary homomorphic operations, randomizes the resulting ciphertext, and sends the encrypted result back to Alice. Upon decryption, Alice learns $y = f(x_1, \cdots, x_n)$. Similarly, a group of users may wish to collectively compute the result of a public function, but without revealing their inputs. This situation presents a voting system, where each participant has a vote, and wishes to learn the final result without revealing who they voted for.

Privacy preserving protocols are usually considered secure under an *honest but curious* model or a *malicious* model. The security of these two threat models is defined with respect to an ideal implementation of a protocol where all participants securely transmit their inputs to a trusted third party, who then performs the required computation and returns the result.

In the *honest but curious* model, all entities honestly provide proper inputs, at each step of the protocol, and properly perform any calculations expected from them. The model is named *honest but curious* because each entity is honest in the sense that it does not provide false input, but curious in the sense that it attempts to gain extra information, if the protocol makes it possible. A protocol is considered as secure, under an honest but curious threat model, if the amount of information gained by each entity is identical to the information gained when using a trusted third party.
Unlike honest but curious adversaries, malicious users may attempt to deviate from the protocol or to provide invalid inputs. As such, a malicious entity is able to completely disrupt the protocol. While controlling the behavior of a malicious user is impossible, security in malicious settings is limited to preventing the malicious user from learning

extra information about another entity's input. It is important to note that a protocol being secure under the malicious model does not necessarily mean that a malicious user cannot compromise privacy. For example, in calculating a set union for two entities, the malicious user could submit the empty set, thus learning the other user's complete set from the result. Because the same technique is possible using a trusted third party, this is not considered a failure of the protocol.

Homomorphic schemes have numerous applications in the context of the cloud [NLV11, MR14]. For example, they enable private queries to a search such that the cloud client submits an encrypted query and the cloud server computes an encrypted answer without ever looking at the query in the clear. In addition, homomorphic schemes enable searching on encrypted data, where the user stores encrypted files on a cloud storage server and can later have the server retrieve only files that (when decrypted) satisfy some constraints, even though the remote server cannot decrypt the files on its own.

As discussed previously, the adoption of cloud services by consumers and businesses is limited by concerns over the loss of privacy or business value of private data. In this section, we introduce concrete and valuable applications of homomorphic encryption schemes, in medical and financial sectors. These applications can help preserving client's privacy while outsourcing various kinds of processing to the cloud.

First, in a cloud medical storage system, homomorphic implementation enables cloud to perform computation on the encrypted data on behalf of the patient. Then, the cloud provider can send the patient updates, alerts, or recommendations based on the received data. The functions to be computed in this scenario may include averages, standard deviations or other statistical functions such as logistical regression which can help predict certain dangerous health situations.

Second, in the financial industry, there is a potential application scenario in which both the data and the function to be computed on the data are private. As an example, data about corporations, their stock price or their inventory is often relevant to making investment decisions. With homomorphic functions, some functions can be evaluated privately as follows. The user uploads an encrypted version of the function to the cloud, for example a program where some of the evaluations include encrypted inputs which are specified. The streaming data are encrypted with the user public key and uploaded to the cloud. The cloud service evaluates the private function by applying the encrypted description of the program to the encrypted inputs it receives. After processing, the cloud returns the encrypted output to the client.

2.4 Formal Security Models

In order to propose efficient cryptographic protocols, it is important to understand the validation tools. In this section, we present two kinds of security validation. We first investigate computational problems in Section 2.4.1. Second, we introduce the provable security in Section 2.4.2.

2.4.1 Computational Security

The field of cryptography has motivated a number of complex computational problems. The supposition that these problems are in general computationally difficult to resolve in polynomial time provides the basis of the most contemporary cryptosystems.

- **Discrete Logarithm Problem (DLP)** – the discrete logarithm is the group equivalent of the logarithm function for real numbers. A formulation of the DLP is given as follows:

  Definition 2.4.1 *Given a generator g of a multiplicative cyclic group $\mathbb{G}$ of order p, and given the public element $y = g^x \in \mathbb{G}$, the problem of finding x is called the Discrete Logarithm Problem.*

  The difficulty of solving DLP depends on the representation and the order of the group considered. The basic solution for solving DLP is the *exhaustive search*. This approach requires $\mathcal{O}(p)$ multiplications, which is inefficient for long prime p. However, other methods, such that the Pollard's lambda and the baby-step giant-step algorithms, require only $\mathcal{O}(\sqrt{p})$. Nowadays, the most efficient method for solving the DLP is referred to as the *index calculus* (more details about DLP can be found in [Sho97, MVO96]).
  When building cryptographic algorithms relying on the difficulty of solving DLP, this often translates to the adversary being unable to distinguish between the discrete logarithms of two group elements.
  The *Computational Diffie Hellman Problem (CDH)* and the *Decisional Diffie Hellman Problem (DDH)* attempt to capture the difficulty of problems related to DLP as they generally arise in cryptography.

- **Computational Diffie Hellman problem (CDH)** – the motivation for CDH Problem is that many security systems use mathematical operations that are fast to compute, but hard to reverse [DH76].

  Definition 2.4.2 *Given a generator g of a multiplicative cyclic group $\mathbb{G}$ of order p, and given two group elements $g^a \in \mathbb{G}$ and $g^b \in \mathbb{G}$, where $a, b \in \mathbb{Z}_p{}^*$ are two secrets, the problem of calculating g^{ab} from g^a and g^b is called the Computational Diffie Hellman problem.*

  Dan Boneh believes that there is a solution to DLP that can be polynomially reduced to a solution to CDH [Bon98].

- **Decisional Diffie Hellman problem (DDH)** – The decisional version of the CDH requires an adversary to distinguish between g^{ab} and g^c for some random integer c.

  Definition 2.4.3 *Given a generator g of a multiplicative cyclic group $\mathbb{G}$ of order p, and given two group elements $g^a \in \mathbb{G}$ and $g^b \in \mathbb{G}$, where $a, b \in \mathbb{Z}_p{}^*$ are two secrets, the problem of distinguishing between tuples of the form (g^a, g^b, g^{ab}) and (g^a, g^b, g^c) for some random integer c, is called the Decisional Diffie Hellman problem.*

Unlike CDH, there exist groups where DDH is tractable, but the DLP is not. For example, an elliptic curve that supports a bilinear pairing function. That is, given a pairing $\hat{e}$ and three group elements g^a, g^b an g^c, then an adversary can easily compute $\hat{e}(g^a, g^b) = \hat{e}(g, g)^{ab}$ and $\hat{e}(g, g^c) = \hat{e}(g, g)^c$. If the two values are equal then $ab = c$ otherwise $ab \neq c$.

- **Elliptic Curve Discrete Logarithm Problem (ECDLP)** – the Elliptic Curve Discrete Logarithm Problem (ECDLP) is the projection of the DLP in an additive group.

  Definition 2.4.4 *Let $\mathbb{G}$ be a subgroup of $E(\mathbb{F}_q)$, which is generated by the point P of prime order p. Given a generator P of an additive group $\mathbb{G}$ of order p, and given the public element $Q = x.P \in \mathbb{G}$, the problem of finding x is called the Elliptic Curve Discrete Logarithm Problem.*

  The ECDLP can be solved using the baby-step giant-step algorithm or the Pollard' Rho algorithm in $\mathcal{O}(\sqrt{p})$ steps ([MVO96]). The ECDLP is widely used to derive EC keys. That is, the EC key derivation function relies on the hardness of this problem to protect the selected secret key x, given the public key Q and the generator P.

2.4.2 Provable Security

The simple fact that a cryptographic algorithm withstands cryptanalytic attacks for several years is often considered as a kind of validation procedure. Nonetheless, there is a completely different paradigm which is provided by the concept of *provable security*. A significant line of research has tried to provide proofs in the framework of complexity theory.
The first step in provable security is to define security goals. Then, it examines whether this goal is achieved by studying the probability that an adversary *wins* an experiment conducted by a challenger. This win condition can take many different forms such as finding the decryption of a given challenge ciphertext or choosing correctly which of the two games is being played. We call the experiment a *security model*. An adversary's *advantage* is a measure of how much more successful it is at winning the experiment compared to simply guessing.

In 1984, Goldwasser and Micali made a progress in provable security, introducing *semantic security* for encryption schemes [GM84]. In [MRS88], Micali et al. proved that the semantic security is equivalent to the *indistinguishability* (IND) under a chosen plaintext attack security. The IND of encryption means that if an adversary has some information about the plaintext, he should not learn about the ciphertext. This security notion requires computational impossibility to distinguish between two messages chosen by the adversary, with a probability significantly greater than a half.
For example, if an adversary selects two equal-lengths plaintexts m_0 and m_1. The challenger picks randomly one of the two plaintexts, encrypts it and gives the resulting challenge ciphertext to the adversary. Then, this adversary should not guess which message had been

encrypted. This model is said IND–Chosen Plaintext Attack (IND–CPA). In fact, the adversary is allowed to query the encryption oracle with any message and will be given the corresponding ciphertext in return. This is formalized in Definition 2.4.5:

Definition 2.4.5 *A cryptosystem is said to be indistinguishable under chosen plaintext attack (IND-CPA), if a probabilistic polynomially bounded adversary cannot win the following game with a probability greater than $\frac{1}{2} + \tau(\xi)$, where ξ is a selected security parameter and τ is a negligible function:*

1. *Using the security parameter ξ, the challenger derives the public key pk and sends it to the adversary. The challenger keeps secret the private key.*
2. *The adversary may perform a polynomially bounded number of encryptions or other operations. Then, the adversary chooses two different messages $m_0, m_1 \in \mathcal{M}$ and sends them to the challenger.*
3. *The challenger randomly selects one bit $b \in \{0, 1\}$ and sends the encryption $E(m_b, pk)$.*
4. *The adversary is free to perform any number of additional computations or encryptions. The adversary responds by either 0 or 1, and wins the game if the selected bit-value is the same chosen by the challenger.*

The IND-CPA allows an adversary to encrypt a polynomially bounded number of arbitrary messages without access to any secret information. Thus, the adversary may select a ciphertext and guess the related plaintext. This attack is called Chosen Ciphertext Attack (CCA). Definition 2.4.6 presents the IND-CCA security, based on a decryption oracle. This latter acts as a black box that takes as input a ciphertext and responds with the related plaintext.

Definition 2.4.6 *A cryptosystem is said to be indistinguishable under non-adaptive chosen plaintext attack (IND-CCA1), if a probabilistic polynomially bounded adversary cannot win the following game with a probability greater than $\frac{1}{2} + \tau(\xi)$, where ξ is a selected security parameter and τ is a negligible function:*

1. *Using the security parameter ξ, the challenger derives the public key pk and sends it to the adversary. The challenger keeps secret the private key.*
2. *The adversary is given access to the decryption oracle and may perform a polynomially bounded number of decryption or other operations.*
3. *The adversary chooses two different messages $m_0, m_1 \in \mathcal{M}$ and sends them to the challenger.*
4. *The challenger randomly selects one bit $b \in \{0, 1\}$ and sends the encryption $E(m_b, pk)$.*
5. *The access to the oracle is stopped. The adversary responds by either 0 or 1, and wins the game if the selected bit-value is the same chosen by the challenger.*

A non-adaptive chosen attack is also called lunchtime attack, because the adversary could sneak a protected system while the owner was out for lunch (Definition 2.4.6, step 2). However, he cannot rely on access later. The strongest form of indistinguishably assumes that the adversary retains access to the decryption oracle, after the challenge ciphertext is received. Otherwise, the decryption oracle does not respond, if the adversary requests the decryption of the challenge.

2.5 Conclusion

In this first chapter, we present a general introduction to public key cryptography. We investigate the usage of attribute based cryptography and homomorphic schemes in clouds. As a specific variant of attribute based cryptography, we describe the ID-based cryptosystems which rely on the use of elliptic curve groups. ECC is a promising approach that significantly reduces the size of keys and encryption algorithms. Thus, it is well-suited for the security applications designed to resource constrained devices. These cryptographic systems are referred to as interesting mechanisms, to mitigate cloud data security concerns, thanks to their attractive features. For instance, they provide much more flexible decryption schemes, and allow malleability on ciphertexts. Finally, we present some validation tools, namely the computational security and the provable security.

In the next chapter, we leverage the usage of ID Based Cryptography in cloud storage environments. We propose an original ID-based client side encryption approach, where cloud clients are assigned the IBC–PKG function. So that, they can issue their own public elements, and can keep confidential their resulting IBC secrets.
Thanks to the lightweight ID-based public key computation process and contrary to the existing classical sharing schemes, our proposal does not require for the depositor to be connected, when the recipients want to retrieve the shared data.

Chapter 3

ID-Based Cryptography for Secure Cloud Data Storage

Integrity simple means not violating one's own identity

Erich Fromm - 1900-1980

Contents

3.1 Introduction

By moving their data to the cloud, users remove the burden of building and maintaining a local storage infrastructure. As such, they only have to pay their cloud service providers for the allocated resources. Indeed, these providers offer to their clients the possibility to store, retrieve and share data with other users in a transparent way.
Unfortunately, in addition to its several advantages, cloud storage brings several security issues, namely data confidentiality preservation. Kamara and Lauter [KL10], and Chow et al. [CGJ+09] agreed that encrypting outsourced data by the client is a good alternative to mitigate such concerns of data confidentiality. So that, the client preserves the decrypting keys out of reach of the cloud provider. However, the confidentiality provisioning becomes more complicated with flexible data sharing among a group of users. It requires efficient sharing of decrypting keys between different authorized users. As such, only authorized users are able to obtain the cleartext of data stored in the cloud.

In this chapter, we present our first contribution [KBL13]. That is, we propose a new method for improving data confidentiality in cloud storage systems and enhancing dynamic sharing between users. It can be used by an authenticated client for his data storage, backup and sharing in the cloud. Our proposal relies on the use of ID-Based Cryptography (IBC), where each client acts as a Private Key Generator (PKG). That is, he generates his own public elements and derives his private key using a secret.
The originality of our proposal is twofold. First, it ensures better confidentiality. That is, every client acts as a PKG by computing an ID-based pair of keys to encrypt the data that he intends to store in the cloud. As such, the data access is managed by the data owner. Second, by using a per data ID-based key, we provide a flexible sharing approach. Indeed, the distribution of decrypting keys between the client and the authorized users, does not reveal any information about the client's secret.

This chapter is organized as follows. First, we describe, in Section 3.2, the cloud architecture considered in our work, and the related security requirements. Second, we describe in Section 3.3 our ID-based proposal, while presenting three different scenario, namely the storage, backup and sharing of data between two users and among a group of users. Then, we give a security analysis in Section 3.4 and some possible refinements in Section 3.5. Finally, we present our implementations results in section 3.6, before concluding in Section 3.7.

3.2 Architecture and Security Requirements

We present, in this section, a typical cloud storage architecture. Then, we review the security requirements, while considering realistic threats models. We first point out the case where an untrusted service provider has a curious behavior. Second, we consider the case of a malicious user that intends to get information about outsourced contents of an other data owner.

3.2.1 Architecture

Figure 3.1 illustrates a descriptive network architecture for cloud storage. It relies on the following entities, permitting a customer to store, retrieve and share data with multiple users:

- *Cloud Service Provider (CSP)*– a CSP has significant resources to govern distributed cloud storage servers and to manage its database servers. It also provides virtual infrastructure to host application services. These services can be used by the client to manage his data stored in the cloud servers.
- *Client (C)*– a client is a data owner who makes use of provider's resources to store, retrieve and share data with multiple users. A client can be either an individual or an enterprise. Each client has a unique and authentic identity, denoted by ID_C.
- *Users (U)*– the users are able to access the content stored in the cloud, depending on their access rights which are authorizations granted by the client, like the rights to read, write or re-store the modified data in the cloud. These access rights serve to specify several groups of users. Each group is characterized by an identifier ID_G and a set of access rights.

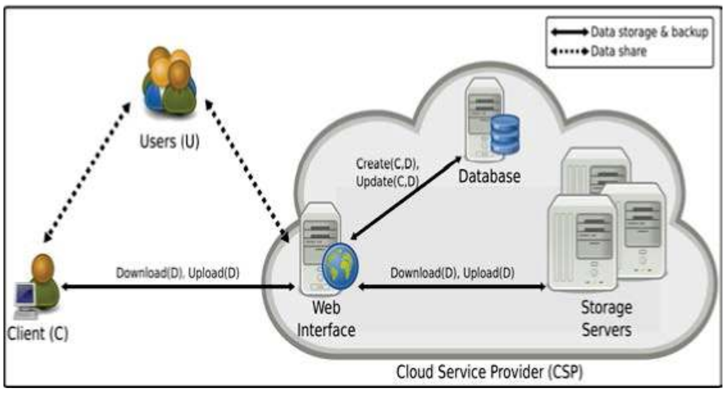

FIGURE 3.1 - Architecture of cloud data storage

In practice, the CSP provides a web interface for the client to store data into a set of cloud servers, which are running in a cooperated and distributed manner. In addition, the web interface is used by the users to retrieve, modify and re-store data from the cloud, depending on their access rights. Moreover, the CSP relies on database servers to map clients identities to their stored data identifiers and groups identifiers.

3.2.2 Security Requirements

When outsourcing data to a third party, providing confidentiality and privacy becomes more challenging and conflicting. Privacy is a critical concern with regards to cloud storage due to the fact that clients' data reside among distributed public servers. Therefore, there are potential risks where the confidential information (e.g., financial data, health record) or personal information (e.g., personal profile) are disclosed. Meanwhile, confidentiality implies that client's data have to be kept secret from both cloud provider and other users. Confidentiality remains as one of the greatest cloud security concerns. This is largely due to the fact that users outsource their data on cloud servers, which are controlled and managed by potentially untrusted CSPs. That is why, it is compulsory to provide secrecy by encrypting data before their storage in cloud servers while keeping the decryption keys out of reach of CSP and any malicious user.

For designing the most suitable security solutions for cloud storage, we are considering a *honest but curious* cloud provider, as a threat model (Section 2.3.3.2). In such cases, the remote server honestly performs the operations defined by our proposed scheme, but it may actively attempt to gain the knowledge of the outsourced data.

In addition, we addressed the case of a malicious user, with regards to the second threat model, presented in Section 2.3.3.2. For instance, an attacker can be either a revoked user with valid data decryption keys, an unauthorized group member or a group member with limited access rights. Therefore, secure data sharing should support flexible security policies including forward and backward secrecy.

- *Forward secrecy* – this property requires that the confidentiality of previously encrypted data has to be ensured even after the long-term secrets are exposed. For example, a user cannot access stored data before he joins a group.

- *Backward secrecy* – this property means that a compromise of the secret key does not affect the secrecy of future encrypted data. A such, a revoked group member is unable to access data that were outsourced after he leaves the group.

Beyond these security requirements, our proposal aims to achieve several security and system performances. That is, the overhead of implemented security mechanisms should be acceptable.

3.3 ID-Based Cryptography for Securing Cloud Applications

In this section, we introduce our ID-based construction for securing cloud applications, before enumerating the considered prerequisites (Section 3.3.1). Then, we describe in depth our proposed solutions for data storage, backup and sharing.

Our main idea consists in using ID-Based Cryptography to provide a per data pair of keys. In fact, our proposition inherits attractive properties from IBC such as being certificate-free and having small key sizes. This potentially offers a more lightweight key management approach.
In [KBL13], we propose to use each client as a Private Key Generator (PKG) which generates his own *ID-Based Cryptography Public Elements* (IBC–PE). These IBC–PE are used to compute ID-based keys. These keys serve to encrypt the data before their storage and

sharing in the cloud. Note that for every different data, the client computes the corresponding private and public keys relying on his IBC–PE and a local secret s_C.

The choice for IBC is motivated by several reasons. First, we benefit from an easier key management mechanism thanks to the certificate-free feature of IBC. That is, the computation of public keys from the unique data identifiers does not require the deployment of a Public Key Infrastructure (PKI) and the distribution of certificates. Second, IBC permits deriving public keys with no need for previous computation of corresponding private keys. That is, contrary to traditional public key derivation schemes, IBC does not require to generate the private key before the public key. Indeed, users have only to generate ID-based public keys to encrypt data before storage. As such, any user can directly encipher data for a client at no extra cost of communication. The derivation of the corresponding private keys is only needed at the time of data recovery. Third, IBC permits to derive a per data key from a unique data identifier thanks to the lightweight key computation. The derivation of a per data key is well suited for a sharing process. That is, the client uses a different ID-based pair of keys for each new data storage. Therefore, he has merely to reveal the ID-based private key needed for shared data decryption. As such, we avoid the use of the same key for enciphering all the outsourced data. That is, when the private key used for the decryption is captured by an attacker, he cannot get any information about the other per data keys. In fact, the client should not use a unique long term key for all his data encryption. He has just to reveal the ID-based private key needed for the data decryption.

3.3.1 Prerequisites

This section gives the prerequisites considered for designing our ID-based solution. First, we assume that there is an established secure channel between the client and the CSP (cf. Fig 3.1). This secure channel supports mutual authentication and data confidentiality and integrity. It can be implemented through the Transport Layer protocol (TLS) [DR08], where the client can authenticate with a certificate or password. TLS permits data to be transmitted securely.

Second, each client generates his own IBC–PE that he intends to use to secure his data storage. By acting as a PKG, the cloud client bears the responsibility of generating the ID-based public elements. That is, the client computes the probabilistic polynomial time algorithm which outputs the groups $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ and the pairing function $\hat{e}$ from $\mathbb{G}_1 \times \mathbb{G}_2$ in $\mathbb{G}_T$.

Note that the client keeps secret s_C which is needed for IBC–PE generation and private keys derivation. We must also note that, in practice, the client should first select the ID-based encryption scheme which will be used for ciphering messages. Our proposal does not depend on the defined scheme. However, that choice depends on the way the private keys are generated.

After successfully authenticating with the CSP, the client starts the storage process as detailed in Section 3.3.2. Indeed, the client enciphers his data using a per data ID-based public key Pub_D that is derived from the concatenation of the client's identity ID_C and the data identifier ID_D, as follows:

$$Pub_D = Hash_{pub}(ID_C || ID_D) \tag{3.1}$$

ID_D is locally generated by the client and is derived from the meta-data (MD) using a one way hash function $H()$ as $ID_D = H(MD)$. We assume that MD supports the data model as specified by the Cloud Data Management Interface standard (CDMI) [Sni10]. The CDMI defines many different types of metadata, including HTTP meta-data, data system meta-data, user meta-data, and storage system meta-data. In our proposal, MD are referred to as user meta-data, which are arbitrarily-defined information that are specified by the CDMI client and attached to objects.
Our choice to hide the content of MD is motivated by the need to ensure the data privacy of a client. After ciphering his data, the client sends his signed encrypted data in order to be stored. When the client wants to get back his information, he starts with the provider the backup process presented in Section 3.3.3. When he wants to share the stored data with other clients, he executes the sharing procedure described in Section 3.3.4.

3.3.2 Secure Data Storage

When a client wants to store data in the cloud, he has to generate the data identifier ID_D. This identifier, associated to a client's identity, must be unique in the CSP database. Thus, the client starts the storage process by sending a *ClientRequestVerif* message to verify the uniqueness of the generated ID_D to his CSP. Recall that, in practice, the client should first select the ID-based encryption scheme which will be used for enciphering messages. Our proposal does not depend on the defined scheme. However, that choice depends on the way of generating the private keys. In addition, the client defines the corresponding IBC–PE$_C$.

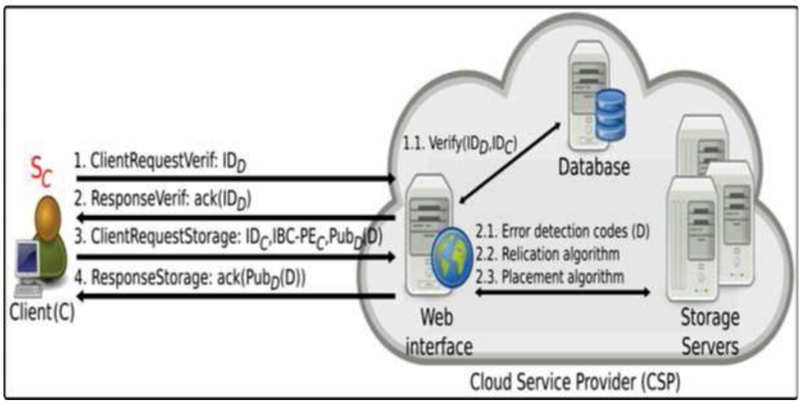
S_C
1. ClientRequestVerif: ID_D
2. ResponseVerif: ack(ID_D)
3. ClientRequestStorage: ID_C,IBC-PE$_C$,Pub_D(D)
4. ResponseStorage: ack(Pub_D(D))
Client(C)
1.1. Verify(ID_D,ID_C)
Database
2.1. Error detection codes (D)
2.2. Relication algorithm
2.3. Placement algorithm
Web interface
Storage Servers
Cloud Service Provider (CSP)

FIGURE 3.2 - Secure Data Storage

The storage process consists in exchanging the four following messages (cf. Fig 3.2):

- *ClientRequestVerif*: this first message contains the generated data identifier ID_D. This message is a request for the verification of the uniqueness of the ID_D. More specifically, the client sends the ID_D derived from MD in order to verify the uniqueness of the data identifier in the cloud database servers. The CSP replies with a *ResponseVerif* message to validate or unvalidate the claimed identifier. Note that the data storage process has to be stopped when the uniqueness verification fails.

- *ResponseVerif*: this acknowledgement message is generated by the CSP to validate the requested ID_D. When receiving this message, the client concatenates ID_C and ID_D for deriving the public key Pub_D used to encipher his data.

- *ClientRequestStorage*: it contains the public elements generated by the client IBC–PE$_C$, the encrypted data Pub_D(D) and optionally a selected group identifier ID_G with the access rights granted by the client to the associated users of the group. ID_G is only required if data are stored for a sharing purpose (Section 3.3.4). Recall that s_C is kept secret by the client.

- *ResponseStorage*: this acknowledgement message, sent by the CSP, is used to confirm to the client the success of his data storage.

3.3.3 Secure Data Backup

The data backup process starts when the client requests for retrieving the data previously stored in the cloud. The data backup process, presented in Figure 3.3, includes two messages:

- *ClientRequestBackup*: this first message is sent by the data owner. It contains the data identifier ID_D of the requested data that the client wants to retrieve.

- *ResponseBackup*: the response of the CSP includes the encrypted outsourced data Pub_D(D). Upon receiving the *ResponseBackup* message, the client derives the data private key $Priv_D$ from the local stored secret s_C and the IBC–PE$_C$, in order to decipher the data.

3.3.4 Secure Data Sharing

We consider the data sharing process, where the client outsources his data to the cloud and authorizes a group of users to access the data. Next, we refer to these user(s) as the recipient(s) and to the data owner as the depositor. Afterwards, we distinguish two different scenarios. First, the data sharing one to one, presented in Section 3.3.4.1, when a depositor stores for one recipient. Second, the data sharing one to many, described in Section 3.3.4.2, when a depositor shares data among a group of recipients. We must note that our proposal does not require from the recipients to be connected during the sharing process. Indeed, recipients' access rights are granted by the data owner and managed by the CSP. That is, the CSP is in charge of verifying each recipient access permissions before sending him the outsourced data.

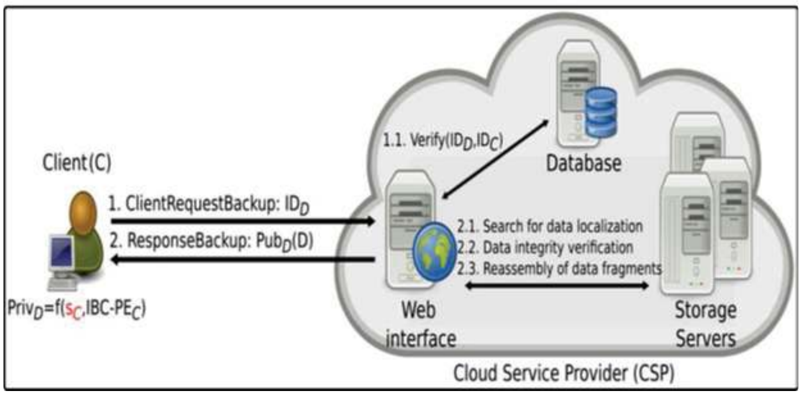
Client(C)
1. ClientRequestBackup: ID_D
2. ResponseBackup: $Pub_D(D)$
$Priv_D=f(s_C,IBC\text{-}PE_C)$
1.1. Verify(ID_D,ID_C)
Database
2.1. Search for data localization
2.2. Data integrity verification
2.3. Reassembly of data fragments
Web interface
Storage Servers
Cloud Service Provider (CSP)

FIGURE 3.3 - Secure Data Backup

3.3.4.1 Scenario E1: Secure Data Sharing One To One

The Data Sharing One To One scenario is defined when a depositor wants to share data with one recipient. That is, the depositor ID_U can store encrypted data for this recipient client ID_C by using a per data ID-based public key and the public elements IBC–PE$_C$ of the recipient. The depositor derives the identifier of the data that he intends to share with the recipient and generates the per data public key as follows:

$$Pub_D = Hash_{pub}(ID_U||ID_C||ID_D)$$

This sharing process includes the following messages (cf. Fig 3.4):

- *UserRequestStorage*: this message is a request sent by the depositor that includes the new generated data identifier ID_D and the data encrypted with Pub_D. After verifying uniqueness of ID_D, the CSP stores the data and sends back the *ResponseStorage* message.

- *ResponseStorage*: it is an acknowledgement message sent by the CSP to the requesting depositor. Then, the CSP sends a notification to the recipient to notify the availability of new data enciphered with his public elements. Note that, the CSP also includes, in this notification, the depositor identity ID_U and the data identifier ID_D. When the recipient receives this notification, he starts a backup process, as detailed in Section 3.3.3.

3.3.4.2 Scenario E2: Secure Data Sharing One To Many

When a depositor wants to share data among a group of recipients, he has first to generate the data identifier ID_D and a selected group identifier ID_G with the access rights

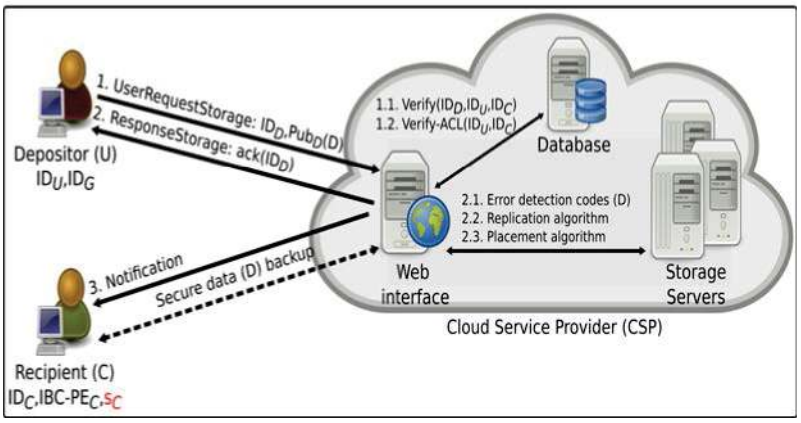
1. UserRequestStorage: ID_D,$Pub_D(D)$
2. ResponseStorage: ack(ID_D)
Depositor (U)
ID_U,ID_G
1.1. Verify(ID_D,ID_U,ID_C)
1.2. Verify-ACL(ID_U,ID_C)
Database
2.1. Error detection codes (D)
2.2. Replication algorithm
2.3. Placement algorithm
3. Notification
Secure data (D) backup
Web interface
Storage Servers
Cloud Service Provider (CSP)
Recipient (C)
ID_C,IBC-PE_C,s_C

FIGURE 3.4 - Secure Data Sharing One To One

granted to the associated users of the group. Then, he computes a per data public key as follows:

$$Pub_D = Hash_{pub}(ID_G||ID_D)$$

In practice, each recipient is assumed to know the corresponding private key for decrypting outsourced data. This private key distribution problem can be solved in two ways. Either the depositor sends the deciphering key to the recipient as soon as he stores data or a proxy is in charge of distributing the private keys. Once the depositor stored the data with the authorized access rights of the group, each member of the group can start the data sharing process based on the two following messages (cf. Fig 3.5):

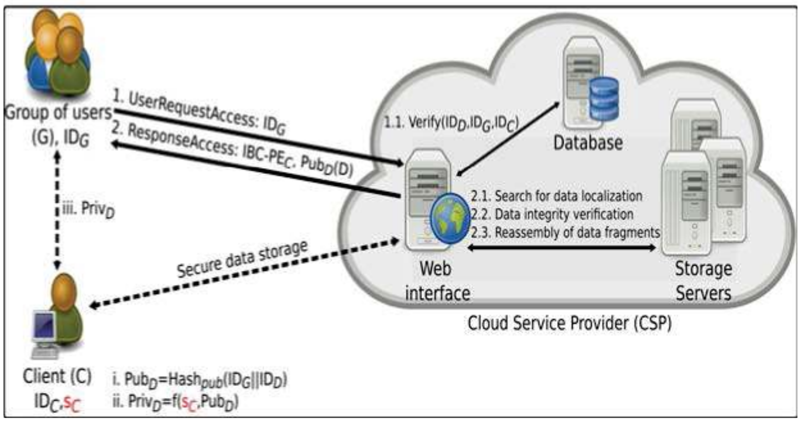
Group of users (G), ID_G
1. UserRequestAccess: ID_G
2. ResponseAccess: IBC-PE_C. $Pub_D(D)$
1.1. Verify(ID_D,ID_G,ID_C)
Database
iii. $Priv_D$
2.1. Search for data localization
2.2. Data integrity verification
2.3. Reassembly of data fragments
Secure data storage
Web interface
Storage Servers
Cloud Service Provider (CSP)
Client (C)
ID_C,s_C
i. Pub_D=$Hash_{pub}(ID_G||ID_D)$
ii. $Priv_D$=f(s_C,Pub_D)

FIGURE 3.5 - Secure Data Sharing One To Many

- *UserRequestAccess*: this message contains the requested data identifier ID_D. Once receiving this message, the CSP searches for the read/write permissions of the recipient, and then, he generates a *ResponseAccess* message.

- *ResponseAccess*: the CSP includes, in its response, the public elements of the depositor IBC–PE$_C$ and the encrypted data Pub_D(D).

3.4 Security Analysis

In this section, we give an informal security analysis of our proposal, following the design goals, presented in Section 3.2. Recall that we have considered two threat models namely an *honest but curious* cloud provider and a malicious user that intends to get extra-information from outsourced data. In addition, we expose its possible refinements to mitigate other threats.

- *Privacy*– privacy is a critical concern with regards to cloud storage due to the fact that clients' data reside among distributed public servers. Therefore, there are potential risks where the confidential information (e.g., financial data, health record) or personal information (e.g., personal profile) are disclosed.

  Based on cryptographic solution to keep data content secret, sensitive information are generally included in meta-data (e.g., file name, client identity, keywords) [CM05]. Therefore, in our proposal, we present an ID-based cryptographic solution based on hashed meta-data. As such, these meta-data form the data identifier, which is used to derive the per data public key. Thus, the client has privacy guarantees on his stored data. First, meta-data content can never be disclosed to the CSP, as he only has access to hashed information $ID_D = H(MD)$. Second, the CSP cannot reveal the content of stored data. In fact, although, he has the data identifier and the public elements of the client IBC–PE$_C$, he does not have the secret s_C needed to derive the private key and decipher data.

  Furthermore, searching for stored data, for a backup process, may also endanger the privacy. That is, general retrieval methods are based on keywords search. However, the enforcement of these propositions partially violates privacy protection, since the CSP can guess the content of the stored data based on keywords. To alleviate this privacy problem, we propose to use the *hashed* meta-data as a data identifier. This identifier is unique and it will serve to search data from cloud servers.

  Moreover, privacy is also threatened by the *accountability* requirement. That is, accountability implies the capability of identifying a party, with undeniable evidence. This remains an essential requirement for the Cloud Pricing Model (CPM).
  In order to identify a cloud user, general accountability approaches include user profiling, information logging, replay, tracing [YPX05], etc. These operations may not be completed without revealing some private information. Unfortunately, this security conflict between accountability and user privacy has not been solved yet by our approach. That is, our proposal is based on identities. Nevertheless, we may rely on third trusted party, to manage a federation identity mechanism.

Finally, we note that privacy is tightly related to confidentiality, due to the notion that they both prevent information leakage. Therefore, if data confidentiality is ever violated, privacy will also be violated.

- *Data confidentiality*– when dealing with cloud storage environments, confidentiality implies that client's data have to be kept secret from both cloud provider and other users. As we assumed that data are stored on cloud servers, which are controlled and managed by *honest* but *curious providers*, confidentiality remains as one of the greatest concerns.

  In this proposal, we perform an ID-based cryptographic solution to ensure data confidentiality for secure data storage, backup and sharing.

  First, we propose to outsource encrypted data to cloud servers. In our approach, the client is in charge of enciphering his data before their storage in the cloud. He acts as a PKG entity and he is responsible for generating and managing his secrets on his own. Thus, he is the only entity knowing the IBC secret s_C. This secret, which is kept locally by the client, is needed to derive any deciphering key. Therefore, it is impossible for the CSP or a malicious user to retrieve the deciphering key to decrypt data.

  Second, we propose to use a per data key for enciphering data. This proposal is well suited for the sharing process, as, the client uses a different ID-based pair of keys for each new data storage. He has only to reveal the ID-based private key needed for shared data decryption. As such, we avoid using the same key for enciphering all the outsourced data. In fact, when the private key used for the decryption is captured by an attacker, he cannot get any information about the other per data keys.

- *Access control to data*– the proposed secure sharing scheme, discussed in Section 3.3.4, authorizes recipients to have access to data, based on their respective access rights. In [KBL13], we distinguish two sharing scenarios, on the basis of the number of recipient users. As an example, the one to one scenario ensures forward secrecy, since data encryption is performed using the public key of the recipient. Additionally, we guarantee backward secrecy, as each storage and sharing scenario requires a per data decrypting key. This deciphering key depends on both outsourced data identifier and the recipient identity.

  The issue of unauthorized access to data is twofold. First, the issued access rights to the recipients are granted by the depositor and managed by the CSP. In addition, when a recipient wants to access to data, he has first to authenticate with the CSP. That is to say that the access to data has already been strictly controlled by an authentication phase, before the verification of the authorizations granted by the depositor. Therefore, the enforcement of the access control policy is safeguarded.

  Second, even though the CSP or a malicious recipient can gain access to the data, the enforcement of data confidentiality is still guaranteed. In fact, they can only have access to encrypted data or to hashed meta-data. They cannot have the needed private key to decipher data.

Finally, our first contribution is not restricted to any specific ID-based encryption scheme. So, instantiation of our proposal is given flexibility to implement appropriate ID-based encryption schemes. No change to the adopted ID-based schemes is made, hence, the security properties of the cryptographic primitives are well respected. However, like for any distribution scenario (e.g., sharing one to many), the issue of revoking some users' access privileges arises but can be classically solved at the cost of key re-distribution and data re-encryption, in order to ensure forward and backward secrecy.

3.5 Limitations and Possible Improvements

In this section, we lead a general discussion, while introducing possible improvements of our ID-based proposal.

3.5.1 Computation Complexity

Our proposition assumes that the client acts as a PKG. That is, he bears the responsibility of generating his own public elements IBC-PE$_C$ and deriving the corresponding private keys of all outsourced data files.
In an effort to alleviate the computation complexity at the client side, we propose that the PKG role is distributed partly to the cloud server and to the client. As such, the only one operation that remains to the client is the derivation of the data private key instead of the computation of the ID public elements, as proposed in our solution (Section 3.3).
In fact, the CSP relies on a *SystemInit* procedure, in order to generate the IBC-PE$_{CSP}$. This algorithm takes as input a security parameter ξ and outputs the algebraic groups and pairing functions, which will be used for ciphering data contents.

In practice, the cloud server first selects the ID-based encryption scheme. Then, it defines the corresponding public elements. We must note that the choice of the IBE scheme depends on the way data private keys are generated. Our ID-based contribution does not depends on the defined IBE schemes. However, we next present the storage scenario, using the Boneh and Franklin scheme [BF01]. That is, we suppose that the cloud server chooses to generate his private key by multiplying his public key with its secret s_{CSP} [BF01]. In addition, it creates the following IBC-PE$_{CSP}$ $=\{\mathbb{G}_1,\ \mathbb{G}_T,\ q,\ \hat{e},\ g,\ P,\ Hash_{pub}(),\ H_1()\}$, where $H_1 : \mathbb{G}_T \rightarrow \{0,1\}^n$. The CSP keeps secret the random s_{CSP}, which will serve to generate a part of the data private key as: $PartPriv_D = s_{CSP}.Pub_D$.

When a client (C) wants to store a new data file (D) in cloud servers, he has to authenticate with the CSP, based on EAP-TLS protocol [DR08]. Recall that TLS supports mutual authentication between clients and servers and it enables the secure transmission of all information elements needed for the derivation of a pre-shared key k_{CS}.
Then, the client starts the storage process, which consists in exchanging the four following messages (cf. Fig. 3.6):

- **Message 1:** is the *ClientRequestVerif* message. It is sent by the data owner to the cloud server. It contains the data identifier ID_D. Message 1 is sent enciphered using the pre-shared key k_{CS}. When the CSP receives this first message, it first verifies

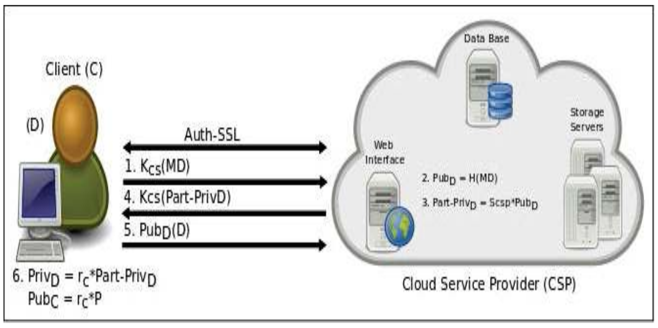

FIGURE 3.6 - Cloud data storage scenario

the uniqueness of the claimed identifier. In fact, the CSP does not accept a data identifier unless it does not exist in its database. Then, the cloud server computes the data public key as: $Pub_D = Hash_{pub}(ID_D)$, using the public elements IBC-PE. We must note that the ID_D is locally generated by the client and is derived from the meta-data (MD) using a one way hash function $H()$ as $ID_D = H(MD)$. In addition, the CSP computes the first part of the data private key as $PartPriv_D = s_{CSP}.Pub_D$.

- **Message 2:** acknowledges the claimed data identifier and it includes $PartPriv_D$ and Pub_D. It is enciphered using the pre-shared key and is sent by the cloud provider to the data owner.
  Upon receiving this message, the data owner first selects a random s_C and computes the data private key, using the IBC-PE and $PartPriv_D = c_{CSP}.Pub_D$, as: $Priv_D = s_C.PartPriv_D = r_C.s_{CSP}.Pub_D$. Then, the client enciphers the data file D, using the public key Pub_D. With the Boneh and Franklin encryption scheme, the enciphered data file is formed by the pair (U, V), such that:

$$C = (U, V) = (k.P, D \oplus H_1(\hat{e}(Pub_D, P_{pub})^k)) \tag{3.2}$$

  where $k \in \mathbb{Z}_n{}^*$, and $P_{pub} = r_C.P$.

- **Message 3:** is a *ClientRequestStorage* message. It contains the enciphered data file $Pub_D(D)$. This message is sent by the client to the cloud server.

- **Message 4:** is a *ResponseStorage* message which is sent by the CSP to confirm the success of the data file storage.

When the data owner wants to retrieve his outsourced data, he has to start a backup process, as presented in Section 3.3.3.
While sharing the PKG function between the data owner and the cloud provider, we significantly reduce the computation complexity at the client side. In fact, the CSP has the responsibility of generating the largest part of IBC-PE. By definition, the cloud server is assumed to have a lot of processing capabilities and storage capacities. As such, managing the IBC-PE and generating private keys will have no effect on its natural function. In addition, the CSP has to refresh these IBC-PE periodically in order to avoid attacks against his master secret key s_{CSP}.

3.5.2 Deduplication Concern

Our ID-based proposal relies on the use of a per data keying approach, while the depositor takes charge of generating the related enciphering key to encrypt data contents, before storing on remote servers. Thus, this approach leads to encrypt the same content several times, and then, to decrease the storage capacities of the cloud provider.
For saving resources consumption in both network bandwidth and storage capacities, many cloud services, namely Dropbox, Wuala and Memopal, apply client side deduplication [HPSP10, Dut08]. This concept avoids the storage of redundant data in cloud servers and reduces network bandwidth consumption associated to transmitting the same contents several times.
Despite these significant advantages in saving resources, client data deduplication brings many security issues, considerably due to the multi-owner data possession challenges [HPSP10]. For instance, several attacks target either the bandwidth consumption or the confidentiality and the privacy of legitimate cloud users. For example, a user may check whether another user has already uploaded a file, by trying to outsource the same file to the cloud.
Recently, to mitigate these concerns, many schemes have been proposed under different security models [NWZ12, DPS12, XCZ13, HHPSP11, SGLM08]. These schemes are called *Proof of Ownership* systems (PoW). They allow the cloud server to check a user data ownership, based on a static and short value (e.g. hash value). These security protocols are designed to guarantee several requirements, such as lightweight of verification and computation efficiency. Even though existing PoW schemes have addressed various security properties, we still need a careful consideration of potential attacks such as data leakage and poison attacks, that target privacy preservation and data confidentiality disclosure.

In order to mitigate such concern, we propose in [KL14] a content hash keying approach, ensuring client side deduplication in cloud storage applications. That is, [KL14] improves data security in cloud storage systems while ensuring efficient data deduplication. Our idea consists in using the *Merkle-based Tree* over encrypted data, in order to derive a unique identifier of outsourced encrypted data [Mer88]. On one hand, this identifier serves to check the availability of the same data in remote cloud servers. On the other hand, it is used to ensure efficient access control in sharing scenarios.
In fact, when a data owner wants to store a new data file D in the cloud, he derives the enciphering key k_D based on a one-way hash function $H()$. Note that data are stored enciphered in cloud servers, based on a symmetric algorithm, using the derived key k_D. Hence, the data owner has first to encipher the data file that he intends to outsource. Then, he generates the data identifier MT_D. That is, it is the *Merkle Tree* over encrypted data. This identifier, associated to the file, must be unique in the CSP database. Thus, the client starts the storage process by sending a *ClientRequestVerif* message to verify the uniqueness of the generated MT_D to his CSP.

- **New Data File Storage** – The storage process consists in exchanging the four following messages (cf. Fig 3.7):
  - *ClientRequestVerif*: this first message contains the generated data identifier MT_D, associated to a nonce n. The nonce is used to prevent from replay

attack or potential capture of the data identifier. This message is a request for the verification of the uniqueness of the MT_D. The CSP replies with a *ResponseVerif* message to validate or unvalidate the claimed identifier. Note that if the sent identifier exists, the client has to perform a subsequent upload extra-proof procedure with the provider. Once the verification holds, the cloud server tags the requesting entity as a data owner and asks the client to send only the access rights of authorized users.

- *ResponseVerif*: this acknowledgement message is generated by the CSP to inform the client about the existence of the requested MT_D in its database.
- *ClientRequestStorage*: this message is sent by the client. If the file does not exist in the cloud servers, the client sends the encrypted file, and the data decrypting key k_D enciphered with the public keys of authorized users. Then, the enciphered k_D is included in the meta data of the file and it serves as an extra access right provision.
- *ResponseStorage*: this acknowledgement message, sent by the CSP, is used to confirm to the data owner the success of his data storage.

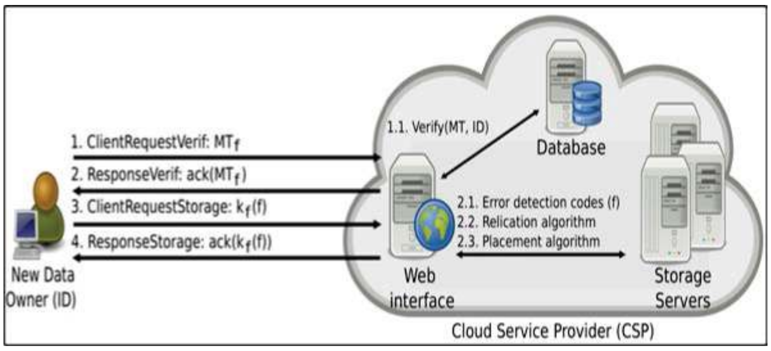
1. ClientRequestVerif: MT_f
2. ResponseVerif: ack(MT_f)
3. ClientRequestStorage: k_f(f)
4. ResponseStorage: ack(k_f(f))
New Data Owner (ID)
1.1. Verify(MT, ID)
Database
2.1. Error detection codes (f)
2.2. Relication algorithm
2.3. Placement algorithm
Web interface
Storage Servers
Cloud Service Provider (CSP)

FIGURE 3.7 - New Data File Storage

- **Subsequent Data File Storage** – When a subsequent data owner wants to store a previously outsourced data file, he sends the data file identifier MT_D to the cloud provider. Since the claimed identifier exists in cloud database, the cloud has to verify that the requesting entity is a legitimate client. That is, the subsequent data storage procedure includes these four messages (cf. Fig 3.8):
  - *ClientRequestVerif*: a subsequent data owner includes in this first message the generated data identifier MT_D, associated to a nonce n, in order to check its uniqueness in cloud database.
  - *OwnershipRequest*: this message is sent by the CSP, to verify the client's data ownership. It contains random leaves' indices of the associated Merkle tree of the requested file. Upon receiving this message, the client has to compute the associated sibling path of each leaf, based on the stored Merkle tree, in order to prove his ownership of the requested file.
  - *ClientResponseOwnership*: in his response, the client must include a valid sibling path of each selected leaf. The CSP verifies the correctness of the paths provided

by the client. We must note that this data subsequent storage process stops if the verification fails.

– *ResponseStorage*: if the data ownership verification holds, the CSP sends an acknowledgement, to confirm the success of storage.

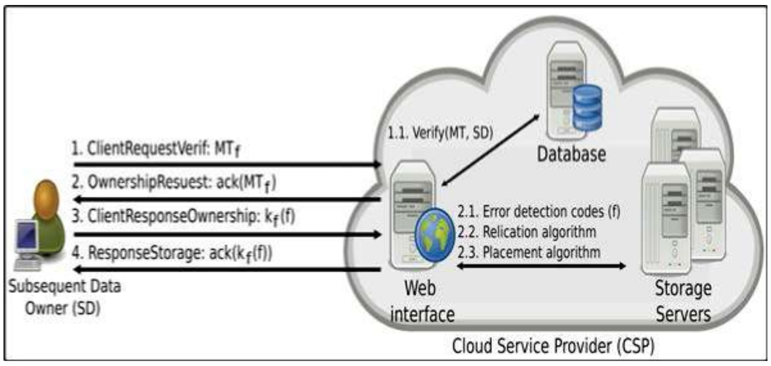
1. ClientRequestVerif: MT_f
2. OwnershipResuest: ack(MT_f)
3. ClientResponseOwnership: k_f(f)
4. ResponseStorage: ack(k_f(f))
Subsequent Data Owner (SD)
1.1. Verify(MT, SD)
Database
2.1. Error detection codes (f)
2.2. Relication algorithm
2.3. Placement algorithm
Web interface
Storage Servers
Cloud Service Provider (CSP)

FIGURE 3.8 - Subsequent Data File Storage

3.6 Implementation Results

As stated above, our proposition does not require any specific ID-based encryption scheme. For instance, the effort to evaluate the performance of our ID-based contribution leads us to study the time performance of some well-known ID-based encryption schemes. Our tests are conducted in order to understand the execution cost of our proposal on real hardware. First, we implemented Boneh–Franklin [BF01], Boneh–Boyen [BB04] and Chen et al. [CCMLS06] encryption algorithms using the Pairing-Based Cryptography (PBC) library [Ben07]. The choice of IBE schemes joins our motivation to evaluate different private key derivation functions. Then, we evaluated their encryption and decryption durations, of the same 10 kB block of random data for each IBE algorithm, using a symmetric pairing function (type A).
For our tests, we used 1000 samples in order to get our average durations. The standard deviations are of order of 10^{-2}, compared to key generation times. We computed the standard deviations with respect to the generation time of our implemented key derivation functions, which are launched before any encryption and decryption schemes. In addition, we conducted our experiments on an Intel core 2 duo, started on *single mode*, where each core relies on 800 MHz clock frequency (CPU). Single mode avoids taken into account the CPU time consumed by other interrupting programs during time evaluation.
In fact, to evaluate the different operation times, we used an assembly code that evaluates the CPU usage time of a function. processor time is different from actual wall clock time because it does not include any time spent waiting for input and output.
Let us suppose that we are going to evaluate the running time of a certain procedure p. The assembly code gets from the processor internal counter the number of ticks before launching the procedure p and the number of ticks [1] at the end of p execution. The

[1]a tick is a system clock pulse

processor internal counter is incremented at every clock pulse since the system is booted. Then, we calculate the number of ticks relative to p by computing the difference of the two previously fixed values. Finally, we divide the function's ticks number by the processor frequency.

Table 3.1 and Figure 3.9 summarize the results of the implementation of Boneh and Franklin, Boneh and Boyen, and Chen et al. encryption and decryption algorithms.

TABLE 3.1 - IBE encryption and decryption duration (in ms).

Security level (in bits)	80	112	128
Encryption time			
Boneh–Franklin	11.2	45.1	106.6
Boneh–Boyen	15.6	53.4	110.8
Chen et al.	5.9	19.8	41.4
Decryption time			
Boneh–Franklin	5.3	25.5	65.5
Boneh–Boyen	10.5	50.8	130.9
Chen et al.	5.3	25.4	65.4

During the encryption phase, we noticed that Boneh and Boyen algorithm was the slowest encryption algorithm even if Boneh and Franklin encryption contained one pairing function. That is, Boneh and Boyen contains three scalar and point multiplications while Boneh and Franklin contains only one scalar and point multiplication. Thus, the number of operations in the group of elliptic curve points is as important as the number of pairings during the selection of the most efficient encryption algorithm, given constrained resources of end users' devices.
We note also from Table 3.1 and Figure 3.9 that Chen et al. algorithm is more faster than the other encryption schemes. Chen et al. encryption scheme relies on Sakai-Kasahara key derivation function which reduces the number of computed pairings and scalar and point multiplications, while defining encryption algorithms.
Table 3.1 shows that the selection of algorithm and security level have great impact over time performances of IBE encryption and decryption operations. This is due to IBE algorithms integrating a varying number and type of pairing functions and operations in the group of elliptic curve (scalar and point multiplications). For example, during the decryption phase, Boneh–Boyen algorithm took twice the time (10 ms) than Boneh–Franklin and Chen et al. algorithms put to end the decryption (5 ms). In fact, Boneh–Boyen relies on 2 pairing computations when the two other algorithms rely only on one pairing calculus.

Better IBE performances can be expected in the future with definition of new pairing functions like Beuchat et al. running in less than 1 ms [BGDM+10]. However, IBE schemes will still remain slower than the classical AES encryption algorithm mostly used today by cloud storage providers [Ama]. As a matter of fact, IBC should be considered as an interesting compromise between computation time and memory storage.

In addition, Figure 3.9 shows that the consumed time for encryption or decryption

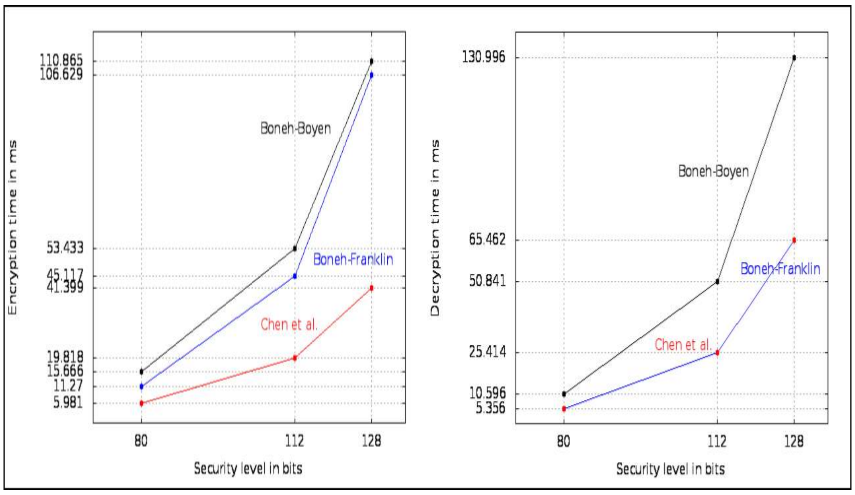

FIGURE 3.9 - IBE encryption and decryption duration (in ms)

increases, independently from the choice of the encryption algorithm, when we increase the level of security. The latter is recurrent concept in cryptography. It permits to evaluate the hardness of breaking an encryption or a signature algorithm. That is, the longer the level of security is, the harder the cryptanalysis of the algorithm becomes. For more details about the security level definition for ID-based encryption algorithms, please refer to Paterson et al.'s work [GPS08].

The definition of several IBC–PE depending on different security level represents an interesting extension to our proposal. Indeed, the client relies on different IBC–PE with respect to the sensitiveness of the data that he intends to share on the cloud. That is, for important data, the client chooses IBC–PE with a higher security level (e.g. 128 bits).

The selected security level must be tightly adapted to the required security level for mitigating performance lowering. As such, the client can apply several IBC-PE and select one of them according to the sensitivity of his data. That is, for critical data, the client can choose IBC-PE with a higher security level (e.g. 128 bits). Consequently, the client's data encryption and decryption will last longer because the elliptic curve keys used for encryption and decryption will be around 256 bits long.

3.7 Conclusion

The growing need for secure cloud storage services and the attractive properties of ID-based cryptography lead us to combine them, thus, defining an innovative solution to the data outsourcing security issue.

Our first contribution is based on a specific usage of IBC [KBL13]. First, the cloud storage clients are assigned the IBC–PKG function. So, they can issue their own public elements, and can keep their resulting IBC secret confidential. Second, a per data key which is derived from a data identifier is used to encipher data.

Thanks to IBC properties, this contribution is shown to support data privacy and confidentiality, as it employs an original ID-based client side encryption approach. It is also shown to be resistant to unauthorized access to data and to any data disclosure during the sharing process, while considering two realistic threat models, namely an honest but curious server and a malicious user adversary.
In addition, due to the lightweight ID-based public key computation process and contrary to the existing classical sharing schemes, our proposal does not require for the depositor to be connected, when the recipients want to retrieve the shared data.
Besides, some experiments are provided, in order to evaluate the computation cost of ID-based encryption schemes on end users' real hardware. For instance, IBE schemes still remain slower than the classical AES encryption algorithm. Otherwise, we believe that IBC should be considered as an interesting compromise between computation cost and memory storage.

Finally, a general discussion is presented, while introducing two possible refinements to our ID-based contribution. The first approach consists on the distribution of the PKG role partly to the cloud server and to the client, in order to alleviate the computation complexity at the client side. The second one introduces a client-side deduplication scheme for cloud applications [KL14]. That is, deduplication is a challenging concern for cloud providers, allowing them to better use their storage capacities. The convergent encryption is a content hash keying approach that ensures the content deduplication. This cryptographic mechanism is adapted to cloud storage environments by our second contribution CloudaSec, in order to handle secure data sharing among a group of users.

Chapter 4

CloudaSec: A Public Key based Framework to handle Data Sharing Security in Clouds

But the heaviest things, I think, are the secrets. They can drown you if you let them

Don't Judge a Girl by Her Cover
A. Carter - 2009

Contents

4.1 Introduction

Data security and privacy are among top concerns for the public cloud environments. Towards these security challenges, we present, in this chapter, CloudaSec framework for securely sharing outsourced data via the public cloud [KLEB14], for one or many recipients. CloudaSec, a public key based solution, ensures the confidentiality of content in public cloud environments with flexible access control policies for subscribers and efficient revocation mechanisms.
CloudaSec proposes the separation of subscription-based key management and confidentiality oriented asymmetric encryption policies. Consequently, our proposal enables flexible and scalable deployment of the solution as well as strong security for outsourced data in cloud servers.

CloudaSec provides several cryptographic tools for data owners, based on a novel content hash keying system. In fact, CloudaSec applies the convergent encryption concept [WQPW10] on the data file, as detailed in Section 4.3.3. That is, the data owner uploads enciphered content to the cloud, relying on a symmetric encryption algorithm. Then, he enciphers the decrypting key using CloudaSec key encryption procedures, before integrating the encrypted deciphering key in metadata. CloudaSec key encryption procedures are based on the hardness of computational problems in elliptic curve groups. This dual encryption on data then on decrypting keys permits data confidentiality preservation, as well as flexible access control policies.

In addition, CloudaSec integrates a conference key distribution scheme, based on parallel Diffie Hellman exchanges, proposed in [BD05], in order to derive a shared group secret. The key distribution algorithm is extended to support two user scenarios, for backward and forward secrecy purposes. That is, only authorized users can access metadata and decipher the decrypting keys. As such, user revocation is achieved without updating the private keys of the remaining users.
Beyond these security properties, a deduplication mechanism is deployed ensuring that only one copy of content is stored in cloud servers. This feature enables the efficient usage of storage capacities and achieves fast data distribution.

The remainder of this chapter is organized as follows. Section 4.2 presents security considerations and design goals. Then, Section 4.3 describes the system model, details the framework design, and describes the prototype and its different procedures. In Section 4.4, rigorous security discussions are given with proofs of correctness and implementation results are discussed in Section 4.5. Finally, we present a comparison between our contributions and some cryptographic mechanisms ensuring data confidentiality in clouds in Section 4.6, before concluding in Section 4.7.

4.2 Problem Statement

The design of CloudaSec is motivated by providing the support of both robustness and efficiency, while considering the limited storage and processing capacities of user devices. It has to fulfill the following requirements:

- ***Data confidentiality*** – our scheme has to protect the secrecy of outsourced data contents against both curious providers and malicious users.
- ***Flexible access control*** – CloudaSec should ensure flexible security policies among users with different granted privileges, belonging to different groups. These access control policies should guarantee backward and forward secrecy of outsourced data contents.
- ***Efficient user revocation*** – the revocation of a group member should not affect the remaining users. That is, contrary to traditional fine-grained access control schemes, the challenge is to define a smooth group revocation which does not require updating the secret keys of the non-revoked members.
- ***Low computation overhead*** – on one hand, for scalability reasons, the amount of computation at the cloud storage server should be minimized, as the server may be involved in concurrent interactions. On the other hand, the proposed algorithms should also have low processing complexity, at the client side.
- ***Low communication overhead*** – CloudaSec should minimize the usage of bandwidth, relying on low communication cost.
- ***Low storage cost*** – the limited storage capacities of the user devices has a critical importance in designing our solution. So, low storage cost at the client side is highly recommended.

Several security solutions have been recently developed, in order to provide data confidentiality in cloud storage environments [XZY+12, ZYG11, YWRL10a, ZVH11, LZWY13, Fug12], while considering access control challenges and user revocation concerns.
In [YWRL10a], Yu et al. proposed an attribute based access control policy to securely outsource sensitive client data to cloud servers. In this approach, data are encrypted using a symmetric encryption algorithm, while the enciphering key is protected by a KP-ABE scheme [GPea06]. To manage dynamic groups, they delegate the key re-encryption procedures to the cloud, without revealing the content of outsourced data. As such, the membership revocation mechanism brings additional computation overhead. CloudaSec defines a new revocation system based on [BD05], without updating the secret keys of the remaining group members, in order to minimize the complexity of key management. That is, our design conveys performance advantages for large scale sharing groups.

Several storage systems are based on the proxy re-encryption algorithms, in order to achieve fine grained access control [XZY+12, GPea06, AFGH]. When a recipient wants to retrieve outsourced data from the depositor, he has first to ask the cloud server to re-encrypt data file using its public key and the public master key, while considering the granted privileges. Ateniese et al. [AFGH] propose a bi-directional proxy re-encryption scheme to secure distributed storage systems and achieve efficient access control. However, a collision attack between the untrusted storage server and a revoked group member can be launched, which enables to learn the decryption keys of all encrypted blocks. In [XZY+12], the authors design an end-to-end content confidentiality protection mechanism for large scale data storage and distribution. They include many cryptographic mechanisms, namely the proxy re-encryption and broadcast revocation. Unfortunately, the subscription of a new user or the revocation of a group member requires the update of the entire group with new parameters and secret keys. That is, the complexity of user participation and revocation in their approach is linearly increasing with the number of data owners and the number of revoked users, respectively.

Recently, in order to achieve efficient membership revocation system, [LZWY13] adopts a group signature mechanism. They propose a multi-owner data sharing scheme, MONA, for dynamic groups in the cloud, while preserving identity privacy from untrusted servers. Nevertheless, MONA brings an extra storage overhead at both the cloud and the group manager side, for each outsourced data file.
In [SNDB13], Seo et al. propose an improved mediated certificateless approach, in order to secure data sharing in cloud servers. In fact, the basic concept of mediated cryptography is the usage of a security mediator (SEM) which can control security capabilities for the participating entities. Once the SEM is notified that a group member is revoked, it can immediately stop the user scenario. Unfortunately, this approach involves a trusted third party, in order to generate the partially decrypting keys. That is, it requires additional storage capacities and computation cost overhead, while considering flexible user management mechanisms.

4.3 CloudaSec Framework

To secure data sharing among a group of users, CloudaSec architecture introduces the role of a group manager, denoted as GM. The group manager takes charge of construction of a group, system parameters generation, user registration and user revocation. Figure 4.1 illustrates a descriptive network architecture for CloudaSec framework.
Next, we refer to these authorized user(s) as the recipient(s) and to the data owner as the depositor. We must note that CloudaSec does not require from the recipients to be connected during the sharing process. Indeed, recipients' access rights are granted by the data owner and managed by the CSP. That is, the CSP is in charge of verifying each recipient access permissions before sending him a redirected access key element.

4.3.1 CloudaSec Overview

To protect outsourced data in public cloud servers from unauthorized entities, CloudaSec provides several cryptographic tools for the data owner in order to guarantee the secrecy of his outsourced data and to ensure that only authorized users are able to obtain the decrypting data keys.

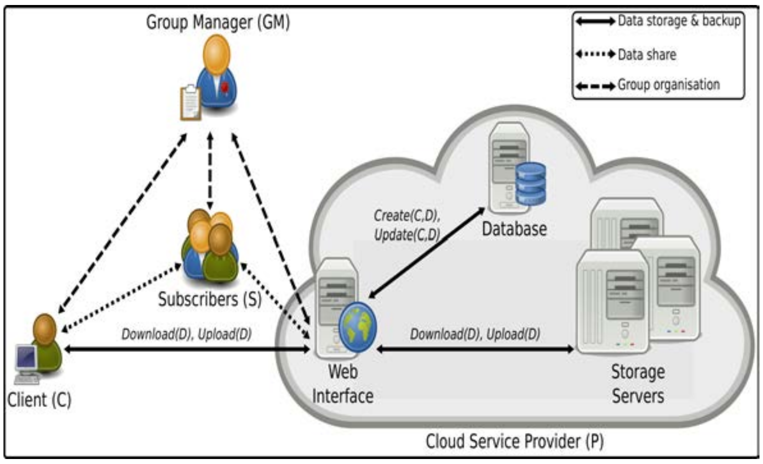

FIGURE 4.1 - CloudaSec architecture

Our framework relies on the convergent encryption [WQPW10] which is a content hash keying cryptographic system. That is, it presents two encryption levels: data encryption level and key encryption level as follows.

- ***symmetric data encryption level*** – before outsourcing data to cloud servers, the depositor encrypts file contents, using a symmetric algorithm. That is, the enciphering data key is derived, from the file plaintext, using a one way hash function. Hence, the choice of the convergent encryption is multifold. First, storage capacity is preserved as the same data encrypted by several users produce the same encrypted data that need to be stored once. As such, the number of redundant copies is minimized in order to preserve the efficiency of the storage service. Second, convergent encryption leads to a per-data enciphering key thus mitigating the usual key sharing problem when content sharing is needed. Third, the generation of the deciphering data key is possible only if the plaintext is known.

- ***asymmetric key encryption level*** – the depositor enciphers the decrypting data key k, based on an asymmetric algorithm, using the public key of the recipient. Then, he includes this resulted encrypted key in user metadata, ensuring flexible access policies. Indeed, any authorized recipient may access to user metadata, in order to decipher the encrypted data key, using his private key. Then, he can decrypt the enciphered contents.

This dual encryption scheme on data then on the decrypting keys provides data confidentiality, as well as flexible access control policies.
CloudaSec procedures involve two joint layers: data layer and management layer. In the data layer, we introduce the operations on data and the related enciphering keys, namely $GenerateParameters$, $EncryptData$, $DecryptData$,
$EncryptKey_{OneToOne}$, $EncryptKey_{OneToMany}$ and $ShrinKey$ procedures. In the management layer, CloudaSec introduces procedures of *user revocation*, when a group member leaves or is revoked from the group, and *user subscription*, when a new user joins the group.

CloudaSec supports flexible access to encrypted contents, by dynamically sharing a group secret key within the group. That is, when the group state is modified due to a user subscription or revocation, the GM broadcasts the new group arrangement to authorized members in order to generate the new secret group key, based on the published public elements, without updating the private keys of the remaining users, as presented in Section 4.3.4.
CloudaSec distinguishes two different data sharing scenarios. First, the data sharing one to one, presented in Section 4.3.3.1, where a data owner stores for one CloudaSec user. Second, the data sharing one to many, described in Section 4.3.3.2, where a data owner shares data among a group of authorized users. These scenarios encompass two different data key encryption algorithms $EncryptKey_{OneToOne}$ and $EncryptKey_{OneToMany}$.

The different notations used in CloudaSec are listed in Table 4.1.

TABLE 4.1 - CloudaSec Notations

Notation	Description
f	file content
k	data key
id_i	identity of a CloudaSec user U_i
sk_i	private key of a CloudaSec user U_i
pk_i	public key of a CloudaSec user U_i
sk_c	private key of the CSP
pk_c	public key of the CSP
d	group secret key

4.3.2 Cryptographic Background

This section reviews a straightforward cryptographic background, used in the design of our CloudaSec framework. We first define the used cryptographic functions, namely bilinear maps and collision resistant hash functions, which are involved in the derivation of the data encrypting keys. Then, we present the group key distribution scheme, adopted by CloudaSec in order to ensure the generation of group keys.

4.3.2.1 Preliminaries

CloudaSec essentially relies on the use of one way functions and bilinear maps, defined as follows.

Collision resistant hash functions [BB06] – Let $H : \{0,1\}^* \rightarrow \{0,1\}^n$ be a hash function. H is a collision resistant function if no efficient algorithm can find a pair $M \neq M' \in \{0,1\}^*$, such that $H(M) = H(M')$.

Bilinear maps { [Ken], [RRP04]} – as exposed in Section 2.3.1.1, an admissible symmetric pairing function $\hat{e}$ from $\mathbb{G}_1 \times \mathbb{G}_1$ in $\mathbb{G}_2$ has to be bilinear, non degenerate and efficiently computable. $\mathbb{G}_1$ is an additive subgroup of the group of points of an Elliptic Curve (EC). However, $\mathbb{G}_2$ is a multiplicative subgroup of a finite field. $\mathbb{G}_1$ and $\mathbb{G}_2$ have the same order q. In addition, $\mathbb{G}_1$ and $\mathbb{G}_2$ are generated by P and the $g = \hat{e}(P, P)$, respectively.

4.3.2.2 Group Key Distribution (GKD)

Burmester and Desmedt propose an unauthorized key exchange protocol [BD05]. It is a two round protocol that extends the concept of the Diffie Hellman assumption, as defined in Chapter 2, Section 2.4.1.
Let $G = \{U_1, ...U_m\}$, be a group of m users arranged into a cycle. To generate a group key k, each member U_i, where $i \in [1, m]_{\mathbb{N}}$, first selects a random secret b_i. Then, he broadcasts $z_i = g^{b_i}$, where g is a nonzero generator of a multiplicative group $\mathbb{G}_p{}^*$. Afterwards, each member U_i publishes $X_i = (\frac{z_{i+1}}{z_{i-1}})^{b_i}$. We must note that the number of exponentiations per user is constant and each user U_i computes the group secret key k, as $k \equiv g^{b_1 b_2 + b_2 b_3 + \cdots + b_{i-1} b_i + b_i b_{i+1} + \cdots + b_m b_1} \bmod(p)$.

4.3.3 CloudaSec Procedures in Data Plane

This section describes the different CloudaSec data layer procedures. CloudaSec, first, requires a system setup procedure, ensured by the execution of the *GenerateParameters* algorithm, before performing the sharing scenarios.
This CloudaSec *GenerateParameters* algorithm is executed only once, to initialize the system. It generates the public parameters *params*, according to a required security parameter ξ, as presented in Algorithm 1.
After the specification of the groups, CloudaSec *GenerateParameters* procedure defines a secure one way hash function $H : \mathbb{E} \to \{0, 1\}^l$, with respect to the required security level, where $\mathbb{E}$ represents the finite data domain and l is the length of the content encrypting key. In addition, it derives an application F to bind an element belonging to the multiplicative group $\mathbb{G}_2^*$ to a binary sequence of length l.
The groups $\mathbb{G}_1$ and $\mathbb{G}_2$, the pairing $\hat{e}$, the point P, the hash function $H()$ and the application F form the public parameters *params* as follows:

$$params = \{\mathbb{G}_1, \mathbb{G}_2, n, \hat{e}, g, P, H(), F\}.$$

We must note that each user has to derive a pair of public and private keys, with respect to the published authentic public parameters *params* and the required security level ξ. As such, a CloudaSec user U_t is characterized by his identity id_t and the derived pair of keys: his private key sk_t, where sk_t is a random secret $s_t \in_R \mathbb{Z}_n$ and his public key as $pk_t = s_t \cdot P$.

$$U_t(id_t, pk_t, sk_t)$$

We must also note that the CSP has a pair of private and public keys as $< sk_c, pk_c >$, where $sk_c = s_c \in_R \mathbb{Z}_n$ presents the provider private key and $pk_c = s_c \cdot P \in \mathbb{G}_1^*$ is his related public key. The CSP public key is derived based on the hardness of the ECDLP. In the

Algorithm 1 GenerateParameters

```
1: Input: Security parameter $\xi$
2: Output: System parameters $params = \{\mathbb{G}_1, \mathbb{G}_2, \hat{e}, P, g, H, F, n\}$
3: Choose an elliptic curve $EC$ over an additive subgroup $\mathbb{G}_1$ of a prime order $n$, where
   $BitLength\ (n) > \xi$ and ECDLP is hard in $\mathbb{G}_1$;
4: Select $P$ a generator of $EC$;
5: Choose a multiplicative subgroup $\mathbb{G}_2$ of a prime order $n$, where $BitLength\ (n) > \xi$ and
   CDH is hard in $\mathbb{G}_2$;
6: Select $g$ a generator of $\mathbb{G}_2$;
7: Generate $\hat{e}$ from $\mathbb{G}_1 \times \mathbb{G}_1$ in $\mathbb{G}_2$ an admissible pairing map;
8: Generate a one way hash function $H : \mathbb{E} \rightarrow (\{0,1\}^l)^*$, where $\mathbb{E}$ is the data space and $l$
   is the length of the encrypting key;
9: Generate $F : \mathbb{G}_2^* \rightarrow \{0,1\}^l$ an application to bind an element of $\mathbb{G}_2^*$ to a binary sequence
   of length $l$
10: return $params = \{\mathbb{G}_1, \mathbb{G}_2, \hat{e}, P, g, H, F, n\}$
```

following, we denote by $\cdot$ the scalar point multiplication in an additive group and by $\star$ two elements multiplication belonging to a multiplicative group.
We consider a data sharing process, where the client outsources his data to the cloud and authorizes a group of users to access the data. This group may be a duo group or a multi-user group.

4.3.3.1 CloudaSec One to One Sharing Scenario

The one to one sharing scenario is defined when a data owner U_i wants to share data with only one recipient user U_j. The depositor U_i first enciphers the data file f, as presented in Algorithm 2, based on a symmetric encryption scheme *SymEnc*, using a data enciphering key k. Based on a convergent cryptographic solution, the data key k is derived from the application of a one way hash function over the original data file f. Subsequently,

Algorithm 2 EncryptData

```
1: Input: $\{f, H, SymEnc\}$, where $f$ is the data file, $H$ is a one way hash function and
   $SymEnc$ is a symmetric encryption algorithm
2: Output: $< C_f, k >$
3: $k = H(f)$;
4: $C_f = SymEnc(f, k)$;
5: return $< C_f, k >$
```

U_i stores the encrypted content C_f for the recipient user U_j, in remote servers. In order to assign the access rights to the recipient, the depositor enciphers the data decrypting key k using the public key of the recipient pk_j, as described in CloudaSec $EncryptKey_{OneToOne}$ procedure (cf. Algorithm 3). That is, CloudaSec introduces a novel asymmetric key encoding, to ensure flexible sharing of outsourced data. For instance, the resulting enciphered key involves a couple of elements $< C_1, C_2 >$. C_1 is included in the user metadata, by the depositor U_i. However, C_2 is sent to the CSP, in order to grant additional access verifications on the outsourced data and to generate a redirected access key element. We assume

Algorithm 3 EncryptKey$_{OneToOne}$

```
1: Input: $\{params, k, sk_i, pk_i, pk_j, pk_c\}$
2: Output: $< C_1, C_2 >$
3: Generate $r \in_R \mathbb{Z}_n$;
4: $C_1 = k \oplus F(\hat{e}(pk_i, pk_j)^r)$;
5: $C_2 = \hat{e}(pk_c, r \cdot P)^{sk_i}$;
6: return $< C_1, C_2 >$
```

in our approach that all key elements belong to a finite domain space D. We denote each key element by ***key.elt*** as defined in Equation 4.1, where D can be either a user metadata element space or a CSP metadata element space.

$$key.elt_{i\in\{1,2,3\}} = \{C_i, D(C_i)\}_{i\in\{1,2,3\}} \tag{4.1}$$

When the CSP receives the second key element C_2, he runs the *ShrinKey* algorithm in order to derive a redirected key element C_3, as presented in Algorithm 4. This latter encodes C_2, using his secret key sk_c and generates the corresponding C_3.

Algorithm 4 ShrinKey

```
1: Input: $\{C_2, sk_c\}$
2: Output: $C_3$
3: $C_3 = (C_2)^{\frac{1}{sk_c}}$;
4: return $C_3$
```

Figure 4.2 depicts a schematic diagram which presents the relation between key encryption procedures of CloudaSec one to one sharing scenario.

Afterwards, when the CloudaSec recipient U_j wants to recover the outsourced data file, he has to retrieve the encrypted data key $< C_1, C_3 >$. As such, the recipient user U_j starts a data backup scenario. In fact, after successfully authenticating with the CSP, U_j requests the redirected key element C_3. Then, based on the outsourced user metadata, the authorized recipient U_j extracts the C_1 key element, which was enciphered, using his public key pk_j by the depositor U_i. In the sequel, relying on his local secret key sk_j, the recipient U_j performs the *DecryptKey$_{OneToOne}$* procedure, in order to decipher the encrypted data key k, as presented in Algorithm 5. Finally, the recipient retrieves the

Algorithm 5 DecryptKey$_{OneToOne}$

```
1: Input: $\{params, < C_1, C_3 >, sk_j\}$
2: Output: Decrypting key $k$
3: $C_1 \oplus F((C_3)^{sk_j})$;
4: return $k$
```

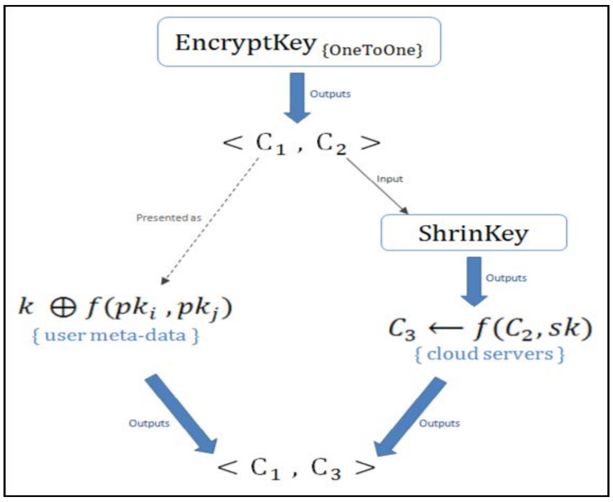

FIGURE 4.2 - Schematic diagram of CloudaSec one to one sharing procedures

data file content. That is, he locally runs the CloudaSec *DecryptData* procedure, based on the derived deciphering key k, using a symmetric algorithm over encrypted data C_f (cf. Algorithm 6).

Algorithm 6 DecryptData

```
1: Input: $\{C_f, k, SymEnc\}$
2: Output: $f$
3: $f = SymEnc(C_f, k)$ ;
4: return $f$
```

4.3.3.2 CloudaSec One to Many Sharing Scenario

When a depositor U_i intends to share data with a multi-user group, he has to encipher the data decrypting key based on his public key pk_i and a secret shared group key d. The secret shared key is a private key, only known to the authorized group members. It is derived by performing the key agreement algorithm (Section 4.3.2.2), based on parallel Diffie Hellman instantiations [BD05], as explained in Section 4.3.4.
The depositor executes the $EncryptKey_{OneToMany}$ procedure (cf. Algorithm 7), in order to encrypt the deciphering data key. The resulting encrypted key includes a couple of elements $< C_1, C_2 >$, where C_1 is integrated in user metadata by the depositor, and C_2 is sent to the cloud provider, in order to generate an accessing key C_3 element (cf. Algorithm 4). When an authorized group member wants to retrieve the data decrypting key, he has first to send a request to the cloud provider to access to the outsourced data. The CSP verifies the granted privileges of the requesting user. Once accepted, the requesting group member receives the redirected key element C_3 obtained by performing the *ShrinKey* procedure, as shown in Section 4.3.3.1. Then, he runs the CloudaSec $DecryptKey_{OneToMany}$ procedure

Algorithm 7 EncryptKey$_{OneToMany}$

```
1: Input: $\{params, k, pk_i, sk_i, d, pk_c\}$
2: Output: $< C_1, C_2 >$
3: Use a deterministic secure pseudo random number generator (SPRNG) with a random
   secret seed to generate $r \in_R \mathbb{Z}_n$;
4: $C_1 = k \oplus F(\hat{e}(pk_i, r \cdot P)^d)$;
5: $C_2 = \hat{e}(pk_c, r \cdot P)^{sk_i}$;
6: return $< C_1, C_2 >$
```

(cf. Algorithm 8) using the secret shared group key d, in order to derive the deciphering data key k.

Algorithm 8 DecryptKey$_{OneToMany}$

```
1: Input: $\{params, < C_1, C_3 >, d\}$
2: Output: Decrypting key $k$
3: $C_1 \oplus F((C_3)^d)$;
4: return $k$
```

4.3.4 CloudaSec Management Layer Procedures

Efficient data sharing between authorized cloud users, among dynamic groups remains a challenging concern. That is, it increases the computation complexity and the bandwidth consumption, due to the sharing of group secret keys. In addition, the heavy overhead and the large size of outsourced data may reduce the advantages of remote sharing services to resource-constrained devices.

In order to tackle this challenging issue, CloudaSec introduces the role of a group manager (GM). This latter is responsible for elementary procedures, namely the initialization of the group parameters and the organization among authorized registered group members. Then, the GM makes the group parameters available by migrating them to the cloud. Such a design can significantly reduce the computation overhead, at the CloudaSec user side.

Let us consider $Gr = \{\{U_0, id_0\}, ..., \{U_{N-1}, id_{N-1}\}\}$ a dynamic group of N users. These group members want to generate a common secret $d \in \mathbb{Z}_n$. In the following, we denote by $pubelts_i$ the public elements of a CloudaSec registered group member U_i as described in Equation 4.2.

$$pubelts_i =< id_i, pk_i > \tag{4.2}$$

where $i \in \{1, ..., N-1\}$ and N is the number of group users including the manager. As such, we note that the couple $< id_0, pk_0 >$ presents the public group elements of the *Group Manager* (GM). First, the GM runs a *GenerateGroup* procedure, in order to derive a multiplicative group and makes public the output of this algorithm, which is used to generate the secret group key d (cf. Algorithm 9). We must note that the order of the multiplicative group is strongly associated to the security level ξ of the cryptographic algorithms.

Then, with respect to the published multiplicative group $\mathbb{G}$, each group user U_i chooses a

Algorithm 9 GenerateGroup

```
1: Input: $n, p, \xi$
2: Output: $< \mathbb{G}, h >$
3: Choose a multiplicative subgroup $\mathbb{G}$ of a prime order $n$, where $BitLength\ (n) > \xi$;
4: Select $h$ a generator of $\mathbb{G}$, where $h^n \equiv 1 \bmod p$;
5: return $< \mathbb{G}, h >$
```

random b_i and locally runs the CloudaSec *UserKeyShareElt* procedure in order to get his first key share element h_i, as presented in Algorithm 10.
The GM receives the public elements $pubelts_i$ of each group member U_i. Then, he updates

Algorithm 10 UserKeyShareElt

```
1: Input: $id_i, b_i, h$
2: Output: $< id_i, h_i >$
3: $h_i = h^{b_i} \in \mathbb{G}^*$;
4: return $< id_i, h_i >$
```

a list of non revoked users L_{NR}, which contains the public elements of all non revoked group users. This list sets the authorized group members arranged into a cycle. As such, each user can easily identify his predecessor U_{i-1} and his successor U_{i+1}. Thus, using the CloudaSec *GenerateGroup* and *UserKeyShareElt* procedures, U_i computes his group key share (h_i, X_i), as depicted in Equation 4.3.

$$(h_i, X_i) = (h^{b_i}, (\frac{h_{i+1}}{h_{i-1}})^{b_i}) \tag{4.3}$$

Once computed, each user U_i sends his group key share to the GM. This latter publishes the received key shares of the non revoked users, as presented in Table 4.2. Afterwards, as presented in Section 4.3.2.2, each user should derive the secret group key d, using the published elements in the L_{NR} list, while respecting the ring construction of the group members [BD05].

TABLE 4.2 - List of Non Revoked users L_{NR}

Group id	**User Pubelts**	**User Key Share**
id_{Gr}	$U_0(id_0, pk_0)$	(h_0, X_0)
	$U_1(id_1, pk_1)$	(h_1, X_1)
	$\vdots$	$\vdots$
	$U_{N-1}(id_{N-1}, pk_{N-1})$	(h_{N-1}, X_{N-1})

4.3.4.1 User Subscription

When a new user $\{U_N, id_N\}$ joins the group $Gr = \{\{U_0, id_0\}, ..., \{U_{N-1}, id_{N-1}\}\}$, he first runs the *UserKeyShareElt* algorithm in order to get his public key share element $< id_N, h_N >$. Then, the new group member computes and sends his key share (h_N, X_N) to the group manager. Hence, The GM sends a notification message to the remaining group members and updates the list of non revoked users L_{NR}.
Afterwards, each group user computes the new secret key d_N, due to the group state modification. Since the derivation of the group secret key depends on members' identifiers, the computation of key shares (h_i, X_i) may be restricted to the solicited members. Consequently, CloudaSec significantly saves the processing time and storage cost at CloudaSec user side.
The user subscription operation prevents new users from accessing to protected content, before joining the group. As such, CloudaSec ensures the forward security. In order to grant access privileges to new subscribers to outsourced data, the sharing of a secrets' list L_S is required. Indeed, the group manager updates a list of previously used secrets L_S by including the new group secret key. Then, he sends it to the CSP in an encrypted format by using symmetric encryption algorithm and the derived secret group key d. In the sequel, any authorized group member authenticates with the CSP and uploads the encrypted list. So that, he can obtain L_S using the derived secret group key and the symmetric decryption algorithm.

4.3.4.2 User Revocation

When a member U_j leaves or is revoked from the group $Gr = \{id_0, id_1, id_2, ..., id_N\}$, where $j \in \{0, 1, ..., N\}$, the group manager first updates the list of non revoked users L_{NR}. That is, he removes the public elements $< id_j, pk_j >$ and the key share (h_j, X_j) of the revoked member from the L_{NR} list. Then, he sends a notification message to other group users and sends the updated list to the CSP. Each group user computes a new secret key d_N by running the *GroupKey* algorithm.
We note that the number of the revoked users *RU* has to be strictly less than $(N-1)$, in order to keep the one to many sharing scenario. In fact, we consider two cases.

1. *Case 1* – There are *RU* revoked users, where $1 \leq RU \leq N-2$. The group manager revokes *RU* users, and updates the L_{NR} list. That is, he withdraws the revoked users' identities and reorganizes the indexing system of the list. The group manager optimizes the changes of the group list, in order to save the computation capacities of resource constrained devices. As such, a non solicited group member is requested to only compute the resulting group key, using the published public group elements.

2. *Case 2* – There are *RU* revoked users, where $RU \geq N-1$. In this case, the group manager is released from his role. As such, the multi-user group becomes a duo group that shares data following the one to one sharing process.

4.4 Security Analysis

In the following security analysis, we discuss the resistance of CloudaSec against two adversaries, based on a realistic threat model. We briefly present the security of our proposed framework in terms of access control and data file confidentiality.

4.4.1 Threat Model

For designing the most suitable security solutions for CloudaSec, we considered two threat models. That is, we point out two adversaries: malicious cloud user and *honest but curious* cloud server, as follows:

- *Malicious user adversary* – an attacker can be either a revoked user with valid data decryption keys, an unauthorized group member or a group member with limited access rights. As such, he targets to get access to the outsourced shared data. The objective of a malicious user is to convince the cloud server that he is a legitimate data owner. That is, we suppose that the adversary successes to gain knowledge of an arbitrary part of the decrypting key.

- *honest but curious cloud server adversary* – this storage server honestly performs the operations defined by our proposed scheme, but it may actively attempt to gain the knowledge of the outsourced sensitive data.

CloudaSec must provide the capabilities to the clients and the service provider to thwart the two threats mentioned above. To this end, our proposed framework must enforce a mutual verification of the actions conducted by a CloudaSec client and the storage server.

4.4.2 Data Confidentiality

In our model, data files are stored encrypted in cloud servers using a symmetric encryption algorithm, and the secret key is protected relying on an asymmetric scheme, in order to ensure efficient access control. As such, the data confidentiality preservation is tightly related to security of the used symmetric algorithm and the secrecy of the data key.

Theorem 4.4.1 ***data confidentiality preservation***
The proposed framework supports data confidentiality preservation.

Proof.
The confidentiality of data contents is twofold. First, it depends on the security level of the encryption algorithm. This latter is a recurrent concept in cryptography. It permits to evaluate the hardness of breaking an encryption or a signature algorithm. That is, the harder the level of security is, the harder the cryptanalysis of the algorithm becomes. Our employed encryption algorithm inherits the unforgeable property from the selected scheme. Therefore, CloudaSec ensures the confidentiality of encrypted content exposed in public cloud servers.
Second, the confidentiality of data relies also on the secrecy of the deciphering key hosted in cloud servers. The demonstration of this state is derived from the following two lemmas. □

Lemma 4.4.2 *Unauthorized users cannot decrypt the deciphering data keys.*

Proof.
The proof of this lemma is equivalent to the security of the key encryption algorithms and the correctness of the key decryption algorithms.
Let us suppose that an unauthorized user can be a revoked group member or a malicious cloud user. Thus, a brief security analysis can be done based on the three following cases.

- *Case A* – a revoked group member U_R should not be able to decrypt new data contents, using the old group secret key d. This latter knows the public elements of the non revoked users published in L_{NR} and the previous organization of the group arranged into a cycle. Moreover, he can merely guess the solicited members after his revocation. As such, taking advantage from published information, U_R tries to deduce the new group secret key d_N or to extract a data key after his revocation from the group. In this case, we may consider two different sessions (α) and (β), where the same data owner U_i shares two different data files f_α and f_β, after the revocation of U_R. In the sequel, two key elements are defined as follows:

$$C_1^{(\alpha)} = k_\alpha \oplus F(\hat{e}(pk_i, r_\alpha \cdot P)^{d_N}) \tag{4.4}$$

$$C_1^{(\beta)} = k_\beta \oplus F(\hat{e}(pk_i, r_\beta \cdot P)^{d_N}) \tag{4.5}$$

  On one side, knowing the public key of the depositor pk_i, we state that the deduction of the new group secret d_N from $C_1^{(\alpha)}$ cannot hold. Obviously, this is due to the usage of a random value r_α. We also state that our scheme inherits the unforgeablility property from the Burmester key distribution algorithm [BD05]. On the other side, U_R cannot deduce secret information from $C_1^{(\alpha)} \oplus C_1^{(\beta)}$, mainly due to the exclusive-or function.
  As such, a revoked group member U_R has no advantage to guess the new secret group, based on the old group key d and the previously published public elements. However, we must note that CloudaSec does not prevent a revoked member from decrypting previously shared contents.

- *Case B* – the main advantage of a malicious cloud user is to deduce information from an unbounded number of sessions, where the same data owner shares different

contents with legitimate group members. As such, two different cases are exposed as follows.
On one hand, in a one to many sharing scenario, let us suppose that a user U_i shares two data files f_1 and f_2, respectively enciphered based on two different keys k_1 and k_2, using the same random r. That is, based on Equation 4.6 and Equation 4.7, an attacker obtains indistinguishable data key elements $key.elt_1$, as follows.

$$C_1^{(f_1)} = k_1 \oplus F(\hat{e}(pk_i, r \cdot P)^d) \tag{4.6}$$

$$C_1^{(f_2)} = k_2 \oplus F(\hat{e}(pk_i, r \cdot P)^d) \tag{4.7}$$

Hence, we notice that there is no Probabilistic Polynomial Time (PPT) algorithm that can deduce secret information from Equation 4.6$\oplus$Equation 4.7 $= k_1 \oplus k_2$, thanks to the usage of the exclusive-or function. As such, the secrecy disclosure of the deciphering keys remains infeasible.
On the other hand, in a one to one sharing scenario, we suppose that a depositor U_i shares data with one recipient U_j, using the same random secret r, in order to encipher different data decrypting keys. Thus, based on two successive sessions (α) and (β), the malicious cloud user gets the following key elements $key.elt_1$:

$$C_1^{(\alpha)} = k_1 \oplus F(\hat{e}(pk_i, pk_j)^{r^{(\alpha)}}) \tag{4.8}$$

$$C_1^{(\beta)} = k_2 \oplus F(\hat{e}(pk_i, pk_j)^{r^{(\beta)}}) \tag{4.9}$$

Therefore, the deduction from the enciphered contents is protected, due to the usage of different data encryption keys. In addition, the security of metadata takes advantages from the properties of the exclusive-or function which ensure the indistinguishability of encryptions.

- *Case C*− let us suppose that a depositor U_i belongs to two different groups G_A and G_B. U_i wants to share the same data file f to these two groups.
  G_A and G_B have two secrets d_A and d_B, respectively. As such, we have two different key elements $key.elt_1$, as follows:

$$C_1^{(A)} = k \oplus F(\hat{e}(pk_i, r_A \cdot P)^{d_A}) \tag{4.10}$$

$$C_1^{(B)} = k \oplus F(\hat{e}(pk_i, r_B \cdot P)^{d_B}) \tag{4.11}$$

  We suppose that there a malicious group member U_M that belongs to the group of users G_A. U_M tries to deduce the group secret key d_B of the group G_B.
  From Equation 4.10, the recipient U_M extracts the data deciphering key k. In the sequel, from $C_1^{(B)}$, U_M computes Equation 4.12 as follows:

$$\begin{aligned} C_1^{(B)} \oplus k &= k \oplus F(\hat{e}(pk_i, r_B \cdot P)^{d_B}) \oplus k && (4.12) \\ &= F(\hat{e}(pk_i, r_B \cdot P)^{d_B}) && (4.13) \end{aligned}$$

  From Equation 4.11 and Equation 4.12, the malicious recipient U_M calculates $\hat{e}(pk_i, r_A \cdot P)^{d_A} \star [1/(\hat{e}(pk_i, r_B \cdot P)^{d_B})]$. So that, U_M executes the following steps:

$$\hat{e}(sk_i \cdot P, r_A \cdot P)^{d_A} \star [\frac{1}{\hat{e}(sk_i \cdot P, r_B \cdot P)^{d_B}}] = \frac{g^{r_A d_A}}{g^{r_B d_B}}$$

Knowing the group secret d_A and the two quantities $g^{r_A d_A}$ and $g^{r_B d_B}$, U_M cannot extract the group secret d_B. Obviously, this contradicts the hardness of the CDH assumption.
Finally, as data may be shared by different depositors or groups, and these depositors may belong to several groups, our framework strongly ensures the confidentiality of outsourced contents, based on the hardness of the CDH assumption.

□

Lemma 4.4.3 *The CSP is unable to learn the content of outsourced data files in his public servers, based on the CDH assumption.*

Proof.
A curious CSP tries to access to the stored data contents. His main problem is that outsourced data files are encrypted. However, the CSP tries to gain knowledge of an arbitrary part of secret information.
That is, after each storage of data content, the CSP receives the second key element $key.elt_2$, from the depositor U_i as $C_2 = \hat{e}(pk_c, r \cdot P)^{sk_i}$, where $sk_i \in \mathbb{Z}_n$ is the secret element of the depositor U_i and $pk_c = sk_c \cdot P \in \mathbb{G}_1^*$ is the public key of the storage provider. As such, in order to extract secret information, the CSP computes $\hat{e}(pk_i, P) = g^{sk_i}$. This deduction cannot hold. Clearly, this contradicts the CDH assumption.
Given the redirected key element $C_3 = C_2^{\frac{1}{c}} = (\hat{e}(pk_c, r \cdot P)^{sk_i})^{\frac{1}{c}}$, the CSP computes $C_3 = (\hat{e}(pk_i, r \cdot P)^{sk_c})^{\frac{1}{c}} = \hat{e}(pk_i, r \cdot P)$. Then, he executes Equation 4.14 and Equation 4.15 as follows:

$$C_2 \star [\frac{1}{\hat{e}(pk_c, pk_i)}] = \frac{g^{sk_i sk_c r}}{g^{sk_i sk_c}} = g^r \tag{4.14}$$

$$C_3 \star [\frac{1}{\hat{e}(pk_i, pk_i)}] = \frac{g^{sk_i r}}{g^{{sk_i}^2}} = \frac{g^r}{g^{sk_i}} \tag{4.15}$$

From Equation 4.14 and Equation 4.15, this curious storage server cannot extract the secret key of the depositor sk_i or the used random value r. As such, the storage server cannot learn the content of the outsourced data files or deduce extra-information, based on the hardness of the CDH assumption. □

4.4.3 Access Control

CloudaSec is designed to ensure forward and backward secrecy. When a new user joins the group or a group member is revoked, a notification message is sent to CSP and to the remaining members, in order to adjust the access control lists.
On one side, when a new user U_N joins the group, he has to generate his own group public elements $pubelts_N$. These elements will be later used to derive the new group key d_N. On the other side, when a user U_R leaves the group, the GM updates the sharing lists, in order to generate the new decrypting key. Consequently, a new user cannot decrypt the old data files, using the new derived key, and a revoked user cannot decrypt new files, with the old deciphering key.
The access control preservation is ensured, based on the following two lemmas.

Lemma 4.4.4 ***Key Decryption Correctness*** *Unrevoked users are able to access the cloud.*

Proof.
The proof of this lemma is equivalent to the correctness of the key decryption algorithms, on the basis of the two sharing scenarios, as follows.

- *One To One* sharing scenario – the decryption holds if, and only of the decrypting key k^* is equal to $C_1 \oplus F((C_3)^{sk_j})$.
  This verification holds as follows. On one side, the authorized recipient U_j computes the data key element included in user metadata as follows:

$$\begin{aligned} C_1 &= k \oplus F(\hat{e}(pk_i, pk_j)^r) \\ &= k \oplus F(\hat{e}(sk_i \cdot P, sk_j \cdot P)^r) \\ &= k \oplus F(\hat{e}(s_i \cdot P, s_j \cdot P)^r) \end{aligned}$$

  On the other side, the cloud provider sends the redirected key element C_3 to the CloudaSec recipient, which is computed as follows.

$$\begin{aligned} C_3 &= (C_2)^{\frac{1}{sk_c}} \\ &= (\hat{e}(pk_c, r \cdot P)^{sk_i})^{\frac{1}{s_c}} \\ &= (\hat{e}(sk_c \cdot P, r \cdot P)^{s_i})^{\frac{1}{s_c}} \\ &= (\hat{e}(s_c \frac{1}{s_c} \cdot P, r \cdot P)^{s_i}) \\ &= (\hat{e}(P, r \cdot P)^{s_i}) \end{aligned}$$

  In the sequel, given the non singularity property of the bilinear functions, the verification holds if, and only if $k^* = C_1 \oplus F((C_3)^{sk_j})$, where $C_1 \oplus F((C_3)^{sk_j})$ is denoted by (E).

$$\begin{aligned} (E) &= k \oplus F(\hat{e}(sk_i \cdot P, sk_j \cdot P)^r) \oplus F(\hat{e}(P, r \cdot P)^{sk_i sk_j}) \\ &= k \oplus F(\hat{e}(s_i \cdot P, s_j \cdot P)^r) \oplus F(\hat{e}(s_i \cdot P, r \cdot P)^{s_j}) \\ &= k \oplus F(\hat{e}(s_i \cdot P, s_j \cdot P)^r) \oplus F(\hat{e}(s_i \cdot P, s_j \cdot P)^r) \\ &= k \end{aligned}$$

- *One To Many* sharing scenario – the decryption holds if, and only of the data key $k^* = C_1 \oplus F((C_3)^d)$. This verification holds as follows.
  On one hand, the authorized group member U_j computes the data key element C_1 included in user metadata as follows:

$$\begin{aligned} C_1 &= k \oplus F(\hat{e}(pk_i, r \cdot P)^d) \\ &= k \oplus F(\hat{e}(sk_i \cdot P, r \cdot P)^d) \end{aligned}$$

On the other hand, the CSP executes the following operations on $key.elt_2$, in order to get the redirected element C_3

$$\begin{aligned} C_3 &= (C_2)^{\frac{1}{sk_c}} \\ &= (\hat{e}(pk_c, r \cdot P)^{sk_i})^{\frac{1}{s_c}} \\ &= (\hat{e}(sk_c \cdot P, r \cdot P)^{s_i})^{\frac{1}{s_c}} \\ &= (\hat{e}(s_c \frac{1}{s_c} \cdot P, r \cdot P)^{s_i}) \\ &= (\hat{e}(P, r \cdot P)^{s_i}) \end{aligned}$$

As presented in the *One To One* sharing scenario, given the non singularity property of the bilinear functions, the verification holds if, and only if $k^* = C_1 \oplus F((C_3)^d)$, where $C_1 \oplus F((C_3)^d)$ is denoted by (F).

$$\begin{aligned} (F) &= k \oplus F(\hat{e}(sk_i \cdot P, r \cdot P)^d) \oplus F(\hat{e}(P, r \cdot P)^{sk_i d}) \\ &= k \oplus F(\hat{e}(s_i \cdot P, r \cdot P)^d) \oplus F(\hat{e}(s_i \cdot P, r \cdot P)^d) \\ &= k \end{aligned}$$

We state that the authorized CloudaSec users are able to decipher the decrypting data key, thanks to the correctness of the key decryption algorithms. □

Lemma 4.4.5 *Unauthorized entities are unable to access the cloud.*

Proof.
The proof of this lemma is twofold.
On one side, after each group member U_R revocation, the group manager updates the list L_{NR} and sends a notification message to the authorized registered group members. Then, he communicates this list to the cloud provider. This latter sends L_{NR} to the remaining group members after a mutual authentication, in order to verify the updated organization of the group.Then, the remaining group members compute the new secret group key d_N by performing the *GroupKey* algorithm. Therefore, the new data keys are encrypted by using $EncryptKey_{OneToMany}$ algorithm.
As discussed in Lemma 4.4.2 and Lemma 4.4.4, only authorized recipients, knowing the new key d_N, are able to decrypt the enciphered data. However, the CSP and the revoked users cannot extract the key d_N, based on the previously published public elements and the list of authorized members L_{NR}. This is mainly due to the computation of the group secret which requires the private secret key sk_i of each deriving group member U_i.
On the other side, when a group user wants to access the cloud, the CSP has to verify the access control list. That is, the cloud provider gives or rejects access to data contents, based on the granted privileges of the requesting recipient. □

4.5 Performance Evaluation

In this section, we first present the context of CloudaSec implementation with OpenStack Object Storage, and then evaluate the system performances, in terms of computation, communication and storage costs.

4.5.1 Context

In order to evaluate the performances of our proposal, we build a simulated CloudaSec framework, based on OpenStack Storage system (Swift) [chac]. Swift is a cloud based storage system, which stores data and allows write, read, and delete operations on them. To achieve security enhancement of Swift, we extend its functionalities with algorithms and protocols designed in CloudaSec.
We have designed a simplified CloudaSec architecture, based on Swift. Indeed, our architecture consists in dividing the machine drive into four physical volumes. Then, each volume is divided into four logical volumes. In total, we obtain sixteen partitions, each one represents one logical storage zone.
The simulation consists of two components: the client side and the cloud side. We implement *data layer* cryptographic algorithms based on cryptographic functions from the Open-SSL library [The03], the GMP library [ea02] and the Pairing Based Cryptography (PBC) library [Ben07], with independent native processes. We choose Advanced Encryption Standard (AES) as our symmetric encryption algorithm and implement the *CBC mode* of AES. We have conducted a number of experiments to evaluate CloudaSec in the system and cloud levels. We study the client efficiency of the cryptographic algorithms with different pairing types and the user management costs for communication and storage.

In order to include the security procedures at Swift client side, we first implement CloudaSec key encryption procedures. Then, we append encryption functions to the upload and download swift commands, to support encrypted data key in user metadata. As an example, when a Swift client wants to store data for one recipient, he has to apply the following command:

swift* -A** *http://ip:port/auth/v1.0* ***-U *account:login* ***-K*** *password* ***upload*** *container object* ***-I*** *CloudaSec_Encrypt.*

The ***upload*** command attribute leads to the execution of *CloudaSec_Encrypt* procedure. For instance, the *CloudaSec_Encrypt* includes *EncryptData* and $EncryptKey_{OneToOne}$ algorithms which are processed at the client side, before outsourcing the data file (*object*) to the remote server. We must note that *object* is hashed using *SHA-256* and the result is stored in the file *object.AESkey*. Then, the data file, denoted by *object*, is encrypted using the symmetric algorithm *AES-256-CBC* and the key *object.AESkey*.

4.5.2 Computation Cost Evaluation

In order to evaluate the performances at the client side, we conduct data encryption and decryption tests locally. For our tests, we used 1000 samples in order to get our average durations. In addition, we conducted our experiments on an Intel core 2 duo, started on *single mode*, where each core relies on 800 MHz clock frequency (CPU).
Figure 4.3 shows the computation overhead of data encryption and decryption at the client side, with different sizes of data contents. We can notice that the data encryption takes less than 12 ms, in order to encrypt $1MB$ data file. We note that this computation cost

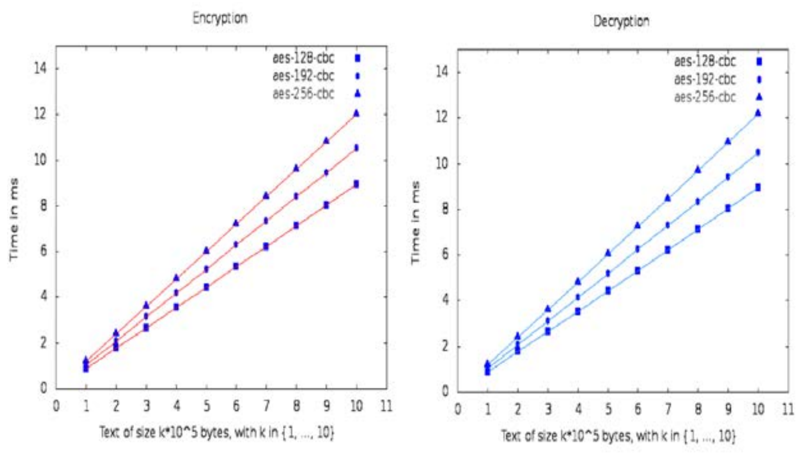

FIGURE 4.3 - Computation overhead of data encryption and decryption at the client side with different data size (from 10^5 to 10^6 bytes) (ms)

remains attractive, as it provides better security to outsourced data and does not deserve the client resources.
Then, we perform the encryption of the deciphering data key k. That is, as our proposed framework relies on the use of bilinear maps, we choose two symmetric pairing functions from the PBC library [Ben07], including type E pairing *e.param* and type A *a.param*. Thus, we examine the impact of different bilinear functions on the performances of CloudaSec, while considering three different security levels (cf. Figure 4.4).
In cryptography, the security level of a symmetric encryption algorithm is defined as the number of operations needed to break the algorithm when a k-bit key length is used. The security level in our proposal depends on the security level of the bilinear function $\hat{e}$ in use, which is related to the hardness of solving the ECDLP in $\mathbb{G}_1$. As such, it is closely related to the groups being selected. As shown in Figure 4.4, the encryption time increases along with the security level, while there is a tiny difference between the two symmetric pairing functions. As such, the type of the pairing function should be taken into account, while implementing CloudaSec data layer procedures. We must note that the type of the pairing function is bound to the choice of the elliptic curve, where the bilinear map is computed.
Finally, we investigate the impact of the cryptographic operations, at CloudaSec client side. We compare the encryption duration against OpenStack upload and download duration, as depicted in Figure 4.5. In fact, we examine the encryption operations compared to the Swift upload procedure and the decryption operations compared to the Swift download procedure. We must note that the computation times include the key generation and the data encryption with *AES-256-CBC* mode. We notice that the cryptographic operations, at the client side are acceptable compared to the upload operations and do not carry exhaustive computation capacities. For example, a $8 * 10^5$ bytes data size requires only 0.1 seconds to be enciphered, compared to 10 second be uploaded. Therefore, the encryption procedures involve 1% from the OpenStack upload overhead. As such, CloudaSec does not deserve the client resources, and presents an interesting processing cost for resource constrained devices.

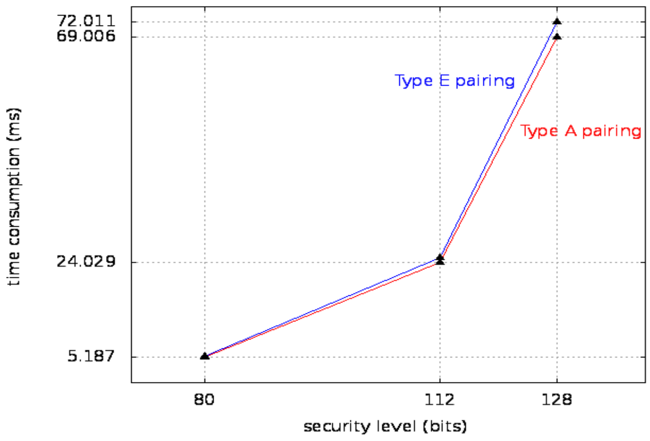

FIGURE 4.4 - Computation duration of Type A vs Type E pairing functions (ms)

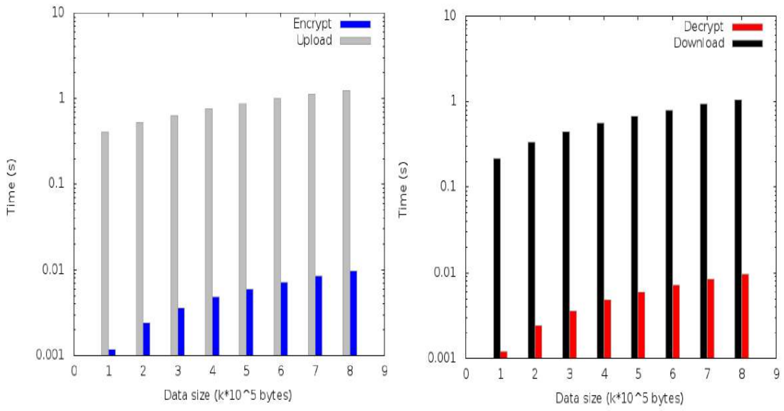

FIGURE 4.5 - Impact of cryptographic operations on CloudaSec at the client side ($log_{10}(ms)$)

4.5.3 Communication Cost Evaluation

We investigate the communication overhead, when a client outsources a data file to remote cloud servers and then when he retrieves the outsourced content. As such, we conduct some experiments with different data sizes and we evaluate the upload and download times of encrypted contents, as shown in Figure 4.6.

Our approach consists at first, to generate a random data file of a fixed size. Then, we encrypt the data file based on the AES-CBC algorithm. The length of the used enciphering key is 256 bits (AES-256-CBC). Afterwards, we upload the encrypted file and we download it from the cloud.

As such, we conducted some tests by choosing different files of different sizes. Each time, we compute the average time for uploading and downloading the encrypted file using the *AES-256-CBC* algorithm. Next, we present the results of average time computation to

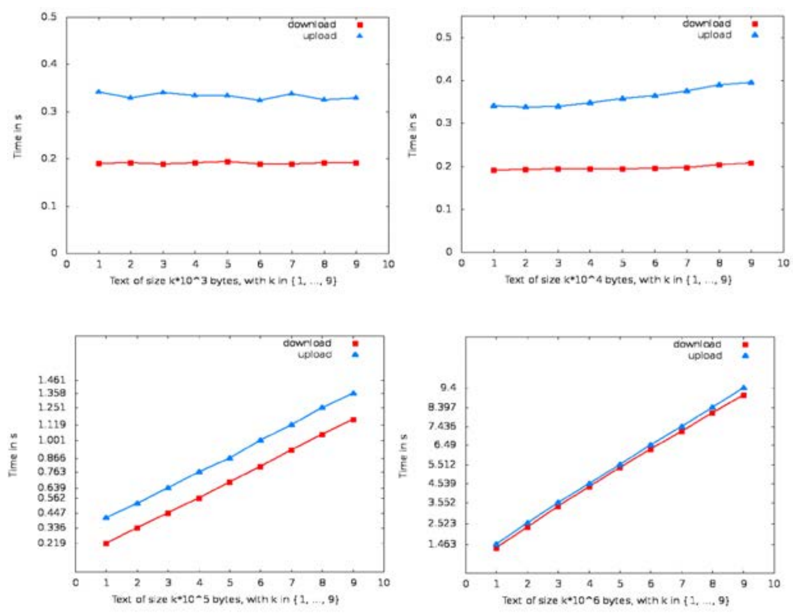

FIGURE 4.6 - OpenStack upload and download overhead with different data size (ms)

upload and download data files in cloud servers.

TABLE 4.3 - Average time to upload/download an encrypted data file of size varying from 10 to 10^4 bytes

	Average Time (in s)		Standard Deviation σ	
data size (in Bytes)	upload	download	upload	download
10	0.338	0.193	0.231	0.067
10^2	0.329	0.192	0.210	0.060
10^3	0.339	0.189	0.233	0.027
10^4	0.326	0.194	0.191	0.080

From Table 4.3, we notice that the average communication times are merely stable, with small data sizes, for the storage and backup scenarios. However, this overhead gradually increases, when the client intends to store large data contents. In addition, we deduce, from Figure 4.6, that the time needed to upload an encrypted data file in Swift cloud servers is greater than the time consumed to download the file from these remote servers. Besides, for data size less than 5×10^4 bits, the time needed to upload the file in remote servers (respectively download encrypted contents from the cloud servers) is almost constant and does not depend on data sizes. However, for a given data size greater than 5×10^4 bits, the time needed to upload/download the encrypted data file in the cloud increases proportionally with the data size.

TABLE 4.4 - Average time to upload/download an encrypted data file of size varying from 10^3 to 9×10^3 bytes

	Average Time (in s)		Standard Deviation σ	
data size (in Bytes)	upload	download	upload	download
1×10^3	0.340	0.190	0.230	0.021
2×10^3	0.328	0.190	0.202	0.035
3×10^3	0.340	0.189	0.260	0.018
4×10^3	0.333	0.191	0.235	0.059
5×10^3	0.333	0.193	0.260	0.127
6×10^3	0.324	0.188	0.200	0.020
7×10^3	0.337	0.188	0.241	0.026
8×10^3	0.325	0.191	0.176	0.026
9×10^3	0.328	0.192	0.194	0.025

TABLE 4.5 - Average time to upload/download an encrypted data file of size varying from 10^6 to 9×10^7 bytes

	Average Time (in s)		Standard Deviation σ	
data size (in Bytes)	upload	download	upload	download
1×10^6	1.464	1.272	0.226	0.086
2×10^6	2.524	2.340	0.248	0.080
3×10^6	3.552	3.376	0.246	0.120
4×10^6	4.540	4.383	0.260	0.150
5×10^6	5.512	5.339	0.260	0.125
6×10^6	6.490	6.272	0.292	0.124
7×10^6	7.437	7.196	0.358	0.130
8×10^6	8.398	8.121	0.308	0.159
9×10^6	9.400	9.034	0.328	0.153
10^7	10.32	10.58	0.291	0.253

We also analyze the communication cost, due to a group update, namely when a new user wants to join the group. In our tests, we are based on pre-computed tables, in order to optimize the computation cost to resource-constrained devices. Thus, we consider that the group includes 10 members at the beginning. Then, 10 users join the group simultaneously, until reaching 100 members. We recall that the computation complexity of the group update increases with respect to the number of new subscribers, as presented in Section 4.3.2.
As depicted in Figure 6.4, the derivation of the new secret group key takes less than 6 ms for 100 users. This computation cost remains interestingly attractive, along with our broadcasting approach.

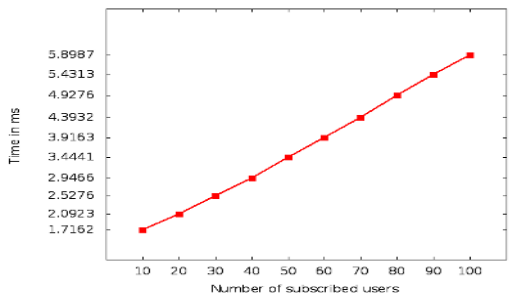

FIGURE 4.7 - Computation complexity of a group update (ms)

4.5.4 Storage Cost Evaluation

We investigate the storage cost for the key management operations at the client side. In order to maintain a group membership, a registered user has only to keep the secret group key. Let suppose that the security parameter $\xi = 80$ bits. We denoted by $E(\mathbb{F}_n)$ the elliptic curve defined over the finite prime field $\mathbb{F}_n$. Meanwhile, we denote $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \longrightarrow \mathbb{G}_2$ the bilinear function. $\mathbb{G}_1$ corresponds to the q-torsion subgroups of $E(\mathbb{F}_n)$ and $E(\mathbb{F}_{n^k})$ where k is the embedding degree of the curve E. $\mathbb{G}_2$ is a multiplicative subgroup of $\mathbb{F}_{n^k}$ of order q.
For example, according to the security parameter $\xi = 80$, we set $q = 160$ bits and $n = 512$ bits length, while the embedding degree is equal to 2. As such, $\mathbb{G}_2$ is a subgroup of $\mathbb{F}_{n^2}$ which has a order 1024 bits order. Therefore, a client has to locally keep a secret d, where $|d| = 160$. As such, CloudaSec introduces an attractive storage cost, especially for limited storage capacities.

4.6 Synthesis

To end this first part, we have established a comparison between our contributions, and several commonly used encryption mechanisms in a cloud environment, based on the security requirements, presented in section 4.2.

From Table 4.6, we notice that our first contribution, relying on the usage of ID-based cryptography does not depend on a trusted entity, unlike all the other mechanisms, including CloudaSec. However, [KBL13] introduces a heavy data enciphering cost, due to the usage of an asymmetric encryption mechanism, comparing with CloudaSec [KLEB14]. Furthermore, CloudaSec presents a low data encryption overhead for a large amount of data due to symmetric encryption scheme usage.
In addition, CloudaSec enables partial update of the group secret key. That is, user revocation and subscription procedures are achieved without updating the private keys of the remaining group members. CloudaSec also guarantees the deduplication that many schemes do not ensure, such as IBC.

TABLE 4.6 - Comparison between Our Contributions and Different Cryptographic mechanisms in Clouds

Scheme	ABC	IBC	Proxy re-encryption	Hybrid Encryption	ID-based proposal [KBL13]	CloudaSec [KLEB14]
Data Encryption Cost	Heavy	Heavy	Heavy	Light	Heavy	Light
Dynamic Group Management	Part	Whole	Whole	Whole	Whole	Part
Scalability	Yes	No	No	No	No	Yes
Trusted Entity	Yes	Yes	Yes	Yes	No	Yes
Forward Secrecy	–	No	Yes	–		
Backward Secrecy	–	No	No	–	Extra Computation	Yes
De-duplication	No	No	No	Yes { Convergent Encryption }	No	Yes

4.7 Conclusion

In this chapter, we presented our second contribution [KLEB14], called CloudaSec. CloudaSec is a secure data sharing scheme for dynamic groups in untrusted cloud storage environments. Our approach ensures the confidentiality of outsourced data in public untrusted cloud servers, while introducing two encryption levels: symmetric data encryption level and asymmetric key encryption level.
Contrary to several proposed schemes, CloudaSec defines an efficient revocation mechanism, with no need to update the secret keys of the remaining group members. That is, flexible access control policies are enforced among users belonging to separate groups with different privileges. Thus, our design conveys performance advantages for large scale sharing groups.

CloudaSec is proved secure under a curious provider security model and a malicious adversary, such as a revoked user. That is, we considered different threat use cases, and we detailed the resistance of CloudaSec to malicious access and confidentiality disclosure

attempts, while providing the correctness' proofs of the proposed algorithms and procedures.
Under a simulated CloudaSec framework based on Openstack Swift service, experimental results showed the efficiency of CloudaSec in scalable data sharing, while considering the impact of the cryptographic operations at the client side.

Part II

Cloud Data Storage Integrity

Table of Contents

In the second part of this dissertation, we investigate a second cloud data storage issue, namely the Proof of Data Possession (PDP) concern. It relieves the description of algorithms allowing a data owner to check data integrity of outsourced files on remote servers. This challenge is more complex, while considering that clients have no technical means of verifying that their files are not vulnerable, for instance, to drive-crashes. Therefore, cloud customers should have an efficient way to perform periodical remote integrity verifications, without keeping the data locally, following three substantial aspects: *security level, public verifiability,* and *performance.* This concern is magnified by clients' constrained storage and computation capabilities and the large size of outsourced data.

In chapter 5, we introduce PDP approaches and we investigate security requirements for designing remote data checking schemes in cloud storage environments, while presenting a short review of literature.

In order to fulfill the data integrity verification requirement, we define, in chapter 6, a new zero-knowledge PDP protocol that provides deterministic integrity verification guarantees, relying on the uniqueness of the Euclidean Division. These guarantees are considered as interesting, compared to several proposed schemes, presenting probabilistic approaches.

In chapter 7, we propose SHoPS, a Set-Homomorphic Proof of Data Possession scheme, supporting the 3 levels of data verification. SHoPS allows an implementation of remote data checking at the three networking interfaces, namely, the client-CSP interface, the CSP-storing nodes interface and between two CSPs interface. This ensures the flexibility of SHoPS application and enables fulfilling each verifier request. Indeed, we present the set homomorphism property, which extends malleability to set operations properties, such as union, intersection and inclusion. For instance, SHoPS saves energy within the CSP by distributing the computation over the multiple nodes. Each node provides proofs of local data block sets. This is to make applicable, a resulting proof over sets of data blocks, satisfying several needs, such as, proofs aggregation.

Chapter 5

Remote Data Checking in Clouds

The price of light is less than the cost of darkness

Arthur C. Nielson - 1897–1980

Contents

5.1 Introduction

Consequences of security breaches in cloud storage could be seriously damaging to both service providers and cloud clients. Without trust from users, service providers could lose their customers. On the other hand, users whose valuable data lost, or sensitive information hacked could experience irrecoverable damage.

In fact, loss of data control has a significant influence on designing the cloud business model. First, the US Patriot Act [Sta02] gives the government unprecedented access to outsourced data which are either physically hosted in the USA territory or generated by

an American actor (i.e; an American enterprise or an enterprise having economic stakes in USA). This US regulation law threatens the European legislation which has strict privacy protective laws. As such, it is important for a cloud user to have the possibility to verify his outsourced data are still hosted in a specific geographic perimeter, supporting the requested legislation.
Second, one of the biggest concerns with cloud data storage is data integrity verification on untrusted servers. In fact, cloud providers generally claim storing data files with redundancy to protect against data loss. Additionally, they often disperse these data across multiple storage placements. Such distribution provides resilience against hardware. Nonetheless, in order to reduce operating costs and save storage capacities, dishonest providers might intentionally neglect these replication procedures, resulting in unrecoverable data errors or even data loss. More seriously, many byzantine attacks may give rise to data leakage attacks. Hence, a cloud customer should have an efficient way to perform periodical remote integrity verifications, without keeping the data locally. This customer's concern is magnified by his constrained storage and computation capabilities and the large size of outsourced data.

There are three basic requirements for data integrity verification process, namely, efficiency, unbounded use, and self-protected mechanism. Efficiency implies minimal network bandwidth and client storage capacity which are needed for the verification procedure. The client does not need to access the entire data for verification purpose. Unbounded use represents verification process should support unlimited number of queries. Self-protect mechanism means the process itself should be secure against malicious server that passes the integrity test without accessing the data. A number of different techniques and methods have been proposed and designed for cloud data integrity verification process. These mechanisms are referred to as *Provable Data Possession* (PDP) and *Proof of Retrievability* (PoR), which originally emerged with a similar concept but different approaches. The difference between PDP and PoR schemes is that PoR checks the possession and integrity of data and it can recover data in case of a failure, while PDP only verifies the possession of outsourced data by the remote servers. Usually, a PDP can be transformed to a PoR by adding erasure or error correcting codes. Since then, each mechanism had gone through further development along different directions such as dynamic data support, public verifiability, and privacy against verifiers. Dynamic data support allows clients to dynamically update their data partially after uploading to cloud servers. Public verifiability enables an authorized entity, not only data owners, to perform the verification process. Privacy against verifiers ensures that the verification algorithm does not contain any private information of the data owner.

In this chapter, we first introduce PDP and PoR approaches, in Section 5.2. Then, in Section 5.3, we investigate security requirements for designing remote data checking schemes in cloud storage environments and we present a short review of literature explaining improvements of PDP and PoR schemes, with regard to public verifiability and efficiency. Finally, we lead a discussion while comparing performances of several newly emerging schemes, in Section 5.4.

5.2 PDP and PoR Review

In this section, we first introduce the naive approach to perform a remote data checking and we point out its main drawbacks. Then, we give an overview of several emerging PDP and PoR schemes.

5.2.1 Naive Approach

The Proof of Data Possession (PDP) is a challenge response protocol enabling a client to check whether a file data D stored on a remote cloud server is available in its original form. A PDP scheme consists of four procedures: pre-process, challenge, proof, verification (cf. Fig 5.1). For building meta-data of a file, the client runs the *pre-processing* procedure. In most of the cases, the client keeps the meta-data secret and sends a version of the data file to the cloud server (e.g., encrypted data, error coding, embedded watermark). To check the possession of the data file, the client sends a randomized challenge to the server for a proof of a specified file data. In response, the server generates the proof which requires the possession of the original data to compute the proof which depends on the received challenge to avoid the replay attacks. Once received, the client compares the proof with the locally stored meta-data.

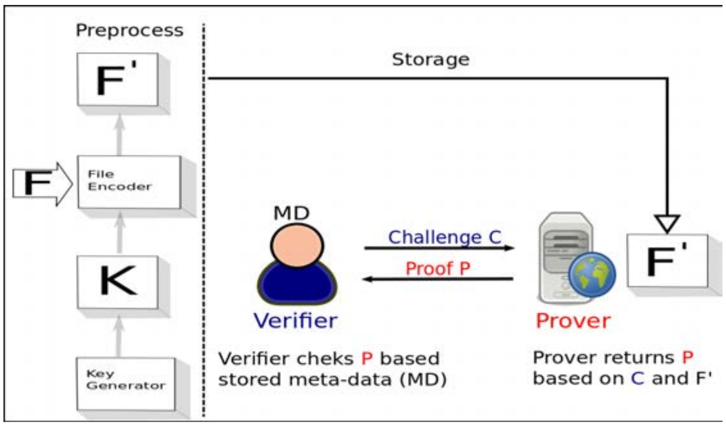

FIGURE 5.1 - Generic PDP scheme

The simplest solution to design a PDP scheme is based on a hash function $H()$. That is, the client pre-calculates k random challenges c_i, where $i \in \{1, k\}$ and computes the corresponding proofs as $p_i = H(c_i||D)$. During the challenging procedure, the client sends c_i to the server which computes $p'_i = H(c_i||D)$. If the comparison holds, the client assumes that the cloud provider preserves the correct data file. The biggest disadvantage of this scheme is the fixed number of challenges that were computed in the pre-processing procedure. That is, the client can request the server, for integrity checking, only k times.

5.2.2 Introduction to Remote Data Checking Schemes

The notion of PDP has first been introduced by Ateniese et al. in [ABC$^+$07]. That is, the client divides the file data D into blocks and creates a cryptographic tag for each block b_i, as $T_{i,b} = (H(W_i)g^{b_i})^d)modN$, where N is an RSA number, g is a public parameter, d is the secret key of the data owner and $H(W_i)$ is a random value. The proof generation is performed by aggregating several cryptographic tags, based on the requested data block indexes.
[ABC$^+$07] is efficient as there is no need to retrieve data blocks for the verification of data possession. The main drawbacks are computation complexity due large number of modular exponentiations in both setup phase and verification phase, and the private verifiability which requires the secret key of the data owner. In [ABC$^+$11], Ateniese et al. propose a publicly verifiable version, which allows any entity to challenge the cloud server. However, [ABC$^+$11] is insecure against replay attacks in dynamic scenarios because of the dependencies of index blocks in proof generation and the loss of homomorphism property in the verification procedure (cf; Section 2.3.3).

Juels et al. [JK07] introduce a method to detect unauthorized changes of stored data by randomly adding *sentinels* in the original data. Their scheme, called Proof of Retrievability (PoR), does not support public verifiability. In addition, only a fixed number of challenges is allowed.
Recently, Xu et al. [XC12] propose a new concept to prove the server data possession. That is, the client creates tags as polynomials and considers the file blocks as coefficients to polynomials. The proof procedure is based on polynomial commitment and uses evaluation in the exponential instead of bilinear maps. This idea has also been adopted by [KK12], based on Lagrangian interpolation.

5.3 Security Requirements

The design of a remote data checking scheme is motivated by providing support of both robustness and efficiency, while considering the limited storage and processing resources of user devices. It has to fulfill the following requirements:

- **Public verifiability:** the public data possession verification is an important requirement, permitting any authorized entity to verify the correctness of outsourced data. Thus, the data owner can be relieved from the burden of storage and computation.
- **Stateless verification:** proofs should be generated according to a randomly produced challenge. Thus, stateless verification requires the use of unpredictable values.
- **Low computation overhead:** on one hand, for scalability reasons, the amount of computation at the cloud storage server should be also minimized, as the server may be involved in concurrent interactions. On the other hand, the proposed algorithms should also have low processing complexity, at the client side.

- **Low communication overhead:** an efficient PDP should minimize the usage of bandwidth, relying on low communication cost.

- **Low storage cost:** the limited storage capacities of the user devices has a critical importance in designing our solution. So that, low storage cost at the client side is highly recommended.

- **Unlimited challenges:** the number of challenges should be unlimited. This condition is considered as important to the efficiency of a PDP scheme.

In the following subsections, we present the developments of PDP and PoR schemes, in order to improve public verifiability, efficiency, and dynamic data support, respectively.

5.3.1 Public Verifiability

Improving public verifiability has become a popular topic for researchers, since the introduction of the first original PoR scheme, proposed by Juels et al. On the basis of [JK07], Shacham et al. [SW08] propose two new PoR schemes. The first mechanism is privately verifiable and it relies on *pseudorandom functions* (PRFs). However, the number of authentication tokens stored on the server is proportional to the number of data blocks, and the proposed technique does not prevent from data blocks' leakage. The second scheme relies on bilinear signatures, proposed by Boneh et al. in [BLS01]. This second method ensures public data verification and the proofs are reduced to a single authentication value, thus reduced communication complexity from $\mathcal{O}(n)$ to $\mathcal{O}(1)$, where n is the number of data blocks. Unfortunately, this scheme still works on static data only, without support of dynamic data update.

In 2009, Wang et al. proposed a novel system model, which relies on a *Third Party Auditor* (TPA) [WWRL10]. Based on a privacy preserving third party auditing protocol, the TPA is considered as a trusted entity, which manages the stored data in cloud. That is, the data owner delegates periodical data integrity verifications to the TPA, which takes charge of monitoring exchanges between the client and the remote cloud server. In [WWRL10], TPA adopts a public key based homomorphic authenticator to perform public auditing without keeping a local copy of data for integrity checking. Homomorphic authenticators are used to verify meta-data generated from individual data blocks while the aggregated authenticators form a linear combination of data blocks.
Afterwards, Zhu et al. [ZWH^{+}11] propose a construction of a dynamic audit scheme for untrusted remote storage systems. Their scheme detects abnormal behavior of the prover by using fragment structure, random sampling, and index-hash table. Even though TPA based schemes allow public data integrity verification, they have a considerable drawback. That is, they require an additional component, which is a third party auditor, added to the existing cloud storage architecture. The implementation of such schemes might be a burden for service providers because of additional costs.

5.3.2 Efficiency

As discussed above, efficiency of remote data checking schemes consists on the optimization of computation complexity, communication overhead and storage cost.
Several research works are devoted to improve efficiency of PDP and PoR schemes. For instance, in 2008, Curtmola et al. integrate error-correcting codes to the PDP scheme, proposed by Ateniese et el. in [ABC+07], in order to secure multiple replicas over distributed system without encoding each separate replica [CKB08]. This technique considerably reduces the computation complexity. In addition, Dodis et al. [DVW09] improve the Shacham PoR scheme by reducing the challenge size to be linear with respect to the security parameter, from $\mathcal{O}(n^2)$ to $\mathcal{O}(n)$.

In [ADPMT08], Ateniese et al. propose an improved version of their original PDP scheme, referred to as scalable PDP scheme. [ADPMT08] adopts symmetric key encryption instead of public key encryption which reduces the computation overhead. It also supports updates on outsourced data. However, scalable PDP does not support the public verifiability requirement, due to the use of the symmetric key cryptography. Besides, all challenges and verifications have to be pre-computed, and the number of updates is limited and fixed a priori.
Afterwards, Bowers et al. introduce a distributed cryptographic system, to prove data retrievability [BJO09]. Their scheme, called HAIL (High Availability and Integrity Layer), differs from all prior works. In fact, HAIL considers a distributed setting in which a client must spread a file across multiple servers with redundancy and only stores a small constant state locally.

5.3.3 Dynamic Data Support

In this section, we give an overview of remote integrity verification schemes that support dynamic data updates. However, we have to note that we do not consider this design requirement, in the following chapters.

Supporting dynamic data updates in remote integrity verification schemes is a challenging concern. In 2008, Ateniese et al. [ADPMT08] introduced the first partially dynamic PDP scheme, where block insertion was not supported. In 2009, Erway et al. proposed a Dynamic Provable Data Possession mechanism (DPDP) [EKPT09]. Their scheme supports full dynamic operations (eg., append, insert, modify, delete), while relying on rank-based authenticated directories. Nevertheless, [EKPT09] maintains a list of tags and stores root metadata, at the client side to prevent replay attacks. As such, the computational complexity raises up to $\mathcal{O}(logn)$, which remains attractive, due to the support of dynamic updates. For instance, to generate a proof for 1 GB file, DPDP produces only 415 KB proof information, with 30 ms computational overhead.
Wang et al. [WWL+] propose a dynamic proof scheme, which relies on the use of homomorphic tokens with distributed verification of erasure-coded data. It provides block update, delete and append operations and does not support the insert function. However, [WWL+] involves a third party auditor to ensure public verifiability.

5.4 Summary

Finally, we state that PDP and PoR schemes are considered as promoting approaches which ensure remote data integrity checking in cloud storage environments. The first PDP and PoR design algorithms are a bit different, regarding several aspects. For instance, PoR schemes are considered to be more secure compared to PDP algorithms. That is, PoR mechanisms require the encryption of the original data and error-correcting codes have to be applied to recover damaged data. However, PDP schemes are known for higher efficiency and applicability to large-scale public databases, such as digital libraries.

Table 5.1 summarizes several remote data checking schemes. That is, we lead a comparison between these emerging approaches, while enumerating the advantages and drawbacks of each mechanism. In addition, we focus on the cryptographic primitives involved in the generation and verification of data proofs.

TABLE 5.1 - Approaches of Data Integrity Checking in the Cloud Storage Environments

Scheme	Advantages	Drawbacks	Primitives
PDP [ABC+07]	- Support of both encrypted and non-enciphered data files - Only a small part of data is needed to generate the proof	- Static data only - Probabilistic approach	- Homomorphic hashing: to compose multiple block inputs into a single value to reduce the size of proofs.
PoR [JK07]	- Ability to recover file with error correcting code.	- Static data only. - File needs to be encrypted before uploading to the server. - Needs additional space to hide *sentinels* in encoded data blocks.	- Error correcting code: to recover a partially corrupted file.
Scalable PDP [ADPMT08]	- No additional encryption is required - Allow outsourcing dynamic data in some degree. - Rely on symmetric key which is more efficient than public key encryption.	- Does not offer public verifiability - All challenges and answers are pre computed. - Number of updates is limited and fixed before.	- Symmetric key cryptography. - Message Authentication Code (MAC)
HAIL [BJO09]	- Ability to check integrity in distributed storage via data redundancy. - Proof is compact in size and is independent of data size.	- Static data only	- Pseudo-random functions - Message Authentication code (MAC) - Universal hash functions.

Recently, in [BVDJ+11], Bowers et al. explore new economic security models for cloud services. They provide a different formulation of the threats that cloud users face. That is, RAFT proposes an approach confirming data redundancy on storage systems, based on a time measure function. The main disadvantage of this scheme is the communication cost which depends on the number of blocks in the challenging request, and the storage cost prohibitively important. In fact, the authors exposed two verification approaches. First, they propose a private verification algorithm to check the exactitude of server's responses based on a local copy stored by the data owner. While this option may efficiently work for some scenarios, it is too much restrictive in many other cases as it undermines much of the benefits of cloud outsourcing. Second, in order to improve storage capacity consumption, they refer to the Merkle Tree signature. Thus, this technique also requires the use of a secret for each outsourced data file.
Considering other challenging concerns to provide remote proof verifications, [VDJO+12] aims to prove correct data encryption at rest by imposing a time basis protocol. An issue arising in the design of this hourglass protocol is the fact that the client needs an authentic version of the outsourced data file, to verify responses from the server. However, the client's storage needs should be of constant size, otherwise the benefits of data outsourcing decrease. In order to optimize storage cost at the client side, [VDJO+12] proposes to use additional MACs or Merkle Tree processes at the client side. This necessitates that the client retrieves the integrity checks during the challenge response protocol which raises the bandwidth consumption. In addition, the verifier must keep a secret for each outsourced data (if MACs are used) or the root of the hash tree.
On the basis of [WS12], Williams and Sion propose SR-ORAM scheme. It allows a client hiding its data access pattern from an untrusted cloud server in a single round protocol. However, this scheme requires a poly-logarithmic storage cost and does not support public sharing verification.

In Table 5.2, we summarize the above reviewed PoR and PDP schemes by a thorough comparison of their performances, based on security requirements presented in Section 5.3. In fact, we review the capability to support an unlimited number of challenges, denoted by the *Nb. of chal.* metric. In addition, we examine the public verifiability design requirement and the retrievability feature, denoted by *Public Verif.* and *Integrity*, respectively. We must note that, by retrievability feature, we mean the support of remote data integrity checking. Besides, we take into consideration the robustness and efficiency of reviewed algorithms, while analyzing the computation and communication complexity at both the client and the server side. We also investigate the deployment of a Third Party Auditor, seeking for a fully delegation of integrity checking operations. Finally, we have to note that schemes marked by an asterisk (*) support either partially or fully dynamic data updates.

It is noteworthy that schemes ensuring dynamic data support suffer from higher complexities compared to their counterparts. Future research directions include improvements on efficiency and fully dynamic data support. To improve efficiency of those schemes, reducing communication cost and storage overhead are rightful considerations. However, fully dynamic data support remains a challenging objective, because it increases complexity while reducing update information at the cloud server side.

TABLE 5.2 - Performances Comparison for Remote Data Verification Schemes in Cloud Data Storage Environments (n is the number of data blocks)

Scheme	*Nb. of Chal.*	*Public Verif.*	*Integrity*	*CSP comp.*	*User comp.*	*Comm. comp.*	TPA
[ABC+07]	fixed	Yes	No	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	No
[JK07]	∞	No	Yes	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(n)$	No
[SW08]	∞	Yes	Yes	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	No
[WWRL10]*	∞	Yes	Yes	$\mathcal{O}(logn)$	$\mathcal{O}(logn)$	$\mathcal{O}(logn)$	Yes
[DVW09]	∞	No	Yes	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$	No
[ADPMT08]*	fixed	No	No	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	No
[CKB08]	∞	Yes	No	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	No
[EKPT09]*	∞	No	Yes	$\mathcal{O}(logn)$	$\mathcal{O}(logn)$	$\mathcal{O}(n)$	No

5.5 Conclusion

In cloud storage environments, it is important to allow users to efficiently and securely verify that cloud storage servers store their data correctly. To address this issue, a number of Proof of Retrievability (PoR) and Proof of Data Possession (PDP) schemes have been proposed wherein servers must prove to a verifier that data are stored correctly. In this chapter, we give an overview of remote data verification schemes, while presenting security requirements for the design of a PDP and a PoR algorithm.

While existing POR and PDP schemes offer decent solutions addressing various practical issues, they either have non-trivial (linear or quadratic) communication and computational complexity, or only support private verification. Newly emerging remote data verification schemes aim to provide both practical and unconditional integrity checking for remote computation. Towards this goal, Setty et al. [SBW11] propose to benefit from early research results such as interactive proof systems [Gol00]. These mechanisms are referred to as ideal methods that enable a cloud client to verify a proof's correctness in a constant time, while relying on a suitably encoded proof under a negligible chance of false positives.

Following this direction, we present, in next chapters, our contributions for securely checking outsourced data integrity. Both PDP propositions ensure public verifiability and stateless verifications, and have following identifiable features :

- lightweight and highly secure PDP scheme with centralized computation done at the CSP: our third contribution [KEML14] relies on zero knowledge proofs. It benefits from the lightweight computation cost of the Euclidean Division (ED) and the high security level of zero knowledge protocols, in order to provide **deterministic** proofs, with constant communication overhead. However, our zero-knowledge proposal requires the CSP to centralize the computation of the proof at the gateway central node after re-assembling the data fragments from the storing nodes.

- distributed proof computation by the CSP: our fourth contribution, referred to as SHoPS proposes a scalable and modular data integrity verification scheme. SHoPS is an interesting approach to save energy, since it proposes to distribute the prover function to many storing nodes, thus providing low computation complexity at the CSP.

Chapter 6

A Zero-Knowledge Scheme for proof of Data Possession in Cloud Storage Applications

A proof is whatever convinces me

Shimon Even,
answering a student's question in his graph-algorithm class - 1987

Contents

6.1 Introduction

Several Proof of Data Possession (PDP) schemes are proposed to ensure integrity verifications of stored data on untrusted remote servers, and are designed to guarantee several requirements, namely lightweight and robust verification, computation efficiency and constant communication cost, based on different security assumptions. As presented above, these PDP techniques are widely analyzed into two categories, according to the role of the verifier in the model: private verifiability, where only the data owner can verify the server's data possession, and public verifiability, where any authorized entity can perform the verification procedure. On one hand, the private verifiability achieves higher efficiency, as the private information needed for verification is kept with the data owner. On the other hand, the public verifiability, allowing any entity to challenge the server for the proof of data possession, ensures higher scalability. That is, clients are able to delegate the verification task to another party, without devoting of their computation resources.

Even though existing PDP schemes have addressed various security properties, we still need a careful consideration of potential attacks, namely data leakage attacks, that may cause potential risks for privacy preservation. To design an effective security model, it is important to analyze the PDP scheme under the framework of Zero-Knowledge Proof scheme (ZKP), while considering an Interactive Proof System (IPS) between the client and the cloud server in a requested geographic perimeter [Gol00].

As such, in the key role of public verifiability and the privacy preservation support, this chapter presents our third contribution [KEML14]. It addresses the issue of provable data possession in cloud storage environments, following three substantial requirements: *security level, public verifiability,* and *performance.* [KEML14] proposes an efficient verification framework based on a fundamental arithmetic Euclidean Division, adapted to limited storage capacities. The framework is demonstrated to be resistant against data privacy leakage within a ZKPS. Introduction of a probabilistic method helps to reduce its computation and communication overheads.

The remainder of this chapter is organized as follows. First, Section 6.2 describes Zero-Knowledge proofs, introducing the general concept of these schemes and highlighting the essential characteristics of an Interactive Proof System (IPS). Then, Section 6.3 provides an overview of our system and security models, while considering realistic use cases. Section 6.4 presents our contribution and Section 6.5 gives a security analysis. Finally, a performance evaluation of the proposed scheme is given before concluding in Section 6.7.

6.2 Zero-Knowledge Proofs

The fundamental notion of zero-knowledge was introduced by Goldwasser et al. in [GMR85]. They considered a setting where a powerful *prover* is proving a theorem to a probabilistic polynomial-time *verifier*. The main idea is that the *prover* wants to convince a *verifier* that a statement is true, without revealing any of his secret. So that, these two entities mutually engage in an interactive protocol, where the verifier does not learn anything from his interaction with the prover.

Definition 6.2.1 *Interactive Proof Systems(IPS) [Gol00] An interactive proof system for a set S is a two-party game between a computationally unrestricted* prover $\mathcal{P}$ *and a probabilistic polynomial-time* verifier $\mathcal{V}$*, such that on input x, which is available to both parties, it satisfies:*

- Completeness – *For every $x \in S$, the verifier always accepts after interacting with the prover, on common input x.*

$$\forall x \in S, Pr[(V \Leftrightarrow P)(x)accepts] = 1 \tag{6.1}$$

- Soundness – *For every $x \notin S$ and every potential prover $\mathcal{P}^*$, the verifier rejects with probability at least $\frac{1}{2}$ after interacting with $\mathcal{P}^*$, on common input x.*

$$\forall x \notin S, \forall \mathcal{P}^*, Pr[(V \Leftrightarrow \mathcal{P}^*)(x)accepts] \leq \frac{1}{2} \tag{6.2}$$

In other words, a proof is complete if a honest verifier is always convinced of a true statement, computed and sent by a honest prover. Whereas, a proof is considered to be sound if a cheating prover can convince an honest verifier that a false statement is true with a negligible probability.
A Zero-Knowledge Interactive Proof System (ZKIPS) is a special kind of interactive proof mechanisms. In fact, there is an additional condition, namely, when $x \in S$, the verifier does not learn anything other than being convinced that the x is indeed in S. In an IPS, the soundness condition protects the verifier from accepting an incorrect claim. In a ZKIPS, the new condition protects the prover from having to reveal any information (other than the correctness of the claim). When the prover follows the protocol for an input $x \in S$, the verifier will learn nothing beyond the fact that $x \in S$.

Figure 6.1 presents a famous example, that explains the zero knowledge protocol. It is the example of a mysterious cave, the cave of *Ali BABA*, which was introduced by Jean-Jacques Quisquater et al. in [QQQ$^+$89] and widely adopted by several research works, later.
After the entrance, the cave splits into two paths, that lead to different sides of a magic door. This secret door only opens by saying the correct password. Alice knows this password, and wants to convince Bob of this. However, she only wants to convince Bob, not anyone else, because she does not want to reveal to others that she can open the famous magic door. Alice and Bob proceed as follows. Alice enters the cave and walks through path A or B to the magic door, leaving her at the A-side or the B-side of the

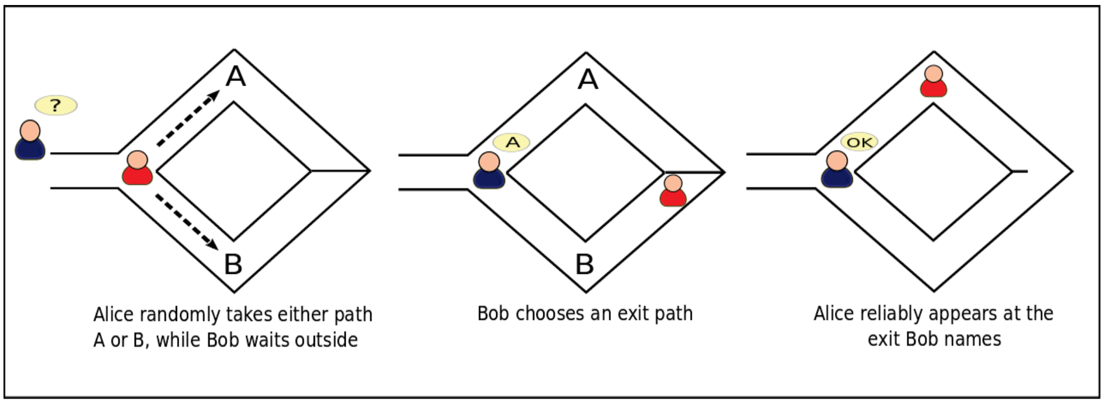
?
A
B
Alice randomly takes either path A or B, while Bob waits outside
A
A
B
Bob chooses an exit path
OK
Alice reliably appears at the exit Bob names

FIGURE 6.1 - Magical Cave [QQQ+89]

magic door respectively. After Alice entered the cave, Bob walks through the entrance to the intersections (not knowing which way Alice chose) and flips a coin. He will shout *exit A* in case of heads, and *exit B* in case of tails. Alice can hear this and will exit the cave through this path.
If she is on the A-side of the door and has to exit through A, she can simply walk out. If she is on the A-side but has to exit B, she must pass the secret door, using her password. In the same way, if Alice chose B and has to exit B, she just walks out, and if she chose B and must exit A, she will go through the door.

Bob will only see Alice exiting through the correct path, but he does not know whether Alice went through the magic door or not. However, when doing this repeatedly, Bob will be convinced that Alice is able to pass the secret door, because otherwise she would fail to exit through the correct path in half of the cases.

6.3 Model Description

Taking advantage from the properties of zero knowledge interactive proof systems, we present a novel PDP model based on the well-known GPS scheme proposed by Girault et al. in [GPS06]. GPS is a public-key-based zero knowledge protocol that was adapted to resource-constrained devices. Hence, we extend the GPS scheme to the verification of the authenticity of files stored on untrusted servers in cloud platforms. In this section, we describe the components of our PDP system and security models.

6.3.1 System Model

We consider three participating entities: the client (C), the user (U) and the cloud service provider (CSP). The client has a collection of data files stored on cloud servers after the pre-processing procedure. The user who shares the stored data with the client may challenge the cloud storage server to provide a proof of possession. The private verification of our proposal contains five probabilistic algorithms, defined as follows:

- *KeyGen* – given a selected security parameter λ, this algorithm outputs the data owner public and secret keys (pk, sk), where pk is a public elliptic curve point.
- *Setup* – given a data file $D \in \{0,1\}^*$ and the public key pk, the setup algorithm generates the data file identifier ID_F and the corresponding public elements (σ_1, σ_2).
- *GenChal* – this algorithm generates a randomized challenge c.
- *ChalProof* – given the challenge c, and the original version of the file data D, the *ChalProof* algorithm produces a proof $P = (y_1, y_2)$. Note that y_1 and y_2 are two elliptic curve points.
- *Verify* – given the proof P, the public elements and the private key of the data owner, *Verify* checks the data possession and outputs a result as either accept or reject.

Our idea makes use of Zero Knowledge Proofs (ZKP) to provide a data possession verification. That is, our approach is closely based on techniques related to the GPS scheme [GPS06], which is a public key verification protocol. Furthermore, we propose two variants of proof of possession, supporting public and private verifiability. The private verification makes use of a secret stored locally in the client's device, while the public proof check is based on pairing functions.

The choice for adopting the elliptic curve variant of GPS scheme is motivated by several reasons. First, ZKP joins randomness into exchanged messages. As such, for each verification session, the prover and the verifier generate new pseudo random values and new composition of the considered file data, thus making messages personalized for each session. Consequently, the randomness involved in the server's responses allows resistance to *data leakage attacks* and preservation of data privacy. Second, the GPS scheme is adapted to the required limited storage capacities on tags. From this perspective, with the prevalence of wireless communication, the mobile devices start sharing the benefits of on demand cloud storage services. Due to the resource-constrained devices, our scheme is based on only one secret which is needed for the verification of all outsourced data. In addition, the main advantage of our approach is the public verifiability, preserving the privacy of the outsourced data. That is, an authorized verifier makes only use of public elements and does not request the data owner for extra-computation procedures.

6.3.2 Security Model

For our technique to be efficient in cloud storage applications, we have to consider realistic threat models. We first point out the case where an untrusted cloud provider has a malicious behaviour. In such cases, the storage server claims that it possesses the data file, even if the file is totally or partially corrupted. To model this situation, our scheme is based on two important requirements proposed by Shacham [SW08]. On one hand, there exists no polynomial-time algorithm that can roof the verifier with non-negligible probability. On the other hand, there exists no polynomial-time algorithm that can recover the original data files by carrying out multiple challenge response exchanges.

Second, we consider the case of a malicious verifier that intends to get information about the outsourced data of the data owner. The fact that the verification process can be performed using public elements (thanks to the zero-knowledge property of our scheme) makes it possible for malicious clients to gain information about files stored on the untrusted remote servers.

The proposed protocol must provide the capabilities to the verifier and the service provider to thwart the two threats mentioned above. To this end, the PDP scheme must enforce a mutual verification of the actions conducted by the client and the storage server.

6.3.3 Assumptions

Our zero knowledge PDP proposal is based on the Elliptic Curve Cryptography (ECC) [HMV03]. To support the public verification, the client must first define a set of *public verification elements* (PVE). The client generates the groups $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ and the pairing function $\hat{e}$ from $\mathbb{G}_1 \times \mathbb{G}_2$ in $\mathbb{G}_T$. $\mathbb{G}_1$ and $\mathbb{G}_2$ are additive subgroups of the group of points of an Elliptic Curve (EC). However, $\mathbb{G}_T$ is a multiplicative subgroup of a finite field. $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ have the same order q. In addition, $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ are generated by P, Q and the generator $g = \hat{e}(P, Q)$, respectively. The bilinear function $\hat{e}$ is derived from the Weil or Tate pairing [BSSC05].

Moreover, we have to note that our PDP scheme makes use of two cryptographic assumptions, namely the *Elliptic Curve Discrete Logarithm Problem (ECDLP)* and the *Computational Diffie Hellman Problem (CDH)* (i.e; Section 2.4.1).

6.4 A New-Zero Knowledge PDP Protocol

In this section, we propose two new PDP schemes for cloud storage application. The first scheme applies when the verification is performed using public credentials while the second scheme restricts the verification process to the owner of the verified data. Both of our schemes rely on zero-knowledge challenge-response protocols. Therefore, they do not add any storage overhead at the client side, which is an important feature for applications where the access to mobile resource-impoverished devices is possible.
We provide mathematical proofs of the correctness (i.e., the PDP scheme returns a positive feedback if, and only if the file exists on the server and has not been altered) of the proposed schemes based on the properties of the Euclidean Division (ED) and the bilinear functions.

6.4.1 Private Data Possession Scheme

In our scheme, we define an elliptic curve EC over an additive subgroup $\mathbb{G}_1$ of a prime order q. Let P be a generator of $\mathbb{G}_1$.

When a client wants to store a file data D on the cloud, he first decomposes D into two blocks s and n. n represents the quotient and s is the remainder applying the Euclidean Division (ED) on the file D with the divisor b. Note that b is kept secret by the client and is used in the decomposition of several outsourced file data. That is, b represents the unique secret information that the client should preserve for all its requests for proof of data possession verification. We must note that b is tightly related to the security of our remote verification scheme. As such, the definition of several data divisors can extend our proposition. That is, the data owner may rely on different secrets with respect to the sensitiveness of the data that he intends to share on the cloud.

Then, with regards to the ECDLP, the published elements are bP, nP, sP, denoted by pk, σ_1, and σ_2, respectively. pk is referred to as the public key of the data owner, while σ_1 and σ_2 are the public elements of the file D.
In the following, we use R, B and K that satisfy the requirements fulfilled in [GPS06]. We also denote by $\cdot$ the scalar point multiplication in an additive group and by $\star$ two elements multiplication belonging to a multiplicative group.

Figure 6.2 shows the general concept of our private data possession scheme. This scheme consists in two phases. During the first phase, the *KeyGen* and *Setup* procedures are executed. This phase is performed only when the file is uploaded on cloud servers. The second phase occurs when the client wants to verify the authenticity of the outsourced data file. To this purpose, it generates a new challenge *chal* in order to obtain a proof of data possession from the cloud server. This latter runs the *ChalProof* algorithm which is a 3 way procedure. In the following, we provide a detailed description of the steps that are conducted in each of the two aforementioned phases.

Preprocessing: $D = nb + s$
Public Parameters: (EC, $+$) an elliptic curve
P a generator of EC
$\{pk, \sigma_1, \sigma_2\}$
R, B and K three integers such that $R \gg BK$
Secret key: $sk = b$ where $b \in [0, R[$

Client (C)		**Storage Server (CSP)**
choose $b' \in_R [0,\ R[$	$\xrightarrow{Request(b', ID_D)}$	calculate $D_1 = mb' + z$, choose $(r, t) \in_R [0,\ B[^2$
	$\xleftarrow{x_1, x_2}$	calculate $(r \cdot P, t \cdot P) = (x_1, x_2)$
generate $c \in_R [0,\ K[$	$\xrightarrow{c}$	calculate $((r + cz) \cdot P, (t + cm) \cdot P) = (\gamma_1 \cdot P, \gamma_2 \cdot P) = (y_1, y_2)$
	$\xleftarrow{y_1, y_2}$	

Check $y_1 - x_1 - c \cdot \sigma_1 = c.sk \cdot \sigma_2 - b' \cdot (y_2 - x_2)$

FIGURE 6.2 - General Framework of Private Data Possession Scheme

Phase I consists on the two following procedures:

- *KeyGen* – the client public and private key are generated by invoking the *KeyGen*(1^λ) procedure (cf. Algorithm 11).

- *Setup*– when the client wants to store a file data D in the cloud, he runs the *Setup* algorithm, in order to generate the corresponding public elements (σ_1, σ_2) of the data file D (cf. Algorithm 12).

Algorithm 11 KeyGen Procedure (Private Data Possession)

```
1: Input: System security parameter ($\lambda$)
2: Output: public key $pk$ and master secret key $sk$
3: Choose an elliptic curve $EC$ over an additive subgroup $\mathbb{G}_1$ of a prime order $n$, where
   $BitLength\ (n) > \xi$ and ECDLP is hard in $\mathbb{G}_1$;
4: Select $P$ a generator of $EC$;
5: Use a deterministic secure pseudo random number generator (SPRNG) with a random
   secret $seed$ to generate $b \in_R [0,\ R[$;
6: $sk \leftarrow b$;
7: $pk \leftarrow b \cdot P$;
8: return $(pk, sk)$
```

Algorithm 12 Setup Procedure (at the data owner side)

```
1: Input: File data ($D$), pair of secret and public keys $(pk, sk)$ and the point generator
   $P$
2: Output: File identifier $ID_D$ and the file public elements $(\sigma_1, \sigma_2)$
3: Generate the file identifier $ID_D$;
4: $ED(D, sk) = (s, n)$;      ED: Euclidean Division;
5: $\sigma_1 \leftarrow n \cdot P$;
6: $\sigma_2 \leftarrow s \cdot P$;
7: return $(ID_D, \sigma_1, \sigma_2)$
```

Phase II consists in a challenge-response protocol conducted between the verifier and the storage server. The underlying steps are detailed below.

6.4.1.1 GenChal

The *GenChal procedure* is executed by the client and yields a challenge for the cloud storage server. The client chooses at random a challenge divisor $b' \in_R [0,\ R[$. In response, the server has to provide a valid new file decomposition using the random divisor b' sent by the client. It is worth noticing that the client does not store any additional information for the proof verification. That is, the verification procedure makes only use of the secret key of the client sk.

6.4.1.2 ChalProof

The *ChalProof*, executed by the server, has to generate a valid proof of data possession of D. In our construction, the *ChalProof* is a 3 way procedure between the client and the server with common inputs (b', P). For the sake of consistency, we suppose that the server possesses a version of the file which is potentially altered. Hereafter, this version is denoted by D_1. The objective of the following steps is to verify whether $D_1 = D$ or not.

- Commitment $(CSP \to C)$: the storage server calculates the ED of the file using the challenging divisor b' sent by the data owner as:

$$D_1 = mb' + z$$

Then, he chooses at random two integers $(r, t) \in_R [0,\ B[^2$ and sends their commitments to the client as $(x_1, x_2) = (r \cdot P, t \cdot P)$.

- Challenge $(C \to CSP)$: the client chooses a random challenge $c \in_R [0,\ K[$ and sends it to the server storage, in order to provide the proof. Thereby, the client gets fresh instances indistinguishable from the past results.

- Response $(CSP \to C)$: the server calculates two integers γ_1 and γ_2 based on the generated randoms (r, t), and the computed remainder and quotient (m, z) as follows:

$$(\gamma_1, \gamma_2) = (r + cz, t + cm)$$

Then, the CSP performs two scalar point multiplications to generate the proof (y_1, y_2), where $y_1 = \gamma_1 \cdot P$ and $y_2 = \gamma_2 \cdot P$. Afterwards, the storage server sends (y_1, y_2) for integrity verification.

6.4.1.3 Verify

The client verifies the correctness of the server response. He checks the following equality, using on the secret sk, the divisor b', the challenge c, and the responses (x_1, x_2) and (y_1, y_2) got from the server.

$$y_1 - x_1 - c \cdot \sigma_1 = c.sk \cdot \sigma_2 - b' \cdot (y_2 - x_2). \tag{6.3}$$

If the equality holds, the verifier has a proof that the file D exists on the server and that it has not been altered.

Lemma 6.4.1 Private Verification Correctness *The verification procedure of Equation 6.3 holds if, and only if the file $D_1 = D$.*

Proof.
Having received (y_1, y_2), the client calculates $y_1 - x_1 = \gamma_1 \cdot P - x_1 = cz \cdot P$ and $b' \cdot (y_2 - x_2) = b' \cdot (\gamma_2 \cdot P - x_2) = cmb' \cdot P$. Given that $D = D_1 = nb + s = mb' + z$, we have $cnb - cmb' = cz - cs$. Taking into consideration that $sk = b$, this writes to the following.

$$c.sk \cdot \sigma_1 - b' \cdot (y_2 - x_2) = (y_1 - x_1) - c \cdot \sigma_2. \tag{6.4}$$

This proves the correctness of the verification step (i.e., $D = D_1$). The uniqueness of the quotient and remainder of the ED allows to state that Equation 6.3 is true if, and only if $D = D_1$. $\square$

6.4.2 Public Data Possession Scheme

An authorized user, different than the client that initially uploaded the file on the storage server, could also verify the authenticity of this file. However, the protocol proposed, in the foregoing subsection 6.4.1, cannot be used to this purpose since it supposes that the verifier has the private key of the client, which is not the case of the user. In the following, we demonstrate that the public parameters of the client can also be used in order to implement a PDP scheme between a user, different from the owner of the file, and the storage server. Thus, procedures presented in Section 6.4.1 cannot apply to the public data possession scenario, as the client uses his secret sk to verify the proof.

Preprocessing: $D = nb + s$
Public Parameters: (EC, $+$) an elliptic curve
P a generator of EC
$PVE = \{\mathbb{G}_1, \mathbb{G}_2, P, g, \hat{e}\}$
$\{b \cdot P, n \cdot P, s \cdot P\} = \{pk, \sigma_1, \sigma_2\}$
R, B and K three integers such that $R \gg B.K$
Secret key: $b \in [0, S[$

User (U)		**Storage Server (CSP)**
choose $b' \in_R [0, R[$	$\xrightarrow{Request(b', ID_D)}$	calculate $D_1 = mb' + z$, choose $(r, t) \in_R [0, B[^2$
	$\xleftarrow{x_1, x_2}$	calculate $(r \cdot P, t \cdot P) = (x_1, x_2)$
generate $c \in_R [0, K[$	$\xrightarrow{c}$	calculate $\gamma_1 = r + cz, \gamma_2 = t + cm$, compute $y = (\gamma_1 + b'\gamma_2) \cdot P$
	$\xleftarrow{y}$	

Check $\hat{e}(c \cdot \sigma_1, pk) \star \hat{e}(c \cdot \sigma_2, P) = \hat{e}(y, P) \star \hat{e}(x_1 + b'x_2, P)^{-1}$

FIGURE 6.3 - General Framework of Public Data Possession Scheme

As illustrated in Figure 6.3, the client publishes a set of public verification elements (PVE). As described in Section 6.3.3, these elements are returned by the *KeyGen* procedure as $PVE = \{\mathbb{G}_1, \mathbb{G}_2, P, g, \hat{e}\}$. Note that, for ease of exposition, we used a symmetric pairing function $\hat{e}$. In addition, the client maintains the same public elements, used for the private verification (pk, σ_1, σ_2). The verification condition to state that the file exists on the server and has not been altered is expressed in the following Equation:

$$\hat{e}(c \cdot \sigma_1, pk) \star \hat{e}(c \cdot \sigma_2, P) = \hat{e}(y, P) \star \hat{e}(x_1 + b'x_2, P)^{-1}. \qquad (6.5)$$

Lemma 6.4.2 Public Verification Correctness *The verification condition of Equation 6.5 holds if, and only if the file $D_1 = D$.*

Proof.
For checking the correctness of the received proof, the authorized user has to verify the

equality between the two Equation 6.5 sides. That is, he has to compare $\hat{e}(c \cdot \sigma_1, pk) \star \hat{e}(c \cdot \sigma_2, P)$ to $\hat{e}(y, P) \star \hat{e}(x_1 + b' x_2, P)^{-1}$.

On one hand, given the public elements PVE and the generated challenge c, the verifier first computes $\hat{e}(c \cdot \sigma_1, pk)$ and $\hat{e}(c \cdot \sigma_2, P)$ as follows:

$$\hat{e}(c \cdot \sigma_1, pk) = \hat{e}(cn \cdot P, b \cdot P) = \hat{e}(P, P)^{cnb} = g^{cnb} \tag{6.6}$$

$$\hat{e}(c \cdot \sigma_2, P) = \hat{e}(cs \cdot P, P) = \hat{e}(P, P)^{cs} = g^{cs} \tag{6.7}$$

Then, the authorized user multiplies the obtained results as follows:

$$\hat{e}(c \cdot \sigma_1, pk) \star \hat{e}(c \cdot \sigma_2, P) = g^{cnb} \star g^{cs} = g^{c(nb+s)} = g^{cD} \tag{6.8}$$

We must note that the second side of Equation 6.5 may be written as follows (cf. Equation 6.9).

$$\hat{e}(y, P) \star \hat{e}(x_1 + b' \cdot x_2, P)^{-1} = \hat{e}(y, P) \star \hat{e}(-(x_1 + b' \cdot x_2), P) \tag{6.9}$$

On the other hand, the verifier relies on the received responses (x_1, x_2) and y, to compute $\hat{e}(y, P)$ and $\hat{e}(x_1 + b' \cdot x_2, P)$. That is, he first calculates $x_1 + b' \cdot x_2$, while performing one scalar point multiplication and one point addition. Afterwards, he computes the pairing functions to get the following two group elements:

$$\hat{e}(y, P) = \hat{e}((\gamma_1 + b'\gamma_2) \cdot P, P) = \hat{e}([(r + cz) + b'(t + cm)] \cdot P, P) = \hat{e}([r + b't + cD_1] \cdot P, P) = g^{r + b't + cD_1} \tag{6.10}$$

$$\hat{e}(-[x_1 + b' \cdot x_2], P) = \hat{e}(-r \cdot P - b' \cdot (t \cdot P), P) = \hat{e}(-(r + b't) \cdot P, P) = g^{-(r + b't)} \tag{6.11}$$

The authorized user computes the second side of Equation 6.5, as follows:

$$\hat{e}(y, P) \star \hat{e}(-[x_1 + b' \cdot x_2], P) = g^{r + b't + cD_1} \star g^{-(r + b't)} = g^{r + b't + cD_1 - r - b't} = g^{cD_1} \tag{6.12}$$

The condition of Equation 6.5 is equivalent to the equality between Equation6.8 and Equation 6.12. Given the uniqueness of the quotient and remainder of ED and the non-singularity of the pairing function, this condition holds if, and only if $D = D_1$. □

6.5 Security Analysis

In this security analysis, the cloud service provider is not considered to perform preservation of computation resources by reusing the same pair $(x_1, x_2) = (r \cdot P, t \cdot P)$ from one possession proof session to another. The server is assumed to renew the pair of random numbers r and t and to calculate the elliptic points $x_1 = r \cdot P$ and $x_2 = t \cdot P$ for each data integrity checking session.

In Section 6.5.1, we describe the security of our PDP protocol using a game that captures the data possession property. In fact, this game consists in a fraudulent storage server, as an adversary, that attempts to construct a valid proof without possessing the original data file. As the proof generation depends on the variant of the remote data checking protocol, we denote by p the proof computed and sent by the cloud storage server to the verifier, in the last message. That is, p corresponds to (y_1, y_2), for a private data verification, and to y, for a public verification.

When the verifier wants to check the server's possession of data file, he sends a random query b_g to the adversary.

- *ForgeCommit* – without the possession of the data file, the server tries to generate two randoms m^* and z^*, using an iterated hash function. Then, he chooses two integers $(r, t) \in_R [0,\ B[^2$ and sends their commitments to the client as $(x_1, x_2) = (r \cdot P, t \cdot P)$.
- *Challenge* – the verifier requests the adversary to provide a valid proof of the requested file, determined by a random challenge c_g.
- *ForgeProof* – the adversary computes a proof p^* using his random generation (m^*, z^*) and the challenge c_g.

The adversary wins the data possession game, if the *verify* procedure returns **accept**.

In Section 6.5.2, we discuss the resistance of our proposed scheme against a malicious verifier. The verifier attempts to get knowledge about the outsourced data, based on the available public elements PVE and multiple previous exchanges resulting in successful verification sessions with the legitimate storage server.

6.5.1 Security and Privacy Discussion

According to the standard definition of an interactive proof system proposed in [Gol00], our protocol has to guarantee three security requirements: completeness and soundness of verification, and the zero-knowledge property.

6.5.1.1 Soundness of Verification

Recall that the soundness means that it is infeasible to confound the verifier to accept false proofs p^*. That is, even if a collusion is attempted, the CSP cannot prove its possession.
The soundness of our proposition is relatively close to the *Data Possession Game*. Hence, the soundness meets the correctness of verification (Equation 6.3 and Equation 6.5), while considering the uniqueness of the quotient and remainders of the Euclidean Division (ED). This property prevents from forging the soundness of verification of our protocol.

For the nonexistence of a fraudulent server prover, we assume that there is a knowledge extractor algorithm Ψ [Gol00], which gets the public elements as input, and then attempts to break the Elliptic Curve Discrete Logarithm Problem (ECDLP) in $\mathbb{G}$. We state, in Section 2.4.1, that ECDLP holds in $\mathbb{G}$, if there does not exist a Probabilistic Polynomial

Time (PPT) algorithm, with non negligible probability ϵ, that may solve the ECDLP problem.

The Ψ algorithm interacts as follows:

Learning 1– the first learning only relies on the data owner public key $pk = b \cdot P$ as input. Ψ tries to get knowledge of the client secret key sk. That is, the extractor algorithm Ψ picks at random $r_i \in_R [0,\ R[$, where $i \in \mathbb{Z}_p$ and computes $r_i P$. For each r_i, Ψ checks whether the comparison holds between pk and $(r_i \cdot P)$. Based on our assumption, Ψ cannot extract the secret key of the client with noticeable probability.

Learning 2– the input of the second learning is the tuple $(pk, \sigma_1, \sigma_2, PVE)$. The algorithm attempts to extract the secret data divisor $sk = b$ by performing following steps:

1. $\hat{e}(pk, \sigma_1) = \hat{e}(b \cdot P, n \cdot P) = g^{nb}$
2. $\hat{e}(pk, P) = \hat{e}(b \cdot P, P) = g^{b}$
3. $\hat{e}(n \cdot P, P) = g^{n}$

This learning cannot hold, because of the DDH assumption. In [Bon98], Boneh demonstrates that the DDH assumption is much stronger than the CDH.

6.5.1.2 Completeness of Verification

In our scheme, the completeness property implies public verifiability property, which allows any entity, not just the client (data owner), to challenge the cloud server for data possession or data integrity without the need for any secret information. That is, public verification elements, needed in the verification process are publicly known. Thereby, any authorized user may challenge the server storage and efficiently verifies the proof of data possession. Hence, our proposal is a public verifiable protocol.

Lemma 6.5.1 Completeness of verification *Given the tuple of public elements* $(pk, \sigma_1, \sigma_2, PVE)$ *and* $D = D_1$*, the completeness of verification condition implies that Equation 6.5 holds in* $\mathbb{G}_2$*.*

Proof.
Based on Equation6.9 and Equation 6.12, the completeness of our protocol is performed as follows:

$$\begin{aligned}
\hat{e}(y, P) \star \hat{e}(-(x_1 + b' \cdot x_2), P) &= g^{r+b't+cD_1} \star g^{-(r+b't)} \\
&= g^{r+b't+cD_1-r-b't} \\
&= g^{cD_1} \\
&= g^{cD} \\
&= g^{c(nb+s)} \\
&= \hat{e}(P, P)^{cnb} \star \hat{e}(P, P)^{cs} \\
&= \hat{e}(cn \cdot P, b \cdot P) \star \hat{e}(cs \cdot P, P) \\
&= \hat{e}(c \cdot \sigma_1, pk) \star \hat{e}(c \cdot \sigma_2, P)
\end{aligned}$$

There exists a trivial solution when $g = 1$. In this case, the above verification could not determine whether the processed file is available, because the equality remains true. Hence, the completeness of our construction holds, if and only if when $g \neq 1$, giving the uniqueness of the quotient and the non singularity property of the pairing function $\hat{e}$.
$\square$

6.5.1.3 Zero Knowledge Property of Verification

Compared to the original zero-knowledge GPS scheme [GPS06], the prover represents the data storage server and the verifier represents any authorized user. As such, our scheme inherits the zero-knowledge feature from the GPS scheme.

The zero knowledge property ensures the efficiency of a cloud server against malicious attempts to gain knowledge from the outsourced data files. For our construction, this property is achieved thanks to personalized verification sessions. That is, randomness is required in cloud server's responses, in order to resist to *Data Leakage Attacks* (DLA) and to preserve the confidentiality of data (cf. Section 6.5.2.2).

6.5.2 Resistance to Attacks

In the following analysis, we discuss the resistance of our proposed scheme to classical attacks, when only considering the vulnerabilities over the data file. As such, we suppose a malicious verifier. This latter attempts to gain knowledge about the outsourced data, based on the public elements of the data owner and multiple interactions with the legitimate storage server.

6.5.2.1 Resistance to Replay Attacks

For each verification session, the server storage and the verifier generate new pseudo random values r and t. As presented in [GPS06], the probability of impersonation is $1/K^l$, where l is the number of the protocol rounds, and it depends on the challenge c. That is, a secure pseudo random generator can mitigate to replay attacks. In addition, the public proof of data possession variant of our protocol is secure against MIM attacks. [GPS06] demonstrates that an attacker cannot retrieve the secret key from the exchanged messages between the prover and the verifier.

6.5.2.2 Resistance to DLAs

We suppose that the goal of the fraudulent verifier is to obtain information about the outsourced data file. That is, the malicious verifier requests different decomposition of the same outsourced data file. As such, using two different sessions $((\alpha), (\beta))$, the attacker receives two different responses from the honest storage server, such that, $[y^{(\alpha)}, (x_1, x_2)^{(\alpha)}]$ and $[y^{(\beta)}, (x_1, x_2)^{(\beta)}]$.
From the session (α), the verifier receives $y^{(\alpha)} = ({\gamma_1}^{(\alpha)} + {b'}^{(\alpha)}{\gamma_2}^{(\alpha)}) \cdot P$. In the sequel, the

verifier calculates the following equation:

$$y^{(\alpha)} - {x_1}^{(\alpha)} - {b'}^{(\alpha)}{x_2}^{(\alpha)} = c^{(\alpha)} \cdot [(mb')^{(\alpha)} \cdot P + z^{(\alpha)} \cdot P] \tag{6.13}$$

Knowing the challenge c, the malicious verifier may compute $c^{-1}[y^{(\alpha)} - {x_1}^{(\alpha)} - {b'}^{(\alpha)}{x_2}^{(\alpha)}]$ to deduce $(mb')^{(\alpha)} \cdot P + z^{(\alpha)} \cdot P$. Similarly, the attacker deduces, from session (β), the combination $(mb')^{(\beta)} \cdot P + z^{(\beta)} \cdot P$.

The main idea of the Pollard's Rho attack is to find distinct pairs (c_1, d_1) and (c_2, d_2) of integers modulo p, such that $c_1 \cdot P + d_1 \cdot Q = c_2 \cdot P + d_2 \cdot Q$, where p is the prime order of P and $Q =< P >$. This algorithm looks for a collision that validates such equation. In our case, the verifier only deduces the results $(mb')^{(\alpha)} \cdot P + z^{(\alpha)} \cdot P$ and $(mb')^{(\beta)} \cdot P + z^{(\beta)} \cdot P$, with no idea about the internal decomposition of the data file. Therefore, he has to execute the Pollard's Rho algorithm that resolves the Elliptic Curve Discrete Logarithm Problem, under $\mathcal{O}(\sqrt{p})$ steps, while looking for a collision with the specific point $(mb')^{(\alpha)} \cdot P + z^{(\alpha)} \cdot P$. This condition raises the complexity of the attack scenario and reduces the probability of finding the file decomposition. Besides, the output of the Pollard's Rho algorithm is a modulo p result. As such, it is likely impossible to deduce the correct decomposition of the outsourced data file.

Knowing the challenge b', the attacker cannot reconstruct the file data, based on the ECDLP assumption. In fact, the prover sends only the tuple (x_1, x_2, y) to the verifier. As such, it is likely impossible to extract the data file D from the server response. Thus, the randomness property is also necessary for the non triviality of the proof system.

6.6 Performance Evaluation

In this section, we present a performance evaluation, based on the public verification variant of our construction, in terms of bandwidth, computation and storage costs. In addition, we conduct a number of experiments to evaluate our system performances. As such, we demonstrate that the adopted proof mechanism brings acceptable computation costs.

6.6.1 Theoretical Performance Analysis

To evaluate the objectives given in Section 5.3, we compare, in Table 6.1, our proposed protocol with some existing techniques. On the basis of the requirements of a data possession proof system, we choose four different PDP schemes ([ABC+07, DVW09, SW08, EKPT09]), that are most closely-related to our context.

Table 6.1 shows that none of the presented schemes ([ABC+07,DVW09,SW08,EKPT09]) does cover the totality of the fixed requirements, in Section 5.3. In addition, we must notice that these schemes studied the security aspects in a theoretical framework and overlook the issues related to constrained resources user devices.

TABLE 6.1 - Complexity comparison between different PDP techniques (n is the number of data blocks)

Metrics	[ABC$^+$07]	[DVW09]	[SW08]	[EKPT09]	prop.
Nb. of chall.	fixed	∞	∞	∞	∞
Public verif	Yes	No	Yes	No	Yes
CSP cmp. cost	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(logn)$	$\mathcal{O}(logn)$
User cmp. cost	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(logn)$	$\mathcal{O}(logn)$
Band. cost	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$
Storage cost	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$

6.6.1.1 Computation Cost Evaluation

As presented in Section 6.3, our scheme is composed of 5 algorithms: *KeyGen*, *Setup*, *GenChal*, *Chal Proof* and *Verify*. Among these algorithms, *KeyGen* and *Setup* are performed by the data owner. To generate his public key, the client performs one scalar point multiplication in an additive group $\mathbb{G}$. In the *Setup* procedure, this latter implements two scalar point multiplication $(n \cdot P, s \cdot P)$ and an Euclidean Division (ED) of the file data, which remains linearly dependent on the data size. Note that, this *Setup* algorithm is one-time cost for the data owner and can be performed apart the other procedures.
For each proof generation, the server applies the ED of the file data, and performs two scalar multiplication $(\gamma_1 \cdot P, \gamma_2 \cdot P)$. Upon receiving the server proof, the verifier conducts 4 pairing computations.

Table 6.1 states the computation cost comparison between our scheme and previous works, at both client and server side.
On the server side, our construction introduces one point addition and two scalar point multiplications, regardless the number of data blocks. Therefore, contrary to the other approaches, our scheme achieves a $\mathcal{O}(logn)$ server computation complexity.
On the verifier side, we brought additional computation cost, in order to perform a public verifiability. That is, the public verification procedure can also be performed by authorized challengers without the participation of the data owner. As such, this concern can be handled in practical scenarios, compared to private verification schemes ([DVW09, EKPT09]) which have to centralize all verification tasks to the data owner. In our scheme, the authorized verifier has to generate two random scalars $b' \in [0, R[$ and $c \in [0, K[$, in order to conduct his challenge request. Then, he checks the received proof from the cloud server, while performing four pairing computations and two elements multiplications in a multiplicative group. Thus, the public verifiability introduces a $\mathcal{O}(logn)$ processing cost at the verifier side.

6.6.1.2 Bandwidth Cost Evaluation

In our proposed scheme, the bandwidth cost comes from the generated challenge message *GenChal* algorithm and the proof response in each verification request. We neglect the computation cost of algebraic operations because they run fast enough [BGHS07], compared with operations in elliptic curve groups and multiplicative groups.

On one hand, the exchanging challenge algorithm consists in transmitting one random element c, where $c \in_R [0, \ K[$ and two elliptic curves points (x_1, x_2). For a recommended security, we consider a security parameter $\lambda = 80$ bits, thus, the total cost of the challenge message is the double size of a group element of an additive group $\mathbb{G}$.

On the other hand, the proof response consists only in one elliptic curve element $y = \gamma_1 + b'\gamma_2 \cdot P$. Therefore, the total bandwidth cost becomes constant and the bandwidth complexity of our scheme is $\mathcal{O}(1)$.

As shown in Table 6.1, [DVW09] and [SW08] present $\mathcal{O}(n)$ bandwidth complexity, where n is the number of encoded data blocks. As a consequence, the bandwidth cost of these algorithms is linear to n. Considering the number of permitted challenges, [ABC$^+$07] suffers from the problem of pre-fixed number of challenges, which is considered as an important requirement to the design of our construction. Nevertheless, their scheme presents a constant bandwidth cost, just like our proposed protocol. Based on a private proof, [EKPT09] also performs a low bandwidth cost. However, this algorithm supports only private verification. Therefore, along with a public verification, our proposed scheme, allows each verifier to indefinitely challenge the server storage with a constant bandwidth cost.

6.6.1.3 Storage Cost Evaluation

At the client side, our scheme only requires the data owner to keep secret his private key sk, which is a random element $b \in_R [0, R[$, and to store three public elements. These public elements consist in three elliptic curve points $\{pk, \sigma_1, \sigma_2\}$. Thus, the storage size of each client is $3|P|$. We must note that $|P|$ is the size of a group element, which is dependent on the security parameter λ. This storage overhead remains acceptable and attractive for resource constrained devices mainly as it not dependent on the number of data blocks and the size of data contents.

6.6.2 Time Performance Discussion

In this section, we first present the context of the implementation of our proposed scheme. Then, we discuss the computation performances.

6.6.2.1 Context

In an effort to evaluate the performances of our proposal, we build a simulated proof of data possession based on Open Stack Storage system (Swift) [chac].

In order to discuss the communication cost and the computation complexity at the client side, we implement several cryptographic operations at the client side of our simulated cloud environment. First, our scheme essentially relies on the Euclidean Division (ED) of the data file and the multiplication of the remainder and the quotient by an elliptic curve

point. As such, the effort to evaluate the performance of our solution leads us to study the time performance of scalar and point multiplication operation, at the swift client machine and the computation cost of different symmetric pairing functions. Our tests are conducted in order to understand the execution cost of our proposal on real hardware. That is, on one side, we evaluated the scalar point multiplication durations, of different scalar size of random data for each multiplication. On the other side, we studied the processing cost, due to the computation of bilinear functions, relying on different security levels.
Second, we extend the functions of swift, in order to support our security requirements. That is, we set the security parameter at $\lambda = 80$, and we discuss the communication cost of our proposal, for different content size. For our tests, we use the GNU Multiple arithmetic Precision (GMP) library [ea02]. We used 1000 samples in order to get our average durations of scalar point multiplication operations, at the swift client machine.

6.6.2.2 Implementation Results

Four different scalar sizes, the scalar and point multiplication elementary operation is evaluated, in order to present the execution cost of this elementary operation on real hardware. The obtained results are summarized in Table 6.2.

TABLE 6.2 - Mathematical operations cost (in ms).

Sec. level (λ) / Scalar size (bits)	80	112	128
10	0.105	4.980	7.123
100	1.813	12.516	28.475
1000	14.522	41.009	79.868
10000	98.001	257.9	677.012

Table 6.2 shows that the computation time increases with the scalar size. We must note that the selected scalar presents either the remainder or the quotient, while applying the ED on the outsourced file data.
We also notice that the processing time with large integers is still reasonable. In order to increase the computation performances when larger scalars are needed, our scheme can take advantage of pre-computation tables, by expressing this scalar as a linear decomposition of precomputed scalars used in this table. Therefore, the execution cost of the scalar multiplication becomes much more easier. In addition, Table 6.2 shows that the consumed time for multiplication increases, independently from the choice of the scalar size, when we increase the level of security.

In order to show the performances of the public *verify* procedure, we examine the computation duration cost of pairing functions, at the swift client side. That is, for our comparison, we give references to performance analysis presented in chapter 4, section 4.5.2. In fact, we have used two symmetric pairing functions from the PBC library [Ben07], including type E pairing *e.param* and type A *a.param*, to examine the impact of different bilinear functions, based on three different security levels. We noticed that the type of pairing function should be taken into account, while implementing the proposed procedures.

We also investigate the communication overhead of the *GenChal* and the *ChalProof* procedures. For a security parameter λ set to 80, we measure the consumed bandwidth for varying data content size from 1000 bits to 8000 bits (cf. Figure 6.4).
Figure 6.4 shows that our proposal performs an acceptable bandwidth communication,

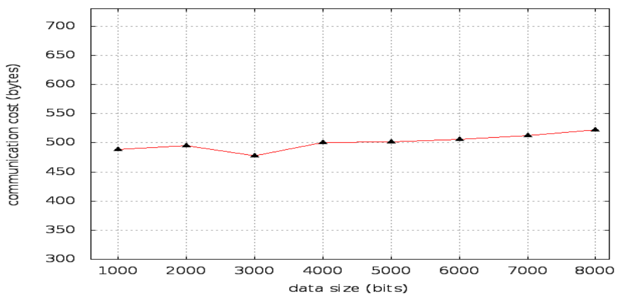

Figure 6.4 - Communication cost (bytes)

which remains constant at about 500 bytes for different data sizes.

6.7 Conclusion

In this chapter, we present our third contribution [KEML14], proposing a new zero-knowledge PDP protocol that provides deterministic integrity verification guarantees, relying on the uniqueness of the Euclidean Division. These guarantees are considered as interesting, compared to several proposed schemes, presenting probabilistic approaches.
Our proposal benefits from the elliptic curve variant of the GPS scheme advantages, namely high security level and low processing complexity. Hence, we extend the GPS scheme to the verification of the authenticity of files stored on untrusted servers. That is, the proposed solution can be performed on any connected terminal. As shown above, resource constrained devices like smartphones take reasonable time for executing the private variant of our scheme, for an acceptable security level. Moreover, for the public variant, we are optimistic that progress done in pairing functions, like Beuchat et al. running in less than 1 ms, will lead to better performances at the client side. Anyway, there is still the alternative that the client delegates to a powerful external entity the task of verifying the integrity of outsourced data.

Based on a data possession game, our scheme is shown to resist to data leakage attacks, while considering either a fraudulent prover or a cheating verifier.
Additionally, our proposal is deliberately designed to support public verifiability and constant communication and storage cost. Thus, we implemented a proof of concept based on the Openstack swift service to demonstrate the feasibility of our proposal and give support to our previous theoretical performance measurements.

Finally, we should state that zero-knowledge PDP schemes present a valuable way to reveal the abstract security assurances devoted to cloud based outsourcing issues.

Chapter 7

SHoPS: Set Homomorphic Proof of Data Possession Scheme in Cloud Storage Applications

The unseen enemy is always the most fearsome

George R.R. Martin - 1979

Contents

7.1 Introduction

THE CSA reports that data breaches and data losses are top critical threats to cloud security [httb]. That is, due to the abstract nature of cloud infrastructures, cloud users have no mean to verify how things are maintained. The CSA introduced the Cloud Transparency (CT) concept, where cloud users ask for and receive information about the elements of transparency as applied to cloud service providers. Assured of such evidence, cloud users become interested to outsource more sensitive and valuable business functions to the cloud, and reap even larger payoffs. For instance, cloud users are provided a way to find out important pieces of information concerning the compliance, security, privacy, integrity, and operational security history of service elements being performed in the cloud.

To this perspective, and with massive cloud storage failures that have come to light, it is important to consider remote testing of data correctness as an essential complement to contractual assurance and service-level specifications.

This concern is particularly important with rarely accessed data. For example, Facebook stores more than 240 billion photos, with users uploading an additional 350 million new photos every single day. However, as reported by the US IDC, outsourced data are generally less accessed after 91 days. In addition, these data are also rarely modified after 4 months [hSIHA]. Several approaches may be adopted by providers, to reduce storage cost while preserving their reputation.

On one side, the *tierced storage* is a strategy that organizes stored data into categories based on their priority – typically hot, warm and cold storage – and then assigns the data to different types of storage capacities to reduce costs. Rarely-used data are generally shifted to cheaper hardware, a move that saves money, such as in Amazon's glacier cold storage [htta] and Facebook Prineville data center [hbndcfcs]. Additionally, as cold data are rarely accessed, many providers may intentionally neglect replication procedures in order to save storage cost. As such, it would be important to conceive robust algorithms to effectively verify data integrity.

On the other side, to provide a low cost, scalable, location-independent platform for managing clients' data, current cloud storage systems adopt several new distributed file systems, such as Apache Hadoop Distribution File System (HDFS), Google File System (GFS), Amazon S3 File System, CloudStore, etc. These file systems share some similar features: a single metadata server provides centralized management by a global namespace; files are split into blocks or chunks and stored on block servers; and the systems are comprised of interconnected clusters of block servers. Those features enable cloud service providers to store and process large amounts of data. However, it is crucial to offer an efficient verification on the integrity and availability of stored data for detecting faults and automatic recovery. Moreover, this verification is necessary to provide reliability by automatically maintaining multiple copies of data and automatically redeploying processing logic in the event of failures. Although existing schemes ensure the correctness of outsourced data without downloading these data from untrusted stores, several algorithms are inefficient in a distributed cloud storage environment. This is due to the lack of homomorphism properties of interactive proofs where clients need to know the exact position of each file block in a multi-cloud environment. Consequently, the verification process leads to high communication overheads and computation costs at client sides as well.

Therefore, it is of utmost necessary to design an efficient PDP model to reduce the storage and network overheads and enhance the transparency of verification activities in scalable cloud storage systems. There is an implementation of remote data checking at the three following levels:

- *Between a client and a CSP* – a cloud customer should have an efficient way to perform periodical remote integrity verifications, without keeping the data locally. Additionally, the client should also detect SLA violation, with respect to the storage policy. This customer's concern is magnified by his constrained storage and computation capabilities and the large size of outsourced data.

- *Within a CSP* – for the CSP to check the integrity of data blocks stored across multiple storage nodes, in order to mitigate byzantine failures and drive-crashes.

- *Between two CSPs* – in the case of the cloud of clouds scenarios, where data are divided on different cloud infrastructures. Therefore, a CSP, through its cloud gate, should periodically verify the authenticity of data blocks hosted by another cloud platform.

For instance, stored across multiple storage nodes, the proof of data possession becomes more absorbing. In fact, taking advantage of the computation and storage capabilities of the storage nodes, each node has to provide proofs of local data block sets. Then, the cloud gate is responsible for performing operations on received proofs, while preserving the authenticity of the resulting proof. These operations include the detection of hardware failures based on redundant data, the extraction of proof of subset of data blocks and the aggregation of proofs. These operations can effectively save energy at the verifier side. That is, the verifier executes one verification for a single complex operation, involving several data blocks.

Nevertheless, most of the proposed schemes aggregate messages through the application of addition, multiplication, or polynomial functions on the original data blocks. They do not apply to the cases where only subsets of the original messages have to be tested during the aggregation process in order to minimize redundancy or to preserve anonymity. This flexible property is important, as it is required for these remote checking schemes, in order to perform efficient aggregation operations [JMXSW02].

This chapter introduces our fourth contribution. That is, we present SHoPS, a novel Set-Homomorphic Proof of data possession Scheme, supporting the 3 levels of data verification. SHoPS enables a verifier not only to obtain a proof of possession from the remote server, but also to verify that a given data file is distributed across multiple storage devices to achieve a certain desired level of fault tolerance. Indeed, we introduce the set homomorphism property, which extends malleability to set operations properties, such as union, intersection and inclusion.

Our set-homomorphic proof scheme allows verifying sets in a way that any authorized verifier can check the union of two proof sets, while considering the whole data file, or the intersection between two data blocks, while checking integrity proofs over different versions of logging files, as conversations on social networks.

Based on the *Pairing* based cryptography, SHoPS supports the application of basic set operations, on received proof sets without affecting the verification procedure. In addition, in the key role of public verifiability and the privacy preservation support, our proposed scheme addresses the issue of provable data possession in cloud storage environments, following three substantial aspects: *security level, public verifiability,* and *performance.*

The remainder of this chapter is organized as follows. First, Section 7.2 highlights security challenges of PDP schemes. Then, Section 7.3 gives a SHoPS overview and provides the security assumptions needed to ensure a secure, scalable and dynamic PDP scheme with public verifiability. Section 7.4 presents our contribution and Section 7.5 gives a security analysis. Finally, experimental and theoretical performance evaluations of the proposed scheme are given in Section 7.6 and Section 7.7 respectively, before concluding in Section 7.8.

7.2 Requirement Analysis

As presented above, the simplest solution to design a PDP scheme is based on a hash function $H()$. That is, the client pre-calculates k random challenges $c_i, i \in \{1, k\}$ and computes the corresponding proofs, $p_i = H(c_i||D)$. This solution is concretely impractical because the client can verify the authenticity of the files on the server only k times. Additionally, stored across multiple storage nodes, each node has to compute the related possession proof, based on a received challenge. As such, the aggregation process results in the removal of some redundant proofs transmitted from different nodes, in order to minimize the communication latency. Generally, the processing overhead at the client side is also reduced, but the new proof is longer than the original generated proofs. To illustrate the need for set-homomorphic proof schemes, we consider two storage nodes s_1 and s_2, possessing each one data block $\{B_i\}_{i\in\{1,2\}}$. The data blocks, considered as subblock sets, are represented as follows: $B_i = \{\pi_{1,1}, \cdots \pi_{i,q}\}$, where $i \in \{1, 2\}$, and q is the number of data subblocks. s_1 and s_2 generate respectively data possession proofs p_1 and p_2. Having removed the redundancy from these sets, the gateway is supposed to send $p_1 \cup p_2$. To guarantee the authenticity of the resulting proof while avoiding the shortcuts of the classical forwarding, we propose a new aggregate proof scheme, using set-homomorphic properties.

The design of our protocol is motivated by providing support of both robustness and efficiency. SHoPS has to fulfill the requirements, introduced in Section 5.3.

7.3 Model Description

This section introduces SHoPS, a set-homomorphic proof of data possession scheme, and it highlights the cryptographic assumptions that should be fulfilled by our proposed protocol.

7.3.1 SHoPS Overview

SHoPS introduces three participating entities: the client, the authorized user and the cloud service provider. The client has a collection of data files stored on cloud servers after the pre-processing procedure. The user, sharing the stored data file with the client, may request the cloud storage server to provide a proof of possession.
To tolerate drive failures, each data file is stored with redundancy, based on n-block erasure coding algorithm. It is, then, placed across l disks. Hence, each outsourced data file is divided into *blocks*, and each block B into q *subblocks*, where q is a system parameter. Each subblock is represented by a single element of the multiplicative group $\mathbb{G}_2$. Our single data block proof scheme is made up of five randomized algorithms, on the basis of two phases. During the first phase, the system initialization procedures are executed. This phase is performed once when the file is uploaded on the cloud.

- `gen` : $\{1\}^\lambda \rightarrow {\mathcal{K}_{pub}}^2 \times \mathcal{K}_{pr} \times {\mathbb{G}_2}^{2q-1}$ – given a selected security parameter λ, this algorithm outputs the data owner public and secret keys $(pk, \hat{pk}, sk)$, and a set of public credentials, with respect to the Diffie-Hellman Exponent assumption.
- `stp` : $2^{\mathcal{M}} \times {\mathbb{G}_2}^q \rightarrow \mathbb{G}_2$ – given a data block $B_i \in \{0,1\}^*$ and the public key pk, the setup algorithm generates the data block identifier ID_{B_i} and the corresponding accumulator $\{B_i, \varpi_i\}$, where $i \in \{1, \cdots, n\}$, and n is the number of blocks of a given data file.

The second phase occurs when the client wants to verify the authenticity of the file.

- `clg` : ${\mathbb{Z}_p}^* \times {\mathbb{Z}_p}^* \rightarrow \mathcal{C}$ – this stateless and probabilistic algorithm is computed by the client and takes as input the number of data blocks q. It generates a challenge $c \in \mathcal{C}$ consisting on a random block index and the public key element $\hat{pk}$ hidden with a random nonce η as $c = (i, \hat{pk}^\eta)$. The aim of `clg` is to verify the correctness of outsourced data.
- `prf` : $\mathcal{K}_{pub} \times 2^{\mathcal{M}} \times \mathcal{C} \rightarrow \mathcal{P}$ – the `prf` algorithm computes the server's response $P = (\sigma_1, \sigma_2)$ to a challenge, using the encoded file blocks stored on the server disks.

  In the following, we denote the algorithm that calculates the second element of the proof σ_2, by $\texttt{prf}_2$. That is, we have $P = \texttt{prf}(pk, B, c) = \{\sigma_1, \texttt{prf}_2(B, c)\}$, where B is the related data block and c represents the challenge.
- `vrf` : $\mathcal{P} \times {\mathcal{K}_{pub}}^2 \rightarrow \{0,1\}$ – a verification function for the cloud server's response P, where 1 denotes *accept*, i.e., the client has successfully verified correct storage by the server. Conversely, 0 denotes *reject*.

A slight difference between SHoPS and traditional proof schemes is that the generation of the possession proof operates on sets of data blocks in $2^{\mathcal{M}}$, instead of operating in data blocks in $\mathcal{M}$. Our choice is mainly motivated by proofs' authenticity, malleability concerns and energy efficiency while applying proof aggregation, as described in Section 7.2.

For instance, taking advantage of the storage and processing capabilities of the storing nodes, SHoPS saves energy within the CSP by distributing the computation over the multiple nodes. In fact, each node provides proofs of local data block sets. This is to make applicable, a resulting proof over sets of data blocks, satisfying several needs, such as proofs aggregation.

Supporting the public verifiability, SHoPS allows an implementation of remote data checking at the three networking interfaces, namely, the client-CSP interface, the CSP-storing nodes interface and between two CSPs interface. This ensures the flexibility of SHoPS application and enables fulfilling each verifier request. This verifier can be:

- a data owner, or an authorized verifier, challenging his service provider for a data possession proof. The proof aggregation, presented by the union of several data blocks proofs, is an interesting feature, as it allows a unique verification per file and ensures energy efficiency at the client side.
- a CSP gate challenging the storing nodes. As the operations on proof' sets are not limited to the aggregation of proofs, the intersection is important to detect byzantine failures at the storing nodes.
- a CSP challenging another CSP, in the case of interleaved clouds. The CSP may ask the hosting cloud to provide an aggregated proof or the intersection between two history log files of complex trade systems. In addition, the subset operator may interest the CSP verifier to check the correctness of replicated data hosted on remote servers. For example, suppose that a CSP outsourced a directory A composed of two data files F_1 and F_2, while preserving a copy of F_2 in its local storing nodes. This CSP may check the correctness of $A \setminus F_2$.

In the following, we refer to the proof aggregation, every set-operation over multiple proofs, namely, the union, the intersection and the inclusion operator. Therefore, we propose to develop an implementation of aggregate proofs using set-homomorphic properties.

Definition 7.3.1 *Set-Homomorphic based Proof*
We consider a message space $\mathcal{M}$, a proof space $\mathcal{P}$, a private key space $\mathcal{K}_{pr}$ and a public key space $\mathcal{K}_{pub}$. A set homomorphic based proof scheme is defined as follows. There exist two operations such as: $\odot : \mathcal{P} \times \mathcal{P} \to \mathcal{P}$ and $\circledcirc : \mathcal{K}_{pub} \times \mathcal{K}_{pub} \to \mathcal{K}_{pub}$, that satisfy the homomorphism and the correctness properties, for a set operation $\bullet$ for any messages B_i and B_j in $2^{\mathcal{M}}$.

- Homomorphism

$$\mathtt{prf}_2(B_i \bullet B_j, c) = \mathtt{prf}_2(B_i, c) \odot \mathtt{prf}_2(B_j, c) \tag{7.1}$$

- Correctness

$$\begin{aligned}&\texttt{vrf}(\texttt{prf}(B_i \bullet B_j, c), pk, \hat{pk}) = \\ &\texttt{vrf}(\texttt{prf}(B_i, c), pk, \hat{pk}) \wedge \texttt{vrf}(\texttt{prf}(B_j, c), pk, \hat{pk})\end{aligned} \tag{7.2}$$

We define SHoPS $= \{\texttt{gen}, \texttt{stp}, \texttt{clg}, \texttt{prf}, \texttt{vrf}, \texttt{agg}\}$, where the algorithm $\texttt{agg} : \mathcal{P} \times \mathcal{P} \to \mathcal{P}$ returns an aggregate proof. it is represented as follows:

$$\texttt{agg}(\texttt{prf}_2(B_i, c); \texttt{prf}_2(B_j, c)) = \texttt{prf}_2(B_i, c) \odot \texttt{prf}_2(B_j, c) \tag{7.3}$$

7.3.2 Complexity Assumptions

In this subsection, we present the complexity assumptions for the design of SHoPS, namely the Computational Diffie Hellman Assumption (CDH), introduced in Section 2.4.1, and the q-Diffie Hellman Exponent Problem (q-DHE), as follows:

Definition 7.3.2 *q-Diffie Hellman Exponent Problem (q-DHE)*
Let $\mathbb{G}$ be a group of a prime order p, and g is a generator of $\mathbb{G}$. The q-DHE problem is, given a tuple of elements $(g, g_1, \cdots, g_q, g_{q+2}, \cdots, g_{2q})$, such that $g_i = g^{\alpha^i}$, where $i \in \{1, \cdots, q, q+2, \cdots, 2q\}$ and $\alpha \xleftarrow{R} \mathbb{Z}_p$, there is no efficient probabilistic algorithm $\mathcal{A}_{qDHE}$ that can compute the missing group element $g_{q+1} = g^{\alpha^{q+1}}$.

Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two cyclic multiplicative groups of the same prime order p. An admissible symmetric pairing function $\hat{e}$ from $\mathbb{G}_1 \times \mathbb{G}_1$ in $\mathbb{G}_2$ has to be bilinear, non degenerate and efficiently computable (i.e; Section 2.4.1). In the following, we denote by $\star$ two elements multiplication belonging to a multiplicative group.

7.4 SHoPS: A New Set Homomorphic PDP Scheme

In this section, we propose SHoPS a novel set-homomorphic PDP scheme for cloud storage applications. SHoPS is based on techniques closely related to the well-known Pederson commitment scheme [Ped92]. That is, we extend the Pederson scheme to obtain a kind of a generalized commitment, in a subblock-index manner, providing fault-tolerance stateless verification. As such, for each verification session, the verifier generates a new pseudo random value and new index challenge position of the considered data file block, thus making messages personalized for each session. Relying on challenge-response protocols, we propose two verification processes. The first scheme restricts the verification to the data owner using only his private key. The second applies when the verification is performed using public credentials. This is inspired by the Boneh-Gentry-Waters (BGW) broadcast encryption system [BGW05]. The public key consists on a sequence of group elements $(g, g_1, \cdots, g_q, g_{q+2}, \cdots, g_{2q})$, where $g_i = g^{\alpha^i}$, defined upon the bilinear Diffie-Hellman exponent assumption. We provide mathematical proofs of the correctness (i.e., the PDP scheme returns a positive feedback if, and only if the file exists on the server and has not been altered) and the homomorphism properties of the proposed schemes.

7.4.1 Single Data Block SHoPS

The single data block proof is a PDP scheme restricted to a single block. The proofs correspond to all subblocks of a data block. This scheme consists in two phases. During the first phase, the `gen` and `stp` procedures are executed. Recall that this phase is performed only when the file is uploaded on the cloud. The second phase occurs when the client wants to verify the authenticity of the file. To this purpose, it generates a new challenge c in order to obtain a proof of data possession from the cloud server. This latter runs the challenge algorithm which is a three way procedure. In the following, we provide a detailed description of the steps, introduced in Section 7.3, that are conducted in each of the two aforementioned phases.

Algorithm 13 `gen` procedure

```
1: Input: System security parameter ($\xi$)
2: Output: Public keys ($pk, \hat{pk}$), master secret key $pr$ and public parameters $param = \{g_i\}_{1\leq i\leq 2q; i\neq q+1}$
3: Choose a multiplicative group $\mathbb{G}_1$ of a prime order $q$, where $BitLength\ (q) > \xi$ and DLP is hard in $\mathbb{G}_1$;
4: Select $g$ a generator of $\mathbb{G}_1$;
5: $\alpha \xleftarrow{R} \mathbb{Z}_p{}^*$;
6: $param = \{g\}$
7: for all $j \in [1 \dots 2q]$ do
8:     $param \leftarrow param \cup \{g^{\alpha^j}\}$;
9: end for
10: Use a deterministic secure pseudo random number generator (SPRNG) with a random secret $seed$ to generate $s \xleftarrow{R} \mathbb{Z}_p$;
11: $pr \leftarrow s$;
12: $pk \leftarrow g^s$;
13: $\hat{pk} \leftarrow g_{q+1}^s$;
14: return $(pk, \hat{pk}, pr, \{g_i\}_{1\leq i\leq 2q; i\neq q+1})$
```

The `stp` algorithm is presented by Algorithm 14. That is, each set of subblocks $\pi_{i,j}$ of B_i is presented by an accumulator $\varpi_i = \prod_{j=1}^{q} g_{q+1-j}^{\pi_{i,j}\ pr}$.

Algorithm 14 `stp` procedure

```
1: Input: File data block ($B_i$), the private key $pr$, and the public parameters $param$
2: Output: Block identifier $ID_{B_i}$ and the data block accumulator $\varpi$
3: Generate the data block identifier $ID_{B_i}$;
4: $\varpi_i = 1$;
5: for all $j \in [1 \dots q]$ do
6:     $\varpi_i \leftarrow \varpi_i * g_{q+1-j}^{\pi_{i,j}\ pr}$;
7: end for
8: return $(ID_{B_i}, \varpi_i)$
```

Phase II consists in a challenge-response protocol conducted between the verifier and the storage server. The underlying steps are detailed below.

7.4.1.1 `clg` procedure

The `clg` procedure is executed by the client and generates a challenge for the cloud storage server. The client chooses at random a subblock position $k \in \{1, q\}$ and a nonce η. The challenge $c \in \mathcal{C}$ consists of a random block index and the public key element $\hat{pk}$ hidden with a random nonce η as $c = (k, \hat{pk}^{\eta})$.

7.4.1.2 `prf` procedure

The `prf`, executed by the server, has to generate a valid proof of data possession of a given data block B_i. That is, in his response, the server has to provide a new valid accumulator using the random η sent by the client. In our construction, the `prf` is presented by Algorithm 15. For the sake of consistency, we suppose that the server possesses a version of the data block file which is potentially altered. Hereafter, this version is denoted by $\hat{B}_i$. The objective of the following steps is to verify whether $\hat{B}_i = B_i$.

Algorithm 15 `prf` procedure

```
1: Input: File data block ($B_i$), public keys of the data owner ($pk, \hat{pk}$), the public parameters $param$ and the challenge $c = (k, \hat{pk}^{\eta})$
2: Output: Proof $P = (\sigma_1, \sigma_2)$
3: $\sigma_1 \leftarrow (\hat{pk}^{\eta})^{\pi_{i,k}}$;
4: $\hat{\varpi}_i = 1$;
5: for all $j \in [1 \ldots q]$ do
6:    if $j \neq k$ then
7:       $\hat{\varpi}_i \leftarrow \hat{\varpi}_i * g_{q+1-j+k}^{\pi_{i,j}}$;
8:    end if
9: end for
10: $\sigma_2 \leftarrow \hat{\varpi}_i$;
11: return $(\sigma_1, \sigma_2)$
```

7.4.1.3 `vrf` procedure

In this section, we first present the public verification correctness. Then, we introduce the private verification process, which restricts the verification to the data owner

- Public Single Data Block Verification

An authorized verifier checks the correctness of the server response, based on public parameters. It is worth noticing that the client does not store any additional information for the proof verification. That is, the verification procedure makes only use of the public

$param$. The verifier checks the following equality, using the random secret η, the challenge c, and the response $P = (\sigma_1, \sigma_2)$ got from the server, as presented in Equation 7.4.

$$[\hat{e}(g_k, \varpi_i)\hat{e}(pk, \sigma_2)^{-1}]^{\eta}\hat{e}(g, \sigma_1)^{-1} = 1 \tag{7.4}$$

If the equality holds, the verifier has a proof that the data block file B_i exists on the server and that it has not been altered.

Lemma 7.4.1 Public Single Data Block Verification Correctness *The verification procedure of Equation 7.4 holds if, and only if the data block file $\hat{B}_i = B_i$.*

Proof.
Having received (σ_1, σ_2) from the cloud, the verifier first calculates $\hat{e}(pk, \sigma_2)$, using the public key of the data owner pk. Then, taking into consideration that $g_k = g^{\alpha^k}$, he computes $\hat{e}(g_k, \varpi_i)$.
Hereafter, based on the random nonce η, the verifier checks that $[\hat{e}(g_k, \varpi_i)\hat{e}(pk, \sigma_2)^{-1}]^{\eta}$ is equal to $\hat{e}(g, \sigma_1)$.

As such, this writes to the following steps.

$$[\hat{e}(g_k, \varpi_i)\hat{e}(pk, \sigma_2)^{-1}]^{\eta}$$

$$\begin{aligned}
&= [\hat{e}(g^{\alpha^k}, \textstyle\prod_{j=1}^{q} g_{q+1-j}^{\pi_{i,j}}{}^{s}) \star \hat{e}(g^s, \textstyle\prod_{j=1;j\neq k}^{q} g_{q+1-j+k}^{\pi_{i,j}})^{-1}]^{\eta} \\
&= [\hat{e}(g^{\alpha^k}, g^{\sum\limits_{j=1}^{q} \pi_{i,j}*\alpha^{q+1-j}{}^{s}}) \star \hat{e}(g^s, g^{\sum_{j=1;j\neq k}^{q} \pi_{i,j}*\alpha^{q+1-j+k}})^{-1}]^{\eta} \\
&= [\frac{\hat{e}(g^{\alpha^k}, g^{\sum\limits_{j=1}^{q} \pi_{i,j}*\alpha^{q+1-j}{}^{s}})}{\hat{e}(g^s, g^{\sum_{j=1;j\neq k}^{q} \pi_{i,j}*\alpha^{q+1-j+k}})}]^{\eta} \\
&= [\frac{\hat{e}(g, g^{s*\sum\limits_{j=1}^{q} \pi_{i,j}*\alpha^{q+1-j+k}})}{\hat{e}(g, g^{s*\sum_{j=1;j\neq k}^{q} \pi_{i,j}*\alpha^{q+1-j+k}})}]^{\eta} \\
&= [\frac{\hat{e}(g, g_{q+1}{}^{s*\pi_{i,k}} \star g^{s*\sum\limits_{j=1,j\neq k}^{q} \pi_{i,j}*\alpha^{q+1-j+k}})}{\hat{e}(g, g^{s*\sum_{j=1;j\neq k}^{q} \pi_{i,j}*\alpha^{q+1-j+k}})}]^{\eta} \\
&= [\frac{\hat{e}(g, g^{s*\sum\limits_{j=1,j\neq k}^{q} \pi_{i,j}*\alpha^{q+1-j+k}})\hat{e}(g, g_{q+1}{}^{s*\pi_{i,k}})}{\hat{e}(g, g^{s*\sum_{j=1;j\neq k}^{q} \pi_{i,j}*\alpha^{q+1-j+k}})}]^{\eta} \\
&= \hat{e}(g, g_{q+1}{}^{s*\pi_{i,k}})^{\eta} \\
&= \hat{e}(g, g_{q+1}{}^{s*\eta*\pi_{i,k}}) \\
&= \hat{e}(g, \sigma_1)
\end{aligned}$$

This proves the correctness of the verification step (i.e., $\hat{B}_i = B_i$). The non-singularity of the pairing function allows to state that Equation 7.4 is true if, and only if $\hat{B}_i = B_i$. □

- Private Single Data Block Verification

In order to improve the energy efficiency, while reducing the processing complexity at the client side, we propose a lightweight private verification variant of SHoPS, relying on the private key of the data owner. For this purpose, we squeeze the proposed checking algorithm, presented in Equation 7.4, in order to support only two pairing functions computation. As such the private verification of a single data block B_i is as follows.

$$\hat{e}({g_k}^{\eta}, \varpi_i) \star \hat{e}(g, \sigma_1 {\sigma_2}^{s\eta})^{-1} = 1 \tag{7.5}$$

Lemma 7.4.2 Private Single Data Block Verification Correctness *The verification procedure of Equation 7.5 holds if, and only if the data block file $\hat{B}_i = B_i$.*

Lemma 7.4.3 Private Single Data Block Verification Correctness – *The verification procedure of Equation 7.5 holds if, and only if the data block file $\hat{B}_i = B_i$.*

Proof.
Having received (σ_1, σ_2) from the cloud server, the data owner first calculates $\hat{e}(g, \sigma_1 {\sigma_2}^{s\eta})$, using his secret key $pr = s$ and the random nonce η.
Then, based on the public accumulator ϖ_i, the client checks that $\hat{e}(g, \sigma_1 {\sigma_2}^{s\eta})$ is equal to $\hat{e}({g_k}^{\eta}, \varpi_i)$ as:

$$\begin{aligned}
\hat{e}(g, \sigma_1 {\sigma_2}^{s\eta}) &= \hat{e}(g, (\hat{pk}^{\eta})^{\pi_{i,k}} \star [\prod_{j=1; j\neq k}^{q} g_{q+1-j+k}^{\pi_{i,j}}]^{s\eta}) \\
&= \hat{e}(g, (\hat{g_{q+1}}^{s\eta})^{\pi_{i,k}} \star [\prod_{j=1; j\neq k}^{q} g_{q+1-j+k}^{\pi_{i,j}}]^{s\eta}) \\
&= \hat{e}(g, [\prod_{j=1}^{q} g_{q+1-j+k}^{\pi_{i,j}}]^{s\eta}) \\
&= \hat{e}(g, g^{s\eta * \sum_{j=1}^{q} \pi_{i,j} * \alpha^{q+1-j+k}}) \\
&= \hat{e}(g, [g^{s * \sum_{j=1}^{q} \pi_{i,j} * \alpha^{q+1-j}}]^{\eta\alpha^k}) \\
&= \hat{e}(g^{\eta\alpha^k}, g^{s * \sum_{j=1}^{q} \pi_{i,j} * \alpha^{q+1-j}}) \\
&= \hat{e}({g_k}^{\eta}, \varpi_i)
\end{aligned} \tag{7.6}$$

This proves the correctness of the verification step (i.e., $\hat{B}_i = B_i$). The non-singularity of the pairing function allows to state that Equation 7.5 is true if, and only if $\hat{B}_i = B_i$. □

In this section, we presented the procedures introduced by SHoPS, for a single data block proof verification. Indeed, according to the standard definition of proof systems, SHoPS has to fulfill the completeness and soundness of verification [Gol00]. These two properties are detailed in Section 7.5.2.

Additionally, We have shown the sequence of operations performed by the storage node, in order to generate a proof of data possession for a given data file block $B_i \in 2^{\mathcal{M}}$, and the sequence of operations performed by the verifier in order to check the correctness of the received proof P.
Next, we extend the verification of a single data block, to support several blocks, relying on set homomorphic properties.

7.4.2 Set-Homomorphic Properties of the proposed Scheme

In this section, we extend the design of the data block elementary checking, in order to support subsets of data blocks. That is, the verifier requests the cloud for a fault tolerance and data correctness proofs, while considering a sequence of set-homomorphism properties. For ease of presentation, we prove the different properties, using two different data blocks B_i and B_j. Our operations can be extended easily to support multiple data blocks checking. In the following, we denote by $\sigma_{1,B_i \bullet B_j}$ the first proof element of $B_i \bullet B_j$ and by $\sigma_{2,B_i \bullet B_j}$ the second proof element, where $\bullet$ is the set operator.

7.4.2.1 Set-Union Operator

In order to prove that our scheme is set-homomorphic with regard to the union operator, we use the received proofs $\texttt{prf}(c, B_i)$ and $\texttt{prf}(c, B_j)$ corresponding to B_i and B_j, respectively, to express $\texttt{prf}(c, B_i \cup B_j)$, based on the same challenge c.

Lemma 7.4.4 *For every data block B_i and B_j, the union operator is defined as: $B_i \cup B_j = B_i + B_j - B_i \cap B_j$*

To this purpose, we first express $\varpi_{B_i \cup B_j}$, using ϖ_{B_i} and ϖ_{B_j}, as follows.

Lemma 7.4.5 *For every data blocks $B_i = \{\pi_{i,1}, \cdots, \pi_{i,q}\}$ and $B_j = \{\pi_{j,1}, \cdots, \pi_{j,q}\}$, where $\pi_{i,k} \in 2^{\mathcal{M}}$ and $1 \leq k \leq q$; and given the accumulators ϖ presented in Algorithm 15:, the union accumulator is such that,*

$$\varpi_{B_i \cup B_j} = lcm(\varpi_{B_i}, \varpi_{B_j}) \tag{7.7}$$

Proof.
The computation of $\varpi_{B_i \cup B_j}$, is performed as follows:

$$\begin{aligned}
\varpi_{B_i \cup B_j} &= \prod_{\pi_{k,l} \in B_i \cup B_j; l \in [1,q]; k \in \{i,j\}} {g_{q+1-l}^{\pi_{k,l}}}^{pr} \\
&= lcm(\prod_{\pi_{i,l} \in B_i; l \in [1,q]} g_{q+1-l}^{pr*\pi_{i,l}}, \prod_{\pi_{j,l} \in B_j; l \in [1,q]} g_{q+1-l}^{pr*\pi_{j,l}}) \\
&= lcm(\varpi_{B_i}, \varpi_{B_j})
\end{aligned}$$

To compute the least common multiple of B_i and B_j, we use the relation between gcd and lcm, as follows.

$$gcd(\varpi_{B_i}, \varpi_{B_j}) * lcm(\varpi_{B_i}, \varpi_{B_j}) = \varpi_{B_i} \varpi_{B_j}$$

In the sequel, we have:

$$\varpi_{B_i \cup B_j} = lcm(\varpi_{B_i}, \varpi_{B_j}) = \frac{\varpi_{B_i} \star \varpi_{B_j}}{gcd(\varpi_{B_i}, \varpi_{B_j})} \tag{7.8}$$

For instance, using the Bézout's lemma, there exist unique integers a and b, such that:

$$a\varpi_{B_i} + b\varpi_{B_j} = gcd(\varpi_{B_i}, \varpi_{B_j}) \tag{7.9}$$

As such, using the Equation 7.8 and Equation 7.9, we find the lcm of the two data blocks B_i and B_j as follows:

$$\varpi_{B_i \cup B_j} = lcm(\varpi_{B_i}, \varpi_{B_j}) = \frac{\varpi_{B_i} \star \varpi_{B_j}}{a\varpi_{B_i} + b\varpi_{B_j}} \tag{7.10}$$

Therefore, we obtain the proof of the lemma 7.4.5. □

We use Lemma 7.4.5 to prove the set homomorphism property of SHoPS, with respect to the union operator.

Theorem 7.4.6 Set-Homomorphism Property – Union Operator *SHoPS considers the algorithms* `clg`, `prf` *and* `vrf` *defined above. Let* `agg` *be the algorithm, presented in Equation 7.1, such that* $\bullet$ *is the set union operator, as follows.*

$$\texttt{prf}_2(B_i \bullet B_j, c) = \texttt{prf}_2(B_i, c) \odot \texttt{prf}_2(B_j, c) =$$

$$\texttt{prf}_2(B_i, c) \star \texttt{prf}_2(B_j, c)(a * \texttt{prf}_2(B_i, c) + b * \texttt{prf}_2(B_j, c))^{-1} \tag{7.11}$$

where a and b satisfy : $a\texttt{prf}_2(B_i, c) + b\texttt{prf}_2(B_j, c) = gcd(\texttt{prf}_2(B_i, c), \texttt{prf}_2(B_j, c))$

Proof.
We prove that SHoPS fulfills the homomorphism and correctness properties.

- **Proof of Homomorphism** – We know that $a\texttt{prf}_2(B_i, c) + b\texttt{prf}_2(B_j, c) = a\hat{\varpi}_{B_i} + b\hat{\varpi}_{B_j}$. Thus, we can write: $b\hat{\varpi}_{B_i}^{-1} + a\hat{\varpi}_{B_j}^{-1} = \hat{\varpi}_{B_i \cup B_j}^{-1}$.

  Consequently, using Equation 7.10, we can write that:

$$\begin{aligned} a\hat{\varpi}_{B_i} + b\hat{\varpi}_{B_j} &= \frac{(a\hat{\varpi}_{B_i} + b\hat{\varpi}_{B_j}) \star \hat{\varpi}_{B_i}^{-1}\hat{\varpi}_{B_j}^{-1}}{\hat{\varpi}_{B_i}^{-1}\hat{\varpi}_{B_j}^{-1}} \\ &= \frac{a\hat{\varpi}_{B_i}\hat{\varpi}_{B_i}^{-1}\hat{\varpi}_{B_j}^{-1} + b\hat{\varpi}_{B_j}\hat{\varpi}_{B_i}^{-1}\hat{\varpi}_{B_j}^{-1}}{\hat{\varpi}_{B_i}^{-1}\hat{\varpi}_{B_j}^{-1}} \\ &= \frac{a\hat{\varpi}_{B_j}^{-1} + b\hat{\varpi}_{B_i}^{-1}}{\hat{\varpi}_{B_i}^{-1}\hat{\varpi}_{B_j}^{-1}} \\ &= \hat{\varpi}_{B_i \cup B_j}^{-1} \star \hat{\varpi}_{B_i} \star \hat{\varpi}_{B_j} \end{aligned}$$

  As such, we demonstrate that $\hat{\varpi}_{B_i \cup B_j}^{-1} = \frac{a\hat{\varpi}_{B_i} + b\hat{\varpi}_{B_j}}{\hat{\varpi}_{B_i} \star \hat{\varpi}_{B_j}}$.

  This proves that our framework fulfills the homomorphism property.

- **Proof of Correctness** – We show that an authorized challenger may check the correctness of two different data blocks B_i and B_j, using an aggregate proof $\texttt{prf}_2(B_i \cup B_j, c)$, based on a challenge $c = (k, \hat{pk}^{\eta})$.
  We suppose that $\pi_{i,k} \neq \pi_{j,k}$. That is, as presented in Equation 7.2, the correctness of SHoPs is that $\texttt{vrf}(\texttt{prf}(pk, B_i \cup B_j), pk, \hat{pk}) = 1$.
  We have:

$$\hat{e}(g_k, \varpi_{B_i \cup B_j})\hat{e}(pk, \sigma_{2,B_i \cup B_j})^{-1} = \frac{\hat{e}(g^{\alpha^k}, \varpi_{B_i \cup B_j})}{\hat{e}(g^{pr}, \hat{\varpi}_{B_i \cup B_j})}$$

$$\begin{aligned}
&= \frac{\hat{e}(g^{\alpha^k}, \frac{\varpi_{B_i}\varpi_{B_j}}{gcd(\varpi_{B_i}, \varpi_{B_j})})}{\hat{e}(g^s, \frac{\hat{\varpi}_{B_i}\hat{\varpi}_{B_j}}{gcd(\hat{\varpi}_{B_i}, \hat{\varpi}_{B_j})})} \\
&= \frac{\hat{e}(g, \frac{\prod_{l=1}^{q} g_{q+1-l+k}^{\pi_{i,l}}{}^{s} \prod_{l=1}^{q} g_{q+1-l+k}^{\pi_{j,l}}{}^{s}}{gcd(\varpi_{B_i}, \varpi_{B_j})^{\alpha^k}})}{\hat{e}(g, \frac{\prod_{l=1;l\neq k}^{q} g_{q+1-l+k}^{s\pi_{i,l}} \prod_{l=1;l\neq k}^{q} g_{q+1-l+k}^{s\pi_{j,l}}}{gcd(\hat{\varpi}_{B_i}, \hat{\varpi}_{B_j})^{s}})} \\
&= \frac{\hat{e}(g, \frac{{g_{q+1}}^{s\pi_{i,k}} \prod_{l=1;l\neq k}^{q} g_{q+1-l+k}^{s\pi_{i,l}} {g_{q+1}}^{s\pi_{i,k}} \prod_{l=1;l\neq k}^{q} g_{q+1-l+k}^{s\pi_{j,l}}}{gcd(\varpi_{B_i}, \varpi_{B_j})^{\alpha^k}})}{\hat{e}(g, \frac{\prod_{l=1;l\neq k}^{q} g_{q+1-l+k}^{s\pi_{i,l}} \prod_{l=1;l\neq k}^{q} g_{q+1-l+k}^{s\pi_{j,l}}}{gcd(\hat{\varpi}_{B_i}, \hat{\varpi}_{B_j})^{s}})} \\
&= \frac{\hat{e}(g, \frac{\hat{\varpi}^s_{B_i}\hat{\varpi}^s_{B_i}}{gcd(\varpi_{B_i}, \varpi_{B_j})^{\alpha^k}})\hat{e}(g, {g_{q+1}}^{s\pi_{i,k}} {g_{q+1}}^{s\pi_{j,k}})}{\hat{e}(g, \frac{\hat{\varpi}^s_{B_i}\hat{\varpi}^s_{B_i}}{gcd(\hat{\varpi}_{B_i}, \hat{\varpi}_{B_j})^{s}})} \\
&= \hat{e}(g, {g_{q+1}}^{s\pi_{i,k}} {g_{q+1}}^{s\pi_{j,k}})
\end{aligned}$$

  As such, based on the random challenge η, we can write $[\hat{e}(g_k, \varpi_{B_i \cup B_j})\hat{e}(pk, \sigma_{2,B_i \cup B_j})^{-1}]^{\eta}$ as follows:

$$\begin{aligned}
&= [\hat{e}(g, {g_{q+1}}^{s\pi_{i,k}} {g_{q+1}}^{s\pi_{j,k}})]^{\eta} \\
&= \hat{e}(g, {g_{q+1}}^{s\pi_{i,k}\eta} {g_{q+1}}^{s\pi_{j,k}\eta}) \\
&= \hat{e}(g, \sigma_{1,B_i} \star \sigma_{1,B_j}) \\
&= \hat{e}(g, \sigma_{1,B_i \cup B_j})
\end{aligned}$$

  Therefore, we prove the correctness of SHoPS with respect to the set-union operator and the compliance to Equation 7.4.

□

The set-homomorphic property proved in Theorem 7.4.6, is useful in practice to generate an aggregate proof for a set of data blocks, with no need for private keys of the storing nodes. That is, SHoPS guarantees the privacy of these storing nodes. For sake of efficiency and granularity, we investigate the case where the cloud gateway generates a proof of a subset of data blocks.

7.4.2.2 Set-Inclusion Operator

In this section, we prove that SHoPS is homomorphic with respect to the set-inclusion operator.

Theorem 7.4.7 Set-Homomorphism Property – Subset Operator *SHoPS considers the algorithms* `clg`, `prf` *and* `vrf` *defined above. Let* `agg` *be the algorithm, presented in Equation 7.1, such that* $\bullet$ *is the set inclusion operator, as follows.*

$$\mathtt{prf}_2(B_i \bullet B_j, c) = \mathtt{prf}_2(B_i, c) \odot \mathtt{prf}_2(B_j, c) =$$

$$\mathtt{prf}_2(B_j, c) \star \mathtt{prf}_2(B_i, c)^{-1} \tag{7.12}$$

where B_i *and* B_j *are two data blocks of* $\in 2^{\mathcal{M}}$*, and* $B_i \subset B_j$*.*

We prove the homomorphism and the correctness of SHoPS with respect to the set inclusion operator.

Proof.
Let B_i and B_j be two data blocks, where $B_i \subset B_j$, and k is the index challenge sent by the verifier.

- **Proof of Homomorphism** – We have $\mathtt{prf}_2(B_j, c) = \hat{\varpi}_{B_j}$. This can write, where $l \neq k$:

$$\begin{aligned} \hat{\varpi}_{B_j} &= \prod_{\pi_{j,l} \in B_j, l \in [1,q]} g_{q+1-l+k}^{s\pi_{j,l}} \\ &= \prod_{\pi_{j,l} \in B_j \setminus B_i, l \in [1,q]} g_{q+1-l+k}^{s\pi_{j,l}} \prod_{\pi_{i,l} \in B_i, l \in [1,q]} g_{q+1-l+k}^{s\pi_{i,l}} \end{aligned}$$

  As such, we show that $\prod_{\pi_{j,l} \in B_j \setminus B_i, l \in [1,q]; l \neq k} g_{q+1-l+k}^{s\pi_{j,l}}$

$$= \prod_{\pi_{j,l} \in B_j, l \in [1,q]; l \neq k} g_{q+1-l+k}^{s\pi_{j,l}} \star \prod_{\pi_{i,l} \in B_i, l \in [1,q]; l \neq k} g_{l-q-1-k}^{-s\pi_{i,l}} \tag{7.13}$$

  Using Equation 7.13, we demonstrate that SHoPS is homomorphic with respect to the set-inclusion operator:

$$\begin{aligned} \mathtt{prf}_2(B_j \setminus B_i, c) &= \prod_{\pi_{j,l} \in B_j \setminus B_i, l \in [1,q]; l \neq k} g_{q+1-l+k}^{\pi_{j,l}} \\ &= \hat{\varpi}_{B_j} \star \hat{\varpi}_{B_i}^{-1} \\ &= \mathtt{prf}_2(B_j, c) \star \mathtt{prf}_2(B_i, c)^{-1} \end{aligned}$$

- **Proof of Correctness** – The correctness of SHoPS is that $\texttt{vrf}(pk, \hat{pk}, \texttt{prf}_2(B_j \setminus B_i, c)) = 1$, where $B_i \subset B_j$.

$$
\begin{aligned}
[\frac{\hat{e}(g_k, \varpi_{B_j \setminus B_i})}{\hat{e}(pk, \sigma_{2,B_j \setminus B_i})}]^{\eta} &= [\frac{\hat{e}(g^{\alpha^k}, \varpi_{B_i}^{-1} \star \varpi_{B_j})}{\hat{e}(g^{pr}, \hat{\varpi}_{B_i}^{-1} \star \hat{\varpi}_{B_j})}]^{\eta} \\
&= [\frac{\hat{e}(g, {\sigma_{1,B_i}}^{\eta^{-1}} {\sigma_{1,B_j}}^{\eta^{-1}} \hat{\varpi}_{B_i}^{-s} \star \hat{\varpi}_{B_j}^{s})}{\hat{e}(g, \hat{\varpi}_{B_i}^{-s} \star \hat{\varpi}_{B_j}^{s})}]^{\eta} \\
&= [\frac{\hat{e}(g, {\sigma_{1,B_i}}^{\eta^{-1}} {\sigma_{1,B_j}}^{\eta^{-1}}) \hat{e}(g, \hat{\varpi}_{B_i}^{-s} \star \hat{\varpi}_{B_j}^{s})}{\hat{e}(g, \hat{\varpi}_{B_i}^{-s} \star \hat{\varpi}_{B_j}^{s})}]^{\eta} \\
&= \hat{e}(g, {\sigma_{1,B_i}}^{\eta^{-1}} {\sigma_{1,B_j}}^{\eta^{-1}})^{\eta} \\
&= \hat{e}(g, \sigma_{1,B_j \setminus B_i})
\end{aligned}
$$

Therefore, we obtain the proof of correctness of Theorem 7.4.7. □

While expanding the subset feature to support multiple data blocks, the homomorphism property with respect to the set-inclusion operator becomes interesting. That is, it allows a verifier to check the correctness of a directory, while excluding a big file.

7.4.2.3 Set-Intersection Operator

We have proved that SHoPS allows the generation of aggregate proofs with respect to the subset and the union operators. That is, we extend our discussion, using the relations between these two set operators. For instance, based on Theorem 7.4.6 and Theorem 7.4.7, we demonstrate that our scheme is set-homomorphic with respect to the intersection operator.

Lemma 7.4.8 *For every data blocks $B_i = \{\pi_{i,1}, \cdots, \pi_{i,q}\}$ and $B_j = \{\pi_{j,1}, \cdots, \pi_{j,q}\}$, where $\pi_{i,k} \in 2^{\mathcal{M}}$ and $1 \leq k \leq q$, the intersection operation may be expressed in terms of the union and the set difference operators as follows.*

$$B_i \cap B_j = (((B_i \cup B_j) \setminus (B_i \setminus B_j)) \setminus (B_j \setminus B_i)) \tag{7.14}$$

This corollary is specifically interesting, when applied at the CSP-storing nodes interface. Thus, the CSP checks the correctness of sets of data blocks, on the storing nodes, in order to mitigate to byzantine failures and unintentionally drive-crashes.

7.4.3 Energy efficiency

The design of SHoPs is motivated by the improvement of the energy-efficiency concern, while providing aggregate fault-tolerance verifications. As such, we show that the verification complexity of two separate block-proofs is more costly than the aggregate processing overhead of $\odot$.

In order to evaluate the energy efficiency, we compare SHoPS with respect to the union operator. That is, we show that SHoPS supports lightweight mechanisms allowing resource constrained devices to dynamic challenging contexts. To this purpose, we denote by M the cost of a multiplication of two elements in a multiplicative group $\mathbb{G}_q$, by E the exponentiation of an element of a multiplicative group by a scalar belonging to $\mathbb{Z}^*$, by A the addition of two multiplicative elements and by m the multiplication of a multiplicative elements by a scalar. We suppose that a classical cloud gate (CC) computes a new proof for the aggregated data blocks. If two data blocks B_i and B_j are stored on two different storing nodes N_i and N_j, the cloud gateway has to retrieve B_i and B_j. Then, he has to perform the aggregated proof of these blocks, before sending the response to the verifier. For instance, the computation complexity of the union of two sets is about $\mathcal{O}(q^2)$ [NF07]. On the other side, SHoPS requires the computation of the integers a and b satisfying $a\varpi_i + b\varpi_j = gcd(\varpi_i, \varpi_j)$. For this purpose, SHoPS executes the Euclidean algorithm involving $\delta log(q)$ operations. Table 7.1, presents a comparison between the different costs of aggregated proofs with respect to SHoPS and a classical CSP (CC), where i presents the number of storing nodes.

Table 7.1 - Comparison between SHoPS and a classical Cloud Service Provider (CC)

Classical CSP (CC)	SHoPS	
	Gate	Storing Nodes N_i
$\mathcal{O}(q^2)+$ $q(E+M)-M$	$i(M+A+m)-$ $A+M+\delta log(q)$	$q(E+M)-M$

Table 7.1 shows that SHoPS reduces the communication overhead and the computation complexity at the cloud gate side. For instance, the processing overhead is distributed over multiple storing nodes. As such, Lemma 7.4.9 evaluates the processing overhead at the cloud gate of SHoPS compared to a classical CSP, to generate an aggregated proof of two data blocks, by using SHoPS.

Lemma 7.4.9 *Let B_i and B_j two data blocks. The computation complexity difference between a classical CSP (CC) and SHoPS is as follows.*

$$\mathcal{P}_{CC} - \mathcal{P}_{SHoPS} = q(E+M) - 4M - 2m - A + \mathcal{O}(q^2) \tag{7.15}$$

Similarly, we show that the processing overhead at the verifier side is also optimized while checking an aggregated proof.

7.5 Security Discussion

In this section, we present a security analysis of SHoPS, based on two different threat models.

7.5.1 Threat Model

Two threat models have been considered. We first considered the case of a *cheap* and *lazy* cloud service provider. As such, the storage server wants to reduce its resources consumption. That is, it stores fewer redundant data. In addition, this *lazy* server claims doing the requested computations to provide responses to the challenger. Second, we pointed out the case of a malicious verifier that intends to get extra-information about the outsourced data of the data owner.

SHoPS must provide the capabilities to the verifier and the service provider to thwart the two threats mentioned above. We prove the security of SHoPS, assuming that the q-Diffie Hellman Exponent Problem (q-DHE) and the Computational Diffie Hellman (CDH) are hard in $\mathbb{G}_2$.

7.5.2 SHoPS Resistance to Cheap and Lazy Server Adversary

In order to save computation resources, the *lazy* cloud provider attempts to provide valid proofs without processing the outsourced data. SHoPS should be secure against *Forgery* and *Replay* attacks, defined as follows:

- *Forgery Attack* – the cloud server attempts to forge a valid accumulator on any block to deceive a verifier.
- *Replay Attack* – the server may use some previous proofs and other public information to generate a valid proof without accessing the challenged data.

To capture the malicious behaviour of the prover, we describe the security of SHoPS, using a security game, referred to as *SS-Game*. This game is an interactive algorithm between a *lazy* storage server adversary and an honest verifier. The adversary attempts to construct a valid proof without processing the original data block. That is, we define an adversary $\mathcal{A}$ which can query the data owner for getting data bock public parameters. We also define a challenger $\mathcal{C}$ which is responsible for simulating the system procedures to interact with the adversary. *SS-Game* includes two different phases, Phase I and Phase II, which are independently executed. We must note that, in Phase I, the challenger always presents the data owner. In Phase II, the challenger may be the data owner in a private verification scenario or an authorized verifier in a public verification scenario.

SS-Game is formally defined as follows:

- ***Phase I*** – the challenger $\mathcal{C}$ runs the first phase algorithms and gives the public parameters to the adversary $\mathcal{A}$:
  - *Setup* – the challenger $\mathcal{C}$ runs the first phase algorithms, in order to get the public elements $\{pk, \hat{pk}, \{g_i\}_{1 \leq i \leq 2q; i \neq q+1}, (ID_{B_i}, \varpi_i)_{i \in \{1,n\}}\}$ and his master private key pr.
  - *Query* – the adversary $\mathcal{A}$ gets from the challenger the public parameters.

- ***Phase II*** – the challenger $\mathcal{C}$ interacts with the adversary $\mathcal{A}$, to execute an interactive proof system:
  - *Challenge* – the challenger $\mathcal{C}$ requests the adversary to provide a valid proof of the requested data block, determined by a random challenge c_g.
  - *ForgeProof* – without processing on the original data file, the adversary $\mathcal{A}$ tries to compute a valid proof (σ_1^*, σ_2^*), the challenge c_g, and the public credentials $params = \{g_i\}_{1 \leq i \leq 2q; i \neq q+1}$.

The adversary $\mathcal{A}$ wins *SS-Game*, if the `vrf` procedure returns *accept*.

Definition 7.5.1 *SHoPS* $= \{$`gen`, `stp`, `clg`, `prf`, `vrf`, `agg`$\}$ *guarantees the integrity of outsourced data blocks, if for any probabilistic polynomial adversary $\mathcal{A}$, the probability that $\mathcal{A}$ wins* SS-Game *on a set of data blocks is negligibly close to the probability that the challenger $\mathcal{C}$ can extract those data blocks, based on a knowledge extractor Ψ.*

Proof.
Concretely, the challenger $\mathcal{C}$ interacts with the adversary $\mathcal{A}$ following the security game. The adversary $\mathcal{A}$ attempts to provide a valid proof (σ_1^*, σ_2^*), without accessing data. The generated proof (σ_1^*, σ_2^*) has to successfully verify Equation 7.4 in a public verification scenario, and Equation 7.5 in a private verification scenario.
Following a *Forge Attack* based on *SS-Game*, $\mathcal{A}$ tries to use only the public parameters, the accumulator ϖ_i of the requested data block B_i and the received challenge c_g. Thus, we have two cases, such that: (1) $\mathcal{A}$ sets σ_1^* and attempts to deduce a valid σ_2^* and (2) $\mathcal{A}$ sets σ_2^* and tries to derive a valid σ_1^*.

For the public data block verification, the challenger $\mathcal{C}$ is based on Equation 7.4 to check the correctness of the received proof. The adversary $\mathcal{A}$ chooses to replace σ_2^* by the public accumulator ϖ_i of the requested data block B_i. While considering that the correctness of verification requires the equality between $[\hat{e}(g_k, \varpi_i)\hat{e}(pk, \sigma_2)^{-1}]^\eta$ and $\hat{e}(g, \sigma_1)$, the first side of Equation 7.4 becomes equal to $[\hat{e}(g_k, \varpi_i)\hat{e}(pk, \sigma_2^*)^{-1}]^\eta$. As such, the adversary has to compute the appropriate σ_1^*, to be able to mislead the challenger, without accessing outsourced data.

In fact, based on σ_2^*, the first side of the verification equation is as follows:

$$\begin{aligned}
[\hat{e}(g_k, \varpi_i)\hat{e}(pk, \sigma_2^*)^{-1}]^\eta &= [\hat{e}(g_k, \varpi_i)\hat{e}(pk, \varpi_i)^{-1}]^\eta \\
&= [\hat{e}(g^{\alpha^k}, \varpi_i)\hat{e}(g^{-s}, \varpi_i)]^\eta \\
&= \hat{e}(g^{\alpha^k} g^{-s}, \varpi_i)^\eta \\
&= \hat{e}(g^{(\alpha^k - s)\eta}, \varpi_i) \\
&= \hat{e}(g, {\varpi_i}^{(\alpha^k - s)\eta})
\end{aligned} \tag{7.16}$$

From Equation 7.16 and Equation 7.4, we state that the adversary has to generate σ_1^* as ${\varpi_i}^{(\alpha^k - s)\eta}$, in order to get a valid proof. Knowing $pr = g^s$, g_k, $\hat{pk}^\eta$ and ϖ_i, the adversary $\mathcal{A}$ has to break the CDH or the q-DHE problems, in order to derive a persuasive response.

This is obviously in contradiction with our security assumptions. Similarly, we prove the resistance of SHoPS to a fraudulent prover, in a private data block verification scenario, while considering a data owner challenger.

After leading several valid proof interactions with a legitimate and honest verifier, the adversary $\mathcal{A}$ may attempts a *Replay Attack*. Thus, the adversary locally kept a set of challenges and the related proof responses as $\{\{c^{(1)}, (\sigma_1, \sigma_2)^{(1)}\}, \{c^{(2)}, (\sigma_1, \sigma_2)^{(2)}\}, \cdots, \{c^{(n)}, (\sigma_1, \sigma_2)^{(n)}\}\}$. Then, based on *SS-Game*, $\mathcal{A}$ tries to deduce a valid proof, while relying on previous sessions' proofs.

For a public verification scenario, we suppose that the challenger $\mathcal{C}$ generates a new challenge $c_g = (k_g, \hat{pk}^{\eta_g})$, where k_g is a previously requested index and $\hat{pk}^{\eta_g}$ is a new generated element, due to the usage of a random nonce η_g, derived by a pseudo random generator. In this setting, the adversary $\mathcal{A}$ may deduce a valid $\sigma_2{}^* = \hat{\varpi}_i$, since the computation of $\hat{\varpi}_i$ depends only on the requested index k. However, $\mathcal{A}$ has to provide a correct $\sigma_1{}^*$ to deceive the challenger. A valid $\sigma_1{}^*$ is equal to $(\hat{pk}^{\eta})^{\pi_{i,k}}$. It is clear that the adversary cannot extract a correct $\sigma_1{}^*$, while only relying on previous interactions without accessing to data, thanks to the usage of the CDH assumption.
In addition, if we suppose, by inconsistency, that the adversary $\mathcal{A}$ chooses to perform a *Replay Attack*, while accessing to data, to provide a correct $\sigma_1{}^*$. This setting requires, for the attacker $\mathcal{A}$, to store q group elements per data block and, nevertheless, to access data in order to retrieve the related π_{i,k_g}. In the sequel, the *Replay Attack* does not cost effective and beneficial to the provider, because it still require the access to correct data block and adds an important storage overhead.

Additionally, in order to prove SHoPS resistance to a fraudulent server prover, we assume that there is a knowledge extractor algorithm Ψ [Gol00], which gets the public parameters as input, and then attempts to break the CDH assumption in $\mathbb{G}_2$. The Ψ algorithm interacts as follows:
Learning 1– the first learning only relies on the data owner public key $pk = g^{pr}$ as input. Ψ tries to get knowledge of the client secret key pr. That is, the extractor algorithm Ψ picks at random $r_i \in_R [0,\ R[$, where $i \in \mathbb{Z}_p$ and computes g^{r_i}. For each r_i, Ψ checks whether the comparison holds between pk and g^{r_i}. Based on our assumption, Ψ cannot extract the secret key of the client with noticeable probability.
Learning 2– the input of the second learning is the tuple $(pk, \hat{pk}, g)$. The algorithm attempts to extract the secret key $pr = s$ by performing following steps:

1. $\hat{e}(pk, \hat{pk}) = \hat{e}(g^s, g_{q+1}{}^s) = g_{q+1}{}^{s^2}$
2. $\hat{e}(g, pk) = \hat{e}(g, g^s) = g^s$
3. $\hat{e}(g, g_{q+1}{}^s) = g_{q+1}{}^s$

This learning cannot hold, because of the DDH assumption. In [Bon98], Boneh demonstrates that the DDH assumption is far stronger than the CDH. □

In the following, we denote by S the set of the correct inputs of the `vrf` procedures. These inputs are needed to provide a valid proof. As introduced in Section 7.4.1.3, SHoPS has to fulfill two security requirements: completeness and soundness of verification.

- ***Soundness of Verification*** – the soundness means that for every input $x \notin S$ and every potential prover $\mathcal{A}$, the challenger $\mathcal{C}$ rejects with probability at least $\frac{1}{2}$ after interacting with $\mathcal{A}$, on common input x.

$$\forall x \notin S, \forall \mathcal{A}, Pr[(\mathcal{C} \Leftrightarrow \mathcal{A})(x) accepts] \leq \frac{1}{2} \tag{7.17}$$

  As such, the soundness implies that it is infeasible to confound the verifier to accept false proofs (σ_1^*, σ_2^*). That is, even if a collusion is attempted, the attacker $\mathcal{A}$ cannot prove the integrity of outsourced data. The soundness of SHoPS is relatively close to the security game *SS-Game*.

- ***Completeness of Verification*** – the completeness means that for every $x \in S$, a legitimate verifier V always accepts after interacting with a honest prover V, on common input x.

$$\forall x \in S, Pr[(V \Leftrightarrow P)(x) accepts] = 1 \tag{7.18}$$

  Hence, the completeness meets the correctness of verification (Equation 7.4 and Equation 7.5), while considering the non singularity of the pairing functions.

  For the public data block verification, the completeness property implies the public verifiability property, which allows any entity, not just the client (data owner), to challenge the cloud server for data possession or data integrity without the need for any secret information. That is, public verification elements, needed in the verification process are publicly known. Thereby, any authorized user may challenge the server storage and efficiently verifies the proof of data possession. Hence, SHoPS is a public verifiable protocol.

  Lemma 7.5.2 Completeness of Public Verification – *Given the tuple of public elements $(pk, \hat{pk}, \sigma_1, \sigma_2, params)$ and $B_i = \hat{B}_i$, the completeness of verification condition implies that Equation 7.4 holds in $\mathbb{G}_2$.*

  For the private data block verification, the completeness requirement is presented by Lemma 7.5.3 as follows:

  Lemma 7.5.3 Completeness of Private Verification – *Given the tuple of public elements $(pk, \hat{pk}, pr, \sigma_1, \sigma_2, params)$ and $B_i = \hat{B}_i$, the completeness of verification condition implies that Equation 7.5 holds in $\mathbb{G}_2$.*

In addition, in order to mitigate a *cheap* and *lazy* cloud service provider, we extend SHoPS to support a time-based function [BVDJ+11]. That is, we suppose that the cloud server claims storing n redundant copies of the same data block, while enforcing less redundancy than promised, or using fewer storing nodes than needed. For instance, the verifier has to send n simultaneous challenges to the cloud gate. Thus, based on the time taken for responding, the verifier may detect a malicious behavior of the cloud. That is, if the cloud takes t seconds to respond a challenge of one data block proof possession, he must take only $t + \theta$, where θ presents the bandwidth latency due to the transmission of the different n data block proofs.

7.5.3 SHoPS Resistance to Malicious Verifier Adversary

In the following analysis, we discuss the resistance of SHoPS to malicious verifier adversary, when only considering the vulnerabilities over the data file. As such, we suppose the existence of a public verifier adversary $\mathcal{A}$. He attempts to gain knowledge about the outsourced data, based on the public elements of the data owner and multiple interactions with the legitimate storage server. The adversary keeps a list of previous queries, including challenges, the proof verification queries, and the responses to those queries.
We suppose that the adversary $\mathcal{A}$ is not considered to perform preservation of computation resources by reusing the same random challenge η from one possession proof session to another. The verifier is assumed to renew the random scalar η to calculate the challenge $\hat{pk}^{\eta}$ for each session.
We suppose that the goal of the fraudulent verifier is to obtain information about the outsourced data file. That is, the attacker may request the same position index challenge k. As such, using two different sessions $((\alpha), (\beta))$, the attacker computes Equation7.19 and Equation 7.20 as follows:

$$\sigma_1{}^{(\alpha)} \star \sigma_1{}^{(\beta)} = \hat{pk}^{\pi_{i,k}\eta_{(\alpha)}} \star \hat{pk}^{\pi_{i,k}\eta_{(\beta)}} = \hat{pk}^{\pi_{i,k}(\eta_{(\alpha)}+\eta_{(\beta)})} \tag{7.19}$$

$$\sigma_1{}^{(\alpha)} \star \sigma_1{}^{(\beta)^{-1}} = \hat{pk}^{\pi_{i,k}\eta_{(\alpha)}} \star \hat{pk}^{-\pi_{i,k}\eta_{(\beta)}} = \hat{pk}^{\pi_{i,k}(\eta_{(\alpha)}-\eta_{(\beta)})} \tag{7.20}$$

Knowing the challenge k, the attacker cannot reconstruct pieces of the file data, based on the CDH assumption. The prover sends only the pair (σ_1, σ_2) to the verifier. Hence, it is likely impossible to extract information $\{\pi_{i,j}\}_{j\in[1,q]}$ from the server response. Thus, the randomness property is also necessary for the non triviality of the proof.

7.6 Experimental Study

In this section, we first present the context of the implementation of our proposed scheme. Then, we discuss the computation performances. In an effort to evaluate the performances of SHoPS, we implement several cryptographic operations at the client side. Our tests are conducted in order to understand the execution cost of SHoPS on real hardware. That is, on one hand, we evaluated some mathematical operations durations, such as exponentiation and multiplication in a multiplicative group $\mathbb{G}$. On the other hand, we study the processing cost of our randomized single data block procedures, relying on different security levels.

As presented in Section 7.3, our single data block proof is made up 5 randomized algorithms: `gen`, `stp`, `clg`, `prf` and `vrf`. Among these algorithms, `gen` and `stp` are performed by the data owner. To generate the public parameters, the client performs $2q + 1$ exponentiations in $\mathbb{G}$. In the `stp` procedure, this latter executes q exponentiations and multiplications in order to generate the accumulator ϖ, which remains linearly dependent on the data size. Note that, this `gen` and `stp` algorithms are one-time cost for the data owner and can be performed apart the other procedures.

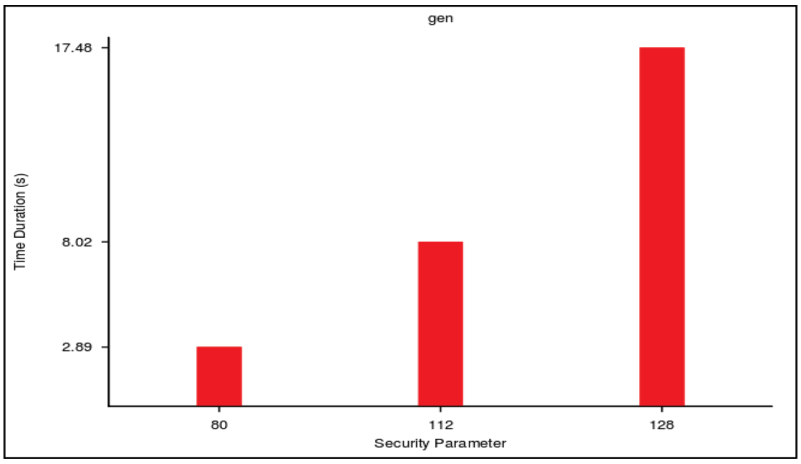

FIGURE 7.1 - Computation cost of the `gen` procedure (s)

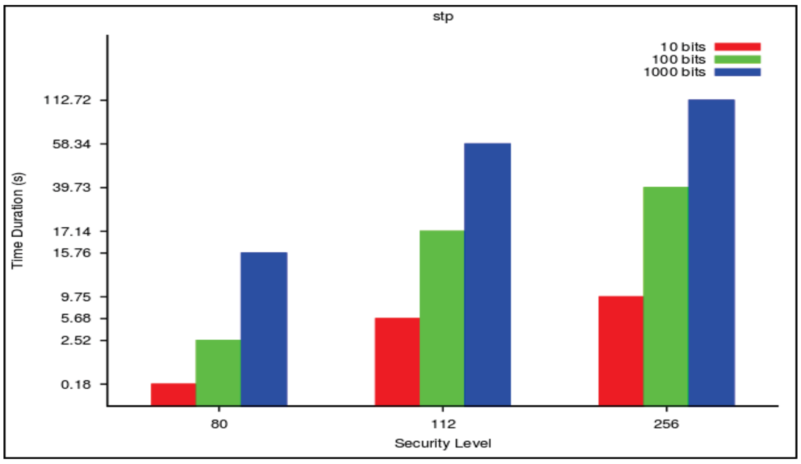

FIGURE 7.2 - Computation cost of the `stp` procedure (s)

Figure 7.1 and Figure 7.2 present the computation cost at the client side. Both procedures increase while increasing the security level parameter. Indeed, for a security level set to 80, the `gen` algorithm takes only 2.89 seconds to generate all the public elements and the private key of the data owner. This overhead remains interesting especially for resource-constrained devices. In addition, we study the behavior of the `stp` procedure, while increasing the size of each data block. We deduce from Figure 7.2 that it is important to take into consideration the size of data blocks, as the computation cost increases with respect to the size of data blocks. As an example, we recommend using 100 bits data block length, for 80 security level. That is, the `stp` algorithm generates the public accumulator, in only 2.52 seconds.

For each proof generation, the server computes the related accumulator of the related data block with respect to the position index k, and performs q exponentiations and multiplications in order to generate the couple (σ_1, σ_2). Upon receiving the server proof, the verifier conducts 3 pairing computations.

As the number of operations in a multiplicative group is important criterion to evaluate the system performance, Table 7.2 summarizes the computation costs of some mathematical operations in a multiplicative group.

Table 7.2 - Mathematical operations costs in a multiplicative group $\mathbb{G}_q$ (in ms)

Security Level	80	112	128
Exponentiation	0.624	2.389	5.022
Multiplication	5.18	21.187	66.813

Figure 7.3 depicts the computation cost of the different interactive procedures of SHoPS: `clg`, `prf` and `vrf`. That is, `clg` and `vrf` algorithms are executed at the client side, while the `prf` procedure is performed at the server side. For our tests, we used an Intel core 2 duo, to evaluate the three procedures, for a security parameter set to 80.

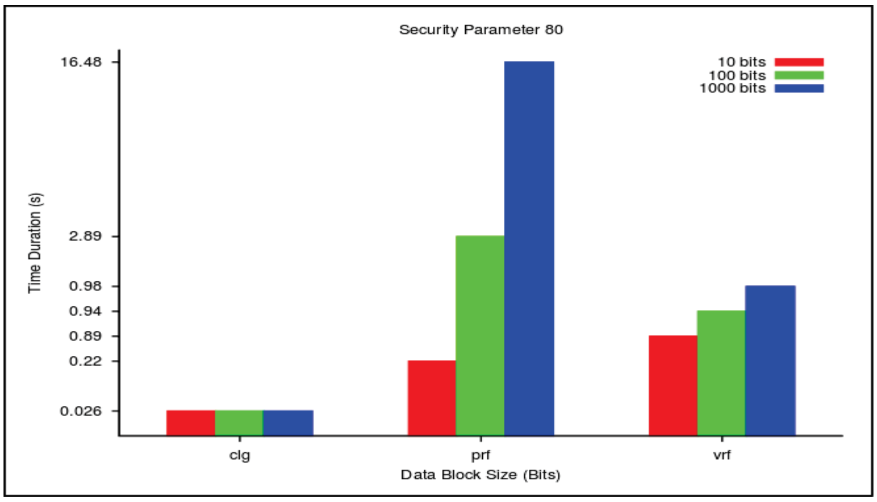

Figure 7.3 - Computation cost of {`clg`, `prf` and `vrf`} procedures (s)

It is worth noticing that `prf` is the slowest procedure, compared to the two other algorithms. That is, this algorithm performs q exponentiations and multiplications in a multiplicative group, for each proof generation. By definition, the cloud server is assumed to have a lot of processing capabilities and storage capacities. Thus, the proof generation will have no considerable effect on its natural function. Additionally, we notice, from Figure 7.3, that the computation complexity, at the client side is around one second for a proof verification, for 100 bits data block length.

7.7 Theoretical Performance Analysis

In this section, we present a theoretical performance analysis, while taking into consideration computation, communication and storage costs.

7.7.1 Computation Cost Evaluation

To evaluate the objectives given in Section 5.3, we compare, in Table 7.3, SHoPS with some existing techniques. On the basis of the requirements of a data possession proof system, we choose four different PDP schemes ([ABC+07,DVW09,SW08,EKPT09]), that are most closely-related to our context.

TABLE 7.3 - Complexity comparison between different PDP techniques (n is the number of data blocks)

Metrics	[ABC+07]	[DVW09]	[SW08]	[EKPT09]	SHoPS
Nb. of chall.	fixed	∞	∞	∞	∞
Public verif	Yes	No	Yes	No	Yes
CSP cmp. cost	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(logn)$	$\mathcal{O}(logn)$
User cmp. cost	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(logn)$	$\mathcal{O}(logn)$
Band. cost	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
Storage cost	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$

Table 7.3 states the computation cost comparison between our scheme and previous works, at both client and server side.
On the server side, SHoPS distributes the processing overhead over the multiple storing nodes. The cloud gate computes only i multiplications, where i is the number of the requested nodes, in an aggregated proof. Therefore, contrary to the other approaches, SHoPS achieves a $\mathcal{O}(logn)$ server computation complexity.
On the verifier side, we brought additional computation cost, in order to perform a public verifiability. That is, the public verification procedure can also be performed by authorized challengers without the participation of the data owner. As such, this concern can be handled in practical scenarios, compared to private schemes ([DVW09, EKPT09]) which have to centralize all verification tasks to the data owner. In our scheme, the authorized verifier has to generate two random scalars $c \in]0, R[$ and $k \in [1, q]$, in order to conduct his challenge request. Then, he checks the received proof from the cloud server, while performing three pairing computations, regardless the number of data blocks. Thus, the public verifiability introduces a $\mathcal{O}(nlogn)$ processing cost at the verifier side.

7.7.2 Bandwidth Cost Evaluation

In our proposed scheme, the bandwidth cost comes from the generated challenge message `clg` algorithm and the proof response in each verification request. On one hand, the exchanging challenge algorithm consists in transmitting one random position index k, where $k \in_R \mathbb{Z}_q$ and one element $\hat{pk}^{\eta}$. For a recommended security, we consider a security parameter $\lambda = 80$ bits, thus, the total cost of the challenge message is the double size of a group element of a multiplicative $\mathbb{G}$.

On the other hand, the proof response consists only in two elements $(\sigma_1, \sigma_2) \in \mathbb{G}^2$. Therefore, the total bandwidth cost becomes constant and the bandwidth complexity of our scheme is $\mathcal{O}(1)$.

As shown in Table 7.3, [DVW09] and [SW08] present $\mathcal{O}(n)$ bandwidth complexity, where n is the number of encoded data blocks. As a consequence, the bandwidth cost of these algorithms is linear to n. Considering the number of permitted challenges, [ABC+07] suffers from the problem of pre-fixed number of challenges, which is considered as an important requirement to the design of our construction. Nevertheless, their scheme presents a constant bandwidth cost, just like our proposed protocol. Based on a private proof, [EKPT09] also performs a low bandwidth cost. However, this algorithm supports only private verification. Therefore, along with a public verification, our proposed scheme, allows each verifier to indefinitely challenge the server storage with a constant bandwidth cost.

7.7.3 Storage Cost Evaluation

On the client side, SHoPS only requires the data owner to keep secret his private key sk. The public elements of a data file consist in the different accumulators of each data block $\varpi_{i\{}i \in [1, n]\}$, where n is the number of data blocks. Thus, the storage size of each client is $|sk|$. We must note that $|sk|$ is the size of the secret key of a SHoPS client, which is dependent on the security parameter λ. This storage overhead remains acceptable and attractive for resource constrained devices, mainly as it not dependent on the number of data blocks and the size of data.

7.8 Conclusion

In this chapter, we presented SHoPS, a Set-Homomorphic Proof of Data Possession scheme, supporting the 3 levels of data verification. SHoPS enables a verifier not only to obtain a proof of possession from the remote server, but also to verify that a given data file is distributed across multiple storage devices to achieve a certain desired level of fault tolerance. Indeed, we presented the set homomorphism property, which extends malleability to set operations properties, such as union, intersection and inclusion.

Additionally, our proposal is deliberately designed to support public verifiability and constant communication and storage cost. That is, SHoPS allows an implementation of remote data checking at the three networking interfaces, namely, the client-CSP interface, the CSP-storing nodes interface and between two CSPs interface. This ensures the flexibility of SHoPS application and enables fulfilling each verifier request.

Besides, SHoPS has be proven resistant to data leakage attacks, while considering either a fraudulent prover or a cheating verifier, on the basis of a Data Possession game.
SHoPS presents high security level and low processing complexity. For instance, SHoPS saves energy within the CSP by distributing the computation over the multiple nodes. In fact, each node provides proofs of local data block sets. This is to make applicable, a resulting proof over sets of data blocks, satisfying several needs, such as, proofs aggregation.

Finally, an experimental study shows the feasibility of our proposal and gives support to theoretical performance measurements. That is, SHoPS provides acceptable computation overhead, mainly at the server side, while delegating the verification process to several storing nodes.

Chapter 8

Conclusion & Perspectives

The open mind never acts: when we have done our utmost to arrive at a reasonable conclusion, we still - must close our minds for the moment with a snap, and act dogmatically on our conclusions.

George Bernard Shaw - 1856-1950

IN CONCLUSION, we summarize how each of the research topics presented in the first chapter has been pursued, and the contributions which have resulted. Next, we reflect on how we can improve our contributions and provide new research directions.
Throughout this thesis, our main objective was to propose efficient cryptographic mechanisms, in order to ensure data security in cloud data storage environments. We build upon the fact that cloud service providers are not totally trusted by their customers.

Objective A consists of defining new methods to improve data confidentiality in cloud applications, while enhancing dynamic sharing between users. In response to this objective, we proposed two different approaches, based on the usage of ID-Based Cryptography (IBC) and the convergent cryptography, respectively.

In chapter 3, we introduced our first contribution based on a specific usage of Identity Based Cryptography [KBL13], in order to fulfill *Objective A*. In [KBL13], cloud storage clients are assigned the IBC–PKG function, where a per data key is derived locally from a data identifier and the data identity. Thanks to IBC properties, this contribution is shown to support data privacy and confidentiality, as it employs an original ID-based client side encryption approach. It is also shown to be resistant to unauthorized access to data and to any data disclosure during the sharing process, while considering two realistic threat models, namely an honest but curious server and a malicious user adversary.

In addition, due to the lightweight ID-based public key computation process and contrary to the existing classical sharing schemes, [KBL13] does not require for the depositor to be connected, when the recipients want to retrieve the shared data.
Two possible refinements were proposed. The first approach consists on the distribution of the PKG role partly to the cloud server and to the client, in order to alleviate the computation complexity at the client side. The second one introduces our second contribution which details a client-side deduplication scheme for cloud applications [KL14].

Second, we presented, in chapter 4, CloudaSec framework for securely sharing outsourced data via the public cloud [KLEB14]. CloudaSec, a public key based solution, ensures the confidentiality of content in the public cloud environments with flexible access control policies for subscribers and efficient revocation mechanisms. For instance, CloudaSec proposes the separation of subscription-based key management and confidentiality-oriented asymmetric encryption policies which enables flexible and scalable deployment of the solution as well as strong security for outsourced data in cloud servers. Our proposed public key framework applies the convergent encryption concept on data contents. That is, the data owner uploads encrypted content to the cloud and seamlessly integrates the deciphering key encrypted into the metadata to ensure data confidentiality. In addition, CloudaSec integrates a conference key distribution scheme, based on parallel Diffie Hellman exchanges, in order to guarantee backward and forward secrecy. That is, only authorized users can access metadata and decipher the decrypting data keys. As such, user revocation is achieved without updating the private keys of the remaining users.

In order to fulfill *Objective C* which consists of implementing the proposed techniques in order to validate their feasibility and impact on real hardware, experimental results have shown the efficiency of our ID-based solution and CloudaSec, while considering the impact of the cryptographic operations at the client side. Several experiments, under a simulated Openstack Swift environment, are conducted to evaluate the computation cost of some well known ID-based encryption schemes, in response to *Objective C*. Thus, our ID-based proposition presented an interesting compromise between computation cost and memory storage.

Objective B consists of addressing provable data possession in cloud storage environments for data integrity verification support. Considering three design requirements: *security level, public verifiability* and *performances*, and the limited storage capacities of user devices, we proposed two public verification schemes, as described below.

In chapter 6, we presented our third contribution [KEML14], proposing a new zero-knowledge Proof of Data Possession protocol that provides deterministic integrity verification guarantees, relying on the uniqueness of the Euclidean Division. These guarantees are considered as interesting, compared to several proposed schemes, presenting probabilistic approaches. Our proposal benefits from the elliptic curve variant of the GPS scheme advantages, namely high security level and low processing complexity. Hence, we extended the GPS scheme to the verification of the authenticity of files stored on untrusted servers. That is, the proposed solution can be performed on any connected terminal. In addition, in response to *Objective C*, our proposal is deliberately designed to support public verifiability and constant communication and storage cost. Thus, we implemented a proof of concept based on the Openstack swift service to demonstrate the feasibility of our proposal and give support to our previous theoretical performance measurements.

In chapter 7, we introduced SHoPS, a Set-Homomorphic Proof of Data Possession scheme, supporting the 3 levels of data verification, in response to *Objective B*. SHoPS allows an implementation of remote data checking at the three networking interfaces, namely, the client-CSP interface, the CSP-storing nodes interface and between two CSPs interface. This ensures the flexibility of SHoPS application and enables fulfilling each verifier request. SHoPS enables a verifier not only to obtain a proof of possession from the remote server, but also to verify that a given data file is distributed across multiple storage devices to achieve a certain desired level of fault tolerance. Indeed, we presented the set homomorphism property, which extends malleability to set operations properties, such as union, intersection and inclusion. Thus, SHoPS has been shown as a promising solution, providing an interesting compromise between computation cost and energy efficiency. SHoPS saves energy within the CSP by distributing the computation over the multiple nodes. In fact, each node provides proofs of local data block sets. This is to make applicable, a resulting proof over sets of data blocks, satisfying several needs, such as, proofs aggregation. Thus, in response to *Objective C*, SHoPS has proven to provide acceptable computation overhead, mainly at the server side, while delegating the verification process to several storing nodes.

Objective D consists of providing mathematical proofs of soundness and correctness of the proposed mechanisms. In order to fulfill this fourth objective, we evaluated the correctness of each proposed mechanism, throughout this dissertation. First, we demonstrated the correctness of CloudaSec key decryption procedures, for *one to one* and *one to many* sharing scenarios. Second, we showed the correctness of public and private verification of our zero-knowledge schemes, and SHoPS.
In addition, based on a data possession game, our zero knowledge PDP scheme, as well as SHoPS, are shown to resist to data leakage attacks, while considering either a fraudulent prover or a cheating verifier. Proofs are presented, relying on multiple interactions between a cloud service provider and a public verifier. The soundness and completeness of the proposed methods are evaluated, upon the hardness of several computational algorithms, and have shown that it is likely impossible to deduce the extra-information from the outsourced data file.

Perspectives

Regarding the performance and the security of the proposed schemes in order to ensure data confidentiality and data integrity, several security issues are still being analyzed, under untrusted cloud providers assumption. This security issues may also support an economic security model, where cloud providers try to minimize their costs and neglect implementing the needed security mechanisms. Therefore, it would be important to motivate commercial providers to properly protect client data. As such, future perspectives of interest include:

- investigate the ID based solution adaptability for asymmetric key encryption procedures, defined by the CloudaSec Framework. That is, our ID-based contribution ensures secrecy of outsourced data and a controlled access to data, relying on personalized keys. However, it presents a heavy data encryption cost. As such, it may be a good alternative to restrict ID-based to enciphering of the data deciphering key.

- validation and evaluation of CloudaSec and SHoPS mathematically using provable security and the random oracle model. That is, it would be interesting to prove our protocols' security against specific adversaries, such as chosen plaintext and chosen ciphertext adversaries. The advantage of provable security is that it gives a mathematical boundary to the probability of performing these kinds of attacks successfully.
- with the prevalence of wireless communication, the mobile devices (i.e; smart phones, tablets, $\cdots$) are emerging and start sharing the benefits of on demand cloud data storage services. Due to their constrained storage and computation capacities, it would be interesting to study lightweight data security mechanisms. Indeed, our zero knowledge proof of data possession protocol deserves to be implemented on real mobile hardware to evaluate the computation overhead. These tests will also serve to predict the power efficiency of mobile devices, while implementing the proposed cryptographic functions of our zero-knowledge protocol.
- evaluation of CloudaSec communication cost, while varying the parameters affecting the bandwidth consumption, such as the locations of end users, and the locations of storage capacities. As such, we may store data contents in different cloud data centers. As an example, we store data files in two Amazon S3 data centers: North California, USA and Paris, France. We use a customized client to continuously request files, in order to evaluate the impact of cloud users location at CloudaSec sharing scenarios.
- implementation and performance evaluation of SHoPS in a cloud of clouds environment. We have shown the effectiveness of SHoPS within the same provider. Consequently, it would be interesting to evaluate its performances, while storing data contents with other providers. As such, SHoPS would be an interesting alternative for many providers which prefer storing data on other sites.
- taking into consideration latest data confidentiality disclosure attacks, it is important to investigate, in depth, authentication mechanisms under a zero knowledge interactive proof framework. That is, zero knowledge proofs join randomness into exchanged messages. As such, for each authentication session, the cloud server and the client generate new random values, thus making messages personalized. For instance, the randomness provided by the storage server's response ensures preservation of data privacy.

To conclude, this research has been an opportunity to investigate a wide variety of concepts, models and technologies in the information and network security fields. Our objective was to investigate cloud data security concerns, while focusing on data confidentiality and remote data integrity verification challenges. We provided novel cryptographic approaches in response to the aforementioned issues, and we have shown that our proposed work is an efficient and encouraging research field.
Finally, we believe that cloud data storage security is still full of challenges and of paramount importance, and many research problems remain to be identified and investigated.

Glossary of Acronyms

ABC	Attribute Based Cryptography
ABE	Attribute Based Encryption
API	Application Programming Interface
AES	Advanced Encryption Standard
BDH	Bilinear Diffie-Hellman
BGW	Boneh-Gentry-Waters
CA	Certification Authority
CCA	Chosen Ciphertext Attack
CapEx	Capital Expenditure
CPA	Chosen Plaintext Attack
CDH	Computational Diffie Hellman
CDMI	Cloud Data Management Interface
CP-ABE	Ciphertext Policy Attribute Based Encryption
CSP	Cloud Service Provider
DDH	Decisional Diffie Hellman
DES	Data Encryption Standard
DH	Diffie Hellman
DHE	Diffie-Hellman Exponent
DHP	Diffie Hellman Problem
DLA	Data Leakage Attack
DLP	Discrete Logarithm Problem
DOS	Denial Of Service
DPDP	Dynamic Provable Data Possession
EC	Elliptic Curve
EC2	Amazon Elastic Compute Cloud
ECC	Elliptic Curve Cryptography
ECDLP	Elliptic Curve Discrete Logarithm Problem

ED	Euclidean Division
EHR	Electronic Health Record
FHE	Fully Homomorphic Encryption
GDPR	General Data Protection Regulation
GKD	Group Key Distribution
GM	Group Manager
GPS	Girault, Poupard, and Stern
HIPAA	Health Insurance Portability and Accountability Act
HAIL	High Availability and Integrity Layer
IaaS	Infrastructure as a Service
IBC	Identity Based Cryptography
IBC-PE	ID-Based Cryptography Public Elements
IBE	Identity Based Encryption
IDC	International Data Corporation
IND	Indistinguishability
IND-CCA	IND-Chosen Ciphertext Attack
IND-CPA	IND-Chosen Plaintext Attack
IPS	Interactive Proof System
IT	Information Technology
KEA	Key Escrow Attack
KP-ABE	Key Policy-Attribute Based Encryption
MAC	Message Authentication Code
MD	Meta Data
MIM	Man In the Middle
MT	Merkle-based Tree
NIST	National Institute of Standards and Technology
NSA	National Security Agency
OpEx	Operational Expenditure
OS	Operating System
PaaS	Platform as a Service
PBC	Pairing Based Cryptography
PC	Portable Computer
PDP	Proof of Data Possession
PKC	Private Key Cryptography
PKG	Private Key Generator
PKI	Public Key Infrastructure
PoR	Proof of data Retrievability
PoW	Proof of Ownership
PPT	Probabilistic Polynomial Time

PRF	Pseudorandom Function
PRISM	Planning tool for Resource Integration, Synchronization, and Management
PVE	Public Verification Elements
PwC	PriceWaterhouseCoopers
q-DHE	q-Diffie-Hellman Exponent Problem
RAFT	Remote Assessment of Fault Tolerance
RBAC	Role Based Access Control
RSA	Rivest, Shamir and Adelman
S3	Amazon Simple Storage Service
SaaS	Software as a Service
SecaaS	Security as a Service
SHoPS	Set-Homomorphic Proof of Data Possession Scheme
SLA	Service Level Agreement
SPRNG	Secure Pseudo Random Number Generator
SR-ORAM	Single Round Oblivious Random Access Memory
TLS	Transport Layer Security protocol
TPA	Third Party Auditor
URI	Uniform Resource Identifier
VM	Virtual Machine
ZKIPS	Zero-Knowledge Interactive Proof System
ZKP	Zero-Knowledge Proof

Publications

- **Nesrine Kaaniche**, Aymen Boudguiga, Maryline Laurent, *ID-Based Cryptography for Secure Cloud Data Storage*, IEEE International Conference on Cloud Computing (IEEE Cloud 2013), Santa Clara, Juin 2013.
- **Nesrine Kaaniche**, Ethmane El Moustaine, Maryline Laurent, *A Novel Zero-Knowledge Scheme for Proof of Data Possession in Cloud Storage Applications*, 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID 2014), Chicago, May 2014.
- **Nesrine Kaaniche**, Maryline Laurent, *A Secure Client Side Deduplication Scheme in Cloud Storage Environments*, 6^{th} International Conference on New Technologies, Mobility and Security (NTMS 2014), Dubai, UAE, Mars 30 – April 2, 2014.
- **Nesrine Kaaniche**, Maryline Laurent, Mohammed El Barbori, *CloudaSec: A Novel Public-Key based Framework to handle Data Storage Security in Cloud*, 11th International Conference on Security and Cryptography (SECRYPT 2014), Vienna, August 2014, **Best Paper Award**.
- **Nesrine Kaaniche**, Maryline Laurent, *SHoPS: Set Homomorphic Proof of Data Possession Scheme in Cloud Storage Applications*, {under submission}

Bibliography

[ABC$^+$07] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song. Provable data possession at untrusted stores. In *Proceedings of the 14th ACM conference on Computer and communications security*, CCS '07, New York, NY, USA, 2007. ACM.

[ABC$^+$11] G. Ateniese, R. Burns, R. Curtmola, J. Herring, O. Khan, L. Kissner, Z. Peterson, and D. Song. Remote data checking using provable data possession. *ACM Trans. Inf. Syst. Secur.*, 14(1):12:1–12:34, June 2011.

[ADPMT08] G. Ateniese, R. Di Pietro, L.V. Mancini, and G. Tsudik. Scalable and efficient provable data possession. In *Proceedings of the 4th International Conference on Security and Privacy in Communication Netowrks*, SecureComm '08, New York, NY, USA, 2008.

[AF12] D. Abts and B. Felderman. A guided tour through data-center networking. *Queue*, 10(5):10:10–10:23, May 2012.

[AFGH] G. Ateniese, K. Fu, M. Green, and S. Hohenberger. Improved proxy re-encryption schemes with applications to secure distributed storage. *ACM Trans. Inf. Syst. Secur.*, 9:1–30.

[AGJ$^+$08] R. Attebury, J. George, C. Judd, B. Marcum, and N. Montgomery. Google docs: a review. *Against the Grain*, 20(2):14–17, 2008.

[AHFG10] A. Aviram, S. Hu, B. Ford, and R. Gummadi. Determinating timing channels in compute clouds. In *Proceedings of the 2010 ACM Workshop on Cloud Computing Security Workshop*, CCSW '10, pages 103–108, New York, NY, USA, 2010. ACM.

[Ama] Amazon. Amazon simple storage service (amazon s3).

[BB04] D. Boneh and X. Boyen. Efficient selective-id secure identity-based encryption without random oracles, 2004.

[BB06] D. Boneh and X. Boyen. On the impossibility of efficiently combining collision resistant hash functions. In *In Proc. Crypto '06*, pages 570–583, 2006.

[BD05] M. Burmester and Y. Desmedt. A secure and scalable group key exchange system. *Inf. Process. Lett.*, 94(3), May 2005.

[Ben07] L. Ben. On the implementation of pairing-based cryptosystems, 2007.

[BF01] D. Boneh and m. Franklin. Identity-based encryption from the weil pairing. In *Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '01, pages 213–229, London, UK, UK, 2001. Springer-Verlag.

[BGDM+10] J.L. Beuchat, J.E. González-Díaz, S. Mitsunari, E. Okamoto, F. Rodríguez-Henríquez, and T. Teruya. High-speed software implementation of the optimal ate pairing over barreto-naehrig curves. In *Proceedings of the 4th international conference on Pairing-based cryptography*, Pairing'10, pages 21–39, Berlin, Heidelberg, 2010. Springer-Verlag.

[BGHS07] P.S. Barreto, S.D. Galbraith, C. Héigeartaigh, and M. Scott. Efficient pairing computation on supersingular abelian varieties. *Des. Codes Cryptography*, 42(3), March 2007.

[BGN05] D. Boneh, E. Goh, and K. Nissim. Evaluating 2-dnf formulas on ciphertexts. In *Proceedings of the Second international conference on Theory of Cryptography*, TCC'05, pages 325–341, Berlin, Heidelberg, 2005. Springer-Verlag.

[BGW05] D. Boneh, C. Gentry, and B. Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. In *Proceedings of the 25th Annual International Conference on Advances in Cryptology*, CRYPTO'05, pages 258–275, Berlin, Heidelberg, 2005. Springer-Verlag.

[BJO09] K.D. Bowers, A. Juels, and A. Oprea. Hail: A high-availability and integrity layer for cloud storage. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, New York, NY, USA, 2009.

[BLS01] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the weil pairing. In *Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, ASIACRYPT '01, pages 514–532, London, UK, UK, 2001. Springer-Verlag.

[Bon98] D. Boneh. The decision diffie-hellman problem. In *Proceedings of the Third International Symposium on Algorithmic Number Theory*, ANTS-III, pages 48–63, London, UK, UK, 1998. Springer-Verlag.

[BSPN04] F. Bunn, N. Simpson, R. Peglar, and G. Nagle. *SNIA Technical Tutorial: Storage Virtualization.* Storage Networking Industry Association (SNIA), 2004.

[BSSC05] I. Blake, G. Seroussi, N. Smart, and J. W. S. Cassels. *Advances in Elliptic Curve Cryptography (London Mathematical Society Lecture Note Series).* Cambridge University Press, New York, NY, USA, 2005.

[BSW07] J. Bethencourt, A. Sahai, and B. Waters. Ciphertext-policy attribute-based encryption. In *Proceedings of the 2007 IEEE Symposium on Security and Privacy*, SP '07, pages 321–334, Washington, DC, USA, 2007. IEEE Computer Society.

[BVDJ$^+$11] D. Bowers, M. Van Dijk, A. Juels, A. Oprea, and Ronald L. Rivest. How to tell if your cloud files are vulnerable to drive crashes. In *Proceedings of the 18th ACM conference on Computer and communications security*, CCS '11, New York, NY, USA, 2011.

[CA10] M. Cafaro and G. Aloisio. *Grids, Clouds and Virtualization.* Springer Publishing Company, Incorporated, 1st edition, 2010.

[CC09] Melissa Chase and Sherman S.M. Chow. Improving privacy and security in multi-authority attribute-based encryption. In *Proceedings of the 16th ACM conference on Computer and communications security*, CCS '09, pages 121–130, New York, NY, USA, 2009. ACM.

[CCMLS06] L. Chen, Z. Cheng, J. Malone-Lee, and N. Smart. Efficient id-kem based on the sakai-kasahara key construction, 2006.

[CGJ$^+$09] R. Chow, P. Golle, M. Jakobsson, E. Shi, J. Staddon, R. Masuoka, and J. Molina. Controlling data in the cloud: outsourcing computation without outsourcing control. In *Proceedings of the 2009 ACM workshop on Cloud computing security*, pages 85–90. ACM, 2009.

[chaa] http://blog.skyhighnetworks.com/only-1-in-100-cloud-providers-meet-proposed-eu-data-protection-requirements/.

[chab] http://europa.eu/rapid/press-release_memo-14-186_fr.htm.

[chac] https://github.com/openstack/swift.

[chad] https://www.dropbox.com/.

[chae] http://www.keylength.com/en/4/.

[chaf] http://www.salesforce.com/.

[Cha10] D. Chappell. Introducing the Windows azure platform. *October*, 30(October):2010, 2010.

[CKB08] R. Curtmola, O. Khan, and R. Burns. Robust remote data checking. In *Proceedings of the 4th ACM International Workshop on Storage Security and Survivability*, StorageSS '08, pages 63–68, New York, NY, USA, 2008. ACM.

[Cla05] T. Clark. *Storage Virtualization: Technologies for Simplifying Data Storage and Management.* Addison-Wesley Professional, 2005.

[CM05] Y. Chang and M. Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In *Proceedings of the Third international conference on Applied Cryptography and Network Security*, ACNS'05, Berlin, Heidelberg, 2005. Springer-Verlag.

[Coh00] P.M. Cohn. *Introduction to Ring Theory.* Springer Undergraduate Mathematics Series. Springer, 2000.

[DH76] W. Diffie and M. Hellman. New directions in cryptography, 1976.

[DPS12] R. Di Pietro and A. Sorniotti. Boosting efficiency and security in proof of ownership for deduplication. In *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '12, pages 81–82, New York, NY, USA, 2012. ACM.

[DR08] T. Dierks and E. Rescorla. RFC 5246 - The Transport Layer Security (TLS) Protocol Version 1.2. Technical report, August 2008.

[Dut08] M. Dutch. Understanding data deduplication ratios. SNIA White Paper, June 2008.

[DVW09] Y. Dodis, S. Vadhan, and D. Wichs. Proofs of retrievability via hardness amplification. In *Proceedings of the 6th Theory of Cryptography Conference on Theory of Cryptography*, TCC '09, pages 109–127, Berlin, Heidelberg, 2009. Springer-Verlag.

[ea02] Torbjorn Granlund et al. GNU multiple precision arithmetic library 4.1.2, December 2002.

[EKPT09] C. Erway, A. Küpçü, C. Papamanthou, and R. Tamassia. Dynamic provable data possession. In *Proceedings of the 16th ACM conference on Computer and communications security*, CCS '09, New York, NY, USA, 2009.

[Elg85] T Elgamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *Information Theory, IEEE Transactions on*, 31(4):469–472, 1985.

[FIP01] N FIPS. 197: Announcing the advanced encryption standard (aes). ... *Technology Laboratory, National Institute of Standards ...*, 2009(12):8–12, 2001.

[Fos02] Foster, I. What is the Grid? - a three point checklist. *GRIDtoday*, 1(6), jul 2002.

[Fos09] Foster, I. and Zhao, Y. and Raicu, I. and Lu, S. Cloud Computing and Grid Computing 360-Degree Compared. *CoRR*, abs/0901.0131, 2009.

[Fug12] S. Fugkeaw. Achieving privacy and security in multi-owner data outsourcing. pages 239–244. IEEE, 2012.

[Gen09] C. Gentry. *A fully homomorphic encryption scheme.* PhD thesis, Stanford, CA, USA, 2009. AAI3382729.

[GM84] S. Goldwasser and S. Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, pages 270–299, 1984.

[GMR85] S Goldwasser, S Micali, and C Rackoff. The knowledge complexity of interactive proof-systems. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, STOC '85, pages 291–304, New York, NY, USA, 1985. ACM.

[Gol00] O. Goldreich. *Foundations of Cryptography: Basic Tools.* Cambridge University Press, New York, NY, USA, 2000.

[Goo11] Google. Google app engine. *Development*, 2009(28th April 2010):1–10, 2011.

[GPea06] Vipul Goyal, Omkant Pandey, and et al. Attribute-based encryption for fine-grained access control of encypted data, 2006.

[GPS06] M. Girault, G. Poupard, and J. Stern. On the fly authentication and signature schemes based on groups of unknown order. *Journal of Cryptology*, 19:463–487, 2006.

[GPS08] S.D. Galbraith, K.G. Paterson, and N.P. Smart. Pairings for cryptographers, 2008. Applications of Algebra to Cryptography.

[GR12] J. Gantz and D. Reinsel. The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east. *IDC iView*, (December):1–16, 2012.

[Grä78] G. Grätzer. *General lattice theory*, volume 75 of *Pure and Applied Mathematics.* Academic Press Inc. [Harcourt Brace Jovanovich Publishers], New York, 1978.

[hbndcfcs] http://www.datacenterknowledge.com/archives/2013/01/18/facebook-builds-new-data-centers-for-cold storage/.

[HHPSP11] S. Halevi, D. Harnik, B. Pinkas, and A. Shulman-Peleg. Proofs of ownership in remote storage systems. In *Proceedings of the 18th ACM conference on Computer and communications security*, CCS '11, pages 491–500, New York, NY, USA, 2011. ACM.

[HMV03] D. Hankerson, A. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003.

[HPSP10] D. Harnik, B. Pinkas, and A. Shulman-Peleg. Side channels in cloud services: Deduplication in cloud storage. *IEEE Security And Privacy*, 8(6):40–47, 2010.

[hSIHA] http://www.storiant.com/resources/Cold Storage-Is-Hot-Again.pdf.

[htta] http://aws.amazon.com/glacier/.

[httb] https://downloads.cloudsecurityalliance.org/initiatives/.

[IAP09] L. Ibraimi, M. Asim, and M. Petkovic. Secure management of personal health records by applying attribute-based encryption, July 2009.

[Inc08] Amazon Inc. *Amazon Elastic Compute Cloud (Amazon EC2).* Amazon Inc., http://aws.amazon.com/ec2/#pricing, 2008.

[IPN$^+$09] L. Ibraimi, M. Petkovic, S. Nikova, P. Hartel, and W. Jonker. Ciphertext-policy attribute-based threshold decryption with flexible delegation and revocation of user attributes (extended version), April 2009.

[ISO89] ISO/IEC. Iso 7498-2:1989, information processing systems - open systems interconnection - basic reference model - part 2: Security architecture. *Information Processing Systems – Open Systems Interconnection – Basic Reference Model*, 1989.

[JK07] A. Juels and B. Kaliski. Pors: proofs of retrievability for large files. In *In CCS '07: Proceedings of the 14th ACM conference on Computer and communications security*, pages 584–597. ACM, 2007.

[JMXSW02] D. Johnson, D. Molnar, D. Xiaodong Song, and D. Wagner. Homomorphic signature schemes. In *CT-RSA*, pages 244–262, 2002.

[KBL13] N. Kaaniche, A. Boudguiga, and M. Laurent. ID based cryptography for cloud data storage. In *2013 IEEE Sixth International Conference on Cloud Computing, Santa Clara, CA, USA, June 28 - July 3, 2013*, pages 375–382, 2013.

[KEML14] N. Kaaniche, E. El Moustaine, and M. Laurent. A novel zero-knowledge scheme for proof of data possession in cloud storage applications. In *2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, Chicago, IL, USA, May 26-29, 2014*, pages 522–531, 2014.

[Ken] W.R. Kenneth. Minimum-complexity pairing functions.

[KK12] L. Krzywiecki and M. Kutylowski. Proof of possession for cloud storage via lagrangian interpolation techniques. In *Proceedings of the 6th international conference on Network and System Security*, NSS'12, Berlin, Heidelberg, 2012. Springer-Verlag.

[KL10] S. Kamara and K. Lauter. Cryptographic cloud storage. In *Proceedings of the 14th international conference on Financial cryptograpy and data security*, FC'10, Berlin, Heidelberg, 2010. Springer-Verlag.

[KL14] N. Kaaniche and M. Laurent. A secure client side deduplication scheme in cloud storage environments. In *6th International Conference on New Technologies, Mobility and Security, NTMS 2014, Dubai, United Arab Emirates, March 30 - April 2, 2014*, pages 1–7, 2014.

[KLEB14] N. Kaaniche, M. Laurent, and M. El Barbori. Cloudasec: A novel public-key based framework to handle data sharing security in clouds. In *ICETE – 11th International Conference on Security and Cryptography*, August 28 - 30, 2014.

[Kob87] N. Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48(177):203–209, January 1987.

[LCH13] C. Lee, P. Chung, and M. Hwang. A survey on attribute-based encryption schemes of access control in cloud environments. *I. J. Network Security*, 15(4):231–240, 2013.

[LDTY09] Hongwei Li, Yuanshun Dai, Ling Tian, and Haomiao Yang. Identity-based authentication for cloud computing. In *Proceedings of the 1st International Conference on Cloud Computing*, CloudCom '09, pages 157–166, Berlin, Heidelberg, 2009. Springer-Verlag.

[LP11] Hoon Wei Lim and Kenneth G. Paterson. Identity-based cryptography for grid security. *Int. J. Inf. Secur.*, 10(1):15–32, February 2011.

[LR04] H W Lim and M J B Robshaw. On identity-based cryptography and grid computing. *Lecture Notes in Computer Science*, pages 474–477, 2004.

[LR05] H W Lim and M J B Robshaw. A dynamic key infrastructure for grid. In *Proceedings of the 2005 European conference on Advances in Grid Computing*, EGC'05, pages 255–264, Berlin, Heidelberg, 2005. Springer-Verlag.

[LZWY13] X. Liu, Y. Zhang, B. Wang, and J. Yan. Mona: Secure multi-owner data sharing for dynamic groups in the cloud. *IEEE Trans. Parallel Distrib. Syst.*, 24(6), June 2013.

[MD11] D. McEwen and H. Dumpel. Hipaa—the health insurance portability and accountability act. *National Nurse (NATL NURSE)*, (July/August):20–27, 2011.

[Mer88] Ralph C. Merkle. A digital signature based on a conventional encryption function. In *A Conference on the Theory and Applications of Cryptographic Techniques on Advances in Cryptology*, CRYPTO '87, pages 369–378, London, UK, UK, 1988. Springer-Verlag.

[Mil86] Victor S. Miller. Use of elliptic curves in cryptography. In *Advances in Cryptology*, CRYPTO '85, pages 417–426, London, UK, UK, 1986. Springer-Verlag.

[MR14] Kurra Mallaiah and S. Ramachandram. Article: Applicability of homomorphic encryption and cryptdb in social and business applications: Securing data stored on the third party servers while processing through applications. *International Journal of Computer Applications*, 100(1):5–19, August 2014. Full text available.

[MRS88] S. Micali, C. Rackoff, and B. Sloan. The notion of security for probabilistic cryptosystems. *SIAM J. Comput.*, 17(2):412–426, April 1988.

[MRSP12] V. Moreno, E. Roberts, S. Sane, and M. Portolani. *Data Center Networking for Cloud Computing Environments.* WebEx Communications, 1st edition, 2012.

[MVO96] Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. *Handbook of Applied Cryptography.* CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 1996.

[NF07] Hanne Riis Nielson and Gilberto Filé, editors. *Static Analysis, 14th International Symposium, SAS 2007, Kongens Lyngby, Denmark, August 22-24, 2007, Proceedings*, volume 4634 of *Lecture Notes in Computer Science.* Springer, 2007.

[NIS99] National Bureau Of Standards NIST. Data encryption standard (des). *Technology*, 46-3(46):1–26, 1999.

[NLV11] M. Naehrig, K. Lauter, and V. Vaikuntanathan. Can homomorphic encryption be practical? In *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop*, CCSW '11, New York, NY, USA, 2011. ACM.

[nsaa] http://www.darkreading.com/cloud-security/nsa-surveillance-first-prism-now-muscled-out-of-cloud/d/d-id/1112686.

[nsab] http://www.zdnet.com/nsa-spying-poisons-the-cloud-market-survey-7000022964/.

[NWZ12] W.K. Ng, Y. Wen, and H. Zhu. Private data deduplication protocols in cloud storage. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, SAC '12, pages 441–446, New York, NY, USA, 2012. ACM.

[Pai99] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Proceedings of the 17th international conference on Theory and application of cryptographic techniques*, EUROCRYPT'99, pages 223–238, Berlin, Heidelberg, 1999. Springer-Verlag.

[Ped92] T.P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '91, pages 129–140, London, UK, UK, 1992. Springer-Verlag.

[PT09] M. Peter and G. Tim. The NIST Definition of Cloud Computing. *National Institute of Standards and Technology*, 53(6):50, 2009.

[QQQ$^+$89] Jean-Jacques Quisquater, Myriam Quisquater, Muriel Quisquater, Michaël Quisquater, Louis C. Guillou, Marie Annick Guillou, Gaïd Guillou, Anna Guillou, Gwenolé Guillou, Soazig Guillou, and Thomas A. Berson. How to explain zero-knowledge protocols to your children. In *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 628–631. Springer, 1989.

[RAD78] R.L. Rivest, L. Adleman, and M.L. Dertouzos. On data banks and privacy homomorphisms. In *Foundations on Secure Computation, Academia Press*, pages 169–179, 1978.

[RRP04] D. Ratna, B. Rana, and S. Palash. Pairing-based cryptographic protocols : A survey. 2004. http://eprint.iacr.org/.

[RSA78] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[RTSS09] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage. Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, pages 199–212, New York, NY, USA, 2009. ACM.

[SBW11] S. Setty, A.J. Blumberg, and M. Walfish. Toward practical and unconditional verification of remote computations. In *Proceedings of the 13th USENIX Conference on Hot Topics in Operating Systems*, HotOS'13, Berkeley, CA, USA, 2011. USENIX Association.

[SDJ$^+$10] C. Schridde, T. Dörnemann, E. Juhnke, M. Smith, and B. Freisleben. An identity-based security infrastructure for cloud environments. In *Proc. of IEEE International Conference on Wireless Communications, Networking and Information Security (WCNIS2010)*, 2010.

[SGLM08] M. Storer, K. Greenan, D.D.E. Long, and E.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.

[Sha49] Claude E. Shannon. Communication Theory of Secrecy Systems. *Bell Systems Technical Journal*, 28:656–715, 1949.

[Sha85] A. Shamir. Identity-based cryptosystems and signature schemes. In *Proceedings of CRYPTO 84 on Advances in cryptology*, pages 47–53, New York, NY, USA, 1985. Springer-Verlag New York, Inc.

[Sho97] V. Shoup. Lower bounds for discrete logarithms and related problems. In *Proceedings of the 16th Annual International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT'97, pages 256–266, Berlin, Heidelberg, 1997. Springer-Verlag.

[SK03] R. Sakai and M. Kasahara. Id based cryptosystems with pairing on elliptic curve. Cryptology ePrint Archive, Report 2003/054, 2003.

[SNDB13] S.H. Seo, M. Nabeel, X. Ding, and E. Bertino. An efficient certificateless encryption for secure data sharing in public clouds. *IEEE Transactions on Knowledge and Data Engineering*, 99:1, 2013.

[Sni10] The Snia. Cloud data management interface. *Representations*, pages 1–173, 2010.

[Sta02] United States. *A report to Congress in accordance with [section] 326(b) of the Uniting and Strengthening America by Providing Appropriate Tools Required to Intercept and Obstruct Terrorism Act of 2001 (USA PATRIOT ACT) [electronic resource] / submitted by the Department of the Treasury.* Dept. of the Treasury [Washington, D.C.], 2002.

[SV10] N. P. Smart and F. Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In *Proceedings of the 13th International Conference on Practice and Theory in Public Key Cryptography*, PKC'10, pages 420–443, Berlin, Heidelberg, 2010. Springer-Verlag.

[SW05] A. Sahai and B. Waters. Fuzzy identity-based encryption. In *Proceedings of the 24th Annual International Conference on Theory and Applications of Cryptographic Techniques*, EUROCRYPT'05, pages 457–473, Berlin, Heidelberg, 2005. Springer-Verlag.

[SW08] H. Shacham and B. Waters. Compact proofs of retrievability. In *Proceedings of the 14th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, ASIACRYPT '08, Berlin, Heidelberg, 2008. Springer-Verlag.

[The03] The OpenSSL Project. April 2003.

[VDHV10] C. Van Dijk, M.and Gentry, S. Halevi, and V. Vaikuntanathan. Fully homomorphic encryption over the integers. In *Proceedings of the 29th Annual International Conference on Theory and Applications of Cryptographic Techniques*, EUROCRYPT'10, pages 24–43, Berlin, Heidelberg, 2010. Springer-Verlag.

[VDJO$^+$12] M. Van Dijk, A. Juels, A. Oprea, Ronald L. Rivest, E. Stefanov, and N. Triandopoulos. Hourglass schemes: how to prove that cloud files are encrypted. In *Proceedings of the 2012 ACM conference on Computer and communications security*, CCS '12, New York, NY, USA, 2012.

[Ver19] G S Vernam. Secret signaling system, 1919.

[WQPW10] C. Wang, Z. Qin, J. Peng, and J. Wang. A novel encryption scheme for data deduplication system. pages 265–269, 2010.

[WRW11] C. Wang, K. Ren, and J. Wang. Secure and practical outsourcing of linear programming in cloud computing. In *INFOCOM, 2011 Proceedings IEEE*, pages 820–828, April 2011.

[WS12] P. Williams and R. Sion. Single round access privacy on outsourced storage. In *Proceedings of the 2012 ACM conference on Computer and communications security*, CCS '12, pages 293–304, New York, NY, USA, 2012. ACM.

[WWL+] Q. Wang, C. Wang, J. Li, K. Ren, and W. Lou. Enabling public verifiability and data dynamics for storage security in cloud computing. In *Proceedings of the 14th European Conference on Research in Computer Security*, ESORICS'09, Berlin, Heidelberg. Springer-Verlag.

[WWRL10] C. Wang, Q. Wang, K. Ren, and W. Lou. Privacy-preserving public auditing for data storage security in cloud computing. In *Proceedings of the 29th Conference on Information Communications*, INFOCOM'10, pages 525–533, Piscataway, NJ, USA, 2010. IEEE Press.

[XC12] J. Xu and E. Chang. Towards efficient proofs of retrievability. In *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '12, New York, NY, USA, 2012.

[XCZ13] J. Xu, E. Chang, and J. Zhou. Weak leakage-resilient client-side deduplication of encrypted data in cloud storage. In *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*, ASIA CCS '13, pages 195–206, New York, NY, USA, 2013. ACM.

[XX12] Z. Xiao and Y. Xiao. Security and privacy in cloud computing. *Communications Surveys Tutorials, IEEE*, PP(99):1 –17, 2012.

[XZY+12] H. Xiong, X. Zhang, D. Yao, X. Wu, and Y. Wen. Towards end-to-end secure content storage and delivery with public cloud. CODASPY '12, pages 257–266. ACM, 2012.

[YPX05] A. Yaar, A Perrig, and D. Xiaodong. Fit: fast internet traceback. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies, 13-17 March 2005, Miami, FL, USA*, pages 1395–1406. IEEE, 2005.

[YWRL10a] Shucheng Yu, Cong Wang, Kui Ren, and Wenjing Lou. Achieving secure, scalable, and fine-grained data access control in cloud computing. In *Proceedings of the 29th conference on Information communications*, INFOCOM'10, pages 534–542, Piscataway, NJ, USA, 2010. IEEE Press.

[YWRL10b] Shucheng Yu, Cong Wang, Kui Ren, and Wenjing Lou. Attribute based data sharing with attribute revocation. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '10, pages 261–270, New York, NY, USA, 2010. ACM.

[Zun12] K. Zunnurhain. Fapa: a model to prevent flooding attacks in clouds. In *Proceedings of the 50th Annual Southeast Regional Conference*, ACM-SE '12, pages 395–396, New York, NY, USA, 2012. ACM.

[ZVH11] L. Zhou, V. Varadharajan, and M. Hitchens. Enforcing role-based access control for secure data storage in the cloud. *Comput. J.*, 54, October 2011.

[ZWH$^+$11] Y. Zhu, H. Wang, Z. Hu, G. Ahn, H. Hu, and S.S. Yau. Dynamic audit services for integrity verification of outsourced storages in clouds. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, SAC '11, New York, NY, USA, 2011.

[ZYG11] S. Zarandioon, D. Yao, and V. Ganapathy. K2c: Cryptographic cloud storage with lazy revocation and anonymous access. In *SecureComm*, volume 96, pages 59–76. Springer, 2011.

Appendix A

French Summary

Au cours de la dernière décennie, avec la standardisation d'Internet, le développement des réseaux à haut débit, le paiement à l'usage et la quête sociétale de la mobilité, le monde informatique a vu se populariser un nouveau paradigme, le *Cloud*. Le recours au *cloud* est de plus en plus remarquable compte tenu de plusieurs facteurs, notamment ses architectures rentables, prenant en charge la transmission, le stockage et le calcul intensif de données. Cependant, ces services de stockage prometteurs soulèvent la question de la protection des données et de la conformité aux réglementations, qui sous tend le problème de la perte de maîtrise et de gouvernance. Cette dissertation vise à surmonter ce dilemme, tout en tenant compte de deux préoccupations de sécurité des données, à savoir la confidentialité des données et l'intégrité des données.

A.1 Définitions

Le cloud, appelé aussi *informatique en nuage*, décrit des ressources évolutives fournies sous forme d'un service externe via Internet, mesuré et facturé à l'usage. Le cloud peut être défini comme étant un modèle de calcul distribué, qui est configuré dynamiquement et fourni à la demande. Ce nouveau paradigme, massivement évolutif, est différent des réseaux traditionnels. Trois niveaux de services sont offerts via le cloud:

- *Infrastructure as a Service (IaaS)* – pour ce modèle de service, les ressources sont gérées, agrégées et livrées aux utilisateurs, comme étant des capacités de stockage (par exemple, service Amazon Simple Storage (S3) [Ama]), des ressources de réseau, ou des capacités de calcul (par exemple, Amazon Elastic Compute Cloud (EC2) [Inc08]). L'utilisateur peut alors exécuter le système d'exploitation de son choix, et des logiciels qui répondent au mieux à ses besoins, sans être en mesure de gérer ou contrôler l'infrastructure cloud sous-jacente.
- *Plateform as a Service (PaaS)* – ce modèle de service offre un environnement ou une plate-forme de développement, sur laquelle les utilisateurs exécutent leurs appli-

cations. Autrement dit, le fournisseur de service met à disposition des utilisateurs une plateforme spécifique leur permettant d'exécuter des applications propriétaires. Ils ont aussi un contrôle total sur leurs applications déployées et la configuration associée.

- *Software as a Service (SaaS)* – l'utilisateur contrôle partiellement des configurations d'applications fonctionnant sur une infrastructure de Cloud, comme Google Apps. Ces applications sont accessibles à travers un grand nombre de périphériques client (téléphone portable, ordinateur, etc.) par le biais d'une interface Web.

Pour mieux comprendre les concepts de base et les technologies associés au cloud, nous extrayons du document de définition de NIST [PT09] cinq attributs, soient;

- *Des services à la demande* – l'utilisateur peut réserver sans aucune interaction humaine avec le fournisseur des capacités de traitement et de stockage selon ses besoins.
- *Un large accès réseau* – les différents services d'un fournisseur de cloud ainsi que les ressources réservées par un utilisateur sont disponibles et accessibles par le biais d'un large éventail de périphériques pouvant se connecter au réseau.
- *Des ressources partagées* – le fournisseur met en commun les ressources informatiques (stockages, mémoire, CPU, bande passante, machine virtuelle, etc.) afin de satisfaire plusieurs utilisateurs par le biais de ressources physiques et virtuelles pouvant être affectées et réaffectées en fonction de la demande des utilisateurs. Ceci induit pour les utilisateurs une méconnaissance de la localisation exacte des ressources qu'ils utilisent.
- *Un dimensionnement rapide* – les ressources peuvent être dimensionnées rapidement à la hausse ou à la baisse selon la demande de l'utilisateur.
- *Un service mesuré et facturé à l'usage* – le cloud mesure automatiquement l'utilisation des ressources (CPU, stockage, bande passante, temps d'utilisation d'une machine virtuelle, etc.) faite par un utilisateur afin de permettre de lui calculer automatiquement sa facture.

La définition du NIST révèle plusieurs contraintes de sécurité, dues aux technologies et aux caractéristiques de cet environnement abstrait. En effet, au vu de ces services taillés sur mesure, le portefeuille des services cloud est enrichi en permanence. La prolifération de ces services relèvent de nouveaux besoins de sécurité. Selon [XX12], il existe trois principaux défis qui sont étroitement liés aux caractéristiques du cloud, à savoir l'externalisation, la co-résidence des données et des processus, et le calcul massif des données.

A.2 Problématiques, Objectifs et Contributions

Les services de stockage de données cloud révèlent de nombreux problèmes de conception complexes, principalement dus à la perte du contrôle physique. Ces défis ont des conséquences significatives sur la sécurité des données et les performances des systèmes de cloud.

D'un côté, la préservation de la **confidentialité des données**, dans des environnements *multi-tenants*, devient plus difficile et conflictuelle. Ceci est largement dû au fait que les utilisateurs stockent leurs données à des serveurs distants contrôlés par des fournisseurs de service (CSP) non fiables. Il est communément admis que le chiffrement des données, du côté client, est une bonne solution garantissant la confidentialité des données [KL10, CGJ+09]. Ainsi, le client conserve les clés de déchiffrement, loin de la portée du fournisseur de cloud. Néanmoins, cette approche suscite plusieurs problèmes de gestion de clés, tels que, le stockage et le maintien de la disponibilité des clés. En outre, la préservation de la confidentialité devient plus compliquée avec un partage de données dynamique entre un groupe d'utilisateurs. Premièrement, il faut un partage efficace des clés de déchiffrement entre les différents utilisateurs autorisés. Le défi consiste à définir un mécansime efficace permettant la révocation d'un membre sans avoir besoin de mettre à jour tous les secrets des autres membres. Deuxièmement, les politiques de contrôle d'accès doivent être souples et permettent de distinguer des privilèges différents selon les utilisateurs pour accéder aux données. En effet, les données peuvent être partagées par différents utilisateurs ou groupes, et les utilisateurs peuvent appartenir à des différents groupes.

D'un autre côté, **l'intégrité des données** est considérée comme une préoccupation majeure, dans des environnements cloud. Afin de renforcer la résilience, les fournisseurs de service ont généralement recours à la distribution des fragments de données selon des politiques de réplication. Néanmoins, afin de réduire les coûts d'exploitation et de libérer des capacités de stockage, les fournisseurs malhonnêtes pourraient volontairement négliger ces procédures de réplication, entraînant des erreurs irréversibles ou même des pertes de données. Même lorsque les fournisseurs de cloud mettent en œuvre une politique de tolérance aux pannes, les clients n'ont pas les moyens techniques pour vérifier que leurs données ne sont pas vulnérables. Il pourrait y avoir des implémentations de vérification, à distance de l'intégrité et la possession des données externalisées, selon trois niveaux, comme suit:

- Entre un client et un CSP – un client devrait avoir un moyen efficace, lui permettant d'effectuer des vérifications périodiques d'intégrité à distance, sans garder les données en local. En outre, le client doit également détecter la violation du SLA, par rapport à la politique de stockage. La préoccupation de ce client est accentuée par ses capacités de stockage et de calcul limitées, et de la grande taille des données externalisées.
- Au sein du CSP – il est important pour un fournisseur de cloud de vérifier l'intégrité des blocs de données stockés sur plusieurs nœuds de stockage, afin de réduire le risque de perte de données, dû à la défaillance du matériel.
- Entre deux CSPs – dans un environnement des clouds imbriqués, où les données sont réparties sur de différentes infrastructures de cloud, un CSP, à travers sa passerelle, doit périodiquement vérifier l'authenticité de blocs de données hébergées par une autre plate-forme de service.

Ces problèmes de sécurité sont d'autant plus importants, que les règlements européens à venir prévoient des mesures de protection plus sévères et rigides, y compris de nouvelles dérogations pour protéger efficacement les données à caractère personnel qui sont externalisées sur des serveurs distants. Le réglement européen sur la protection des données (GDPR) devrait être adoptée cette année et prendre effet début 2015 [chab].

Pour relever les besoins de sécurité mentionnés ci-dessus, nous avons fixé les objectifs suivants:

- ***Objectif A*** – améliorer la confidentialité des données, tout en considérant un partage dynamique des données entre les utilisateurs.
- ***Objectif B*** – s'attaquer au problème de la preuve de possession des données dans les environnements de stockage en nuage. cette preuve vient en soutien au problème de la vérification de l'intégrité des données, et intègre les trois aspects suivants: *niveau de sécurité, vérifiabilité publique,* et *performances*, (compte tenu des capacités de stockage et de traitement limitées des dispositifs de l'utilisateur).
- ***Objectif C*** – mettre en œuvre les techniques proposées à l'aide de normes et de systèmes largement déployés, et valider leur faisabilité et leur impact sur le matériel réel.
- ***Objectif D*** – fournir des preuves mathématiques de la solidité et de l'exactitude des schémas proposés.

Lors de la définition des solutions garantissant la confidentialité et l'intégrité des données externalisées, nous avons pris en considération les aspects suivants: la facilité du déploiement, la robustesse, la flexibilité et les performances. Les contributions de cette thèse se résument à:

- *Contribution* 1 – proposition d'un schéma cryptographique pour le stockage des données sur des serveurs cloud, basé sur une utilisation originale de la Cryptographie Basée sur l'Identité (IBC). Tout d'abord, le schéma proposé assure une meilleure confidentialité des données. En effet, chaque client agit comme un générateur de clé privée (PKG), lui permettant de calculer une paire de clés basée sur un identifiant unique du fichier avant d'envoyer le contenu chiffré au cloud pour le stocker. Par conséquent, l'accès aux données est géré par le propriétaire des données. En outre, en utilisant une clé basée sur l'identifiant des données, nous proposons une approche de partage flexible. En effet, la distribution des clés de déchiffrement entre le client et les utilisateurs autorisés, ne permet pas d'extraire le secret du client (*Objectif A, Objectif C*).
- *Contribution* 2 – définition de CloudaSec, une solution à clé publique permettant d'améliorer la confidentialité des données dans les environnements de stockage en nuage, ainsi que le partage dynamique entre les utilisateurs. CloudaSec applique le chiffrement convergent sur le contenu des données, et intègre un système de distribution des clés, basé sur des échanges parallélisés Diffie Hellman [BD05], afin de garantir la propriété *backward and forward secrecy* (*Objectif A, Objectif C*).

- *Contribution* 3 – définition d'un nouveau protocole de preuve de possession, sans apport de connaissance, qui fournit des garanties déterministes de vérification d'intégrité, en s'appuyant sur l'unicité de la division euclidienne. Ces garanties sont considérées comme intéressantes par rapport à plusieurs schémas proposés de la littérature, présentant des approches probabilistes. Le protocole est adapté aux terminaux ayant des capacités de stockage limitées (*Objectif B, Objectif D*).

- *Contribution* 4 – présentation de SHoPS, un protocole de preuve de possession de données capable de traiter les trois relations d'ensembles homomorphiques. SHoPS permet au client non seulement d'obtenir une preuve de la possession des données du serveur distant, mais aussi de vérifier que le fichier, en question, est bien réparti sur plusieurs périphériques de stockage permettant d'atteindre un certain niveau de tolérance aux pannes. En effet, nous présentons l'ensemble des propriétés homomorphiques, qui étend la malléabilité du procédé aux propriétés d'union, intersection et inclusion (*Objectif B, Objectif D*).

A.3 Confidentialité des données stockées dans le Cloud

En confiant la tâche de stockage de données à une tierce partie, l'assurance de la confidentialité et la protection de la vie privée deviennent plus difficiles. La protection de la vie privée est une préoccupation majeure des utilisateurs du cloud, du fait que leurs données soient stockées dans des serveurs publiques distribués. Par conséquent, il y a un risque potentiel de divulgation des informations confidentielles (par exemple, les données financières, dossiers de santé) ou des informations personnelles (par exemple, profil personnel). Quant à la confidentialité, elle implique que les données des clients doivent être gardées secrètes et ne doivent pas être divulguées à des personnes non autorisées, à savoir le fournisseur de service ou d'autres utilisateurs malveillants.

Dans cette section, nous nous concentrons sur la confidentialité des données, un enjeu important étant donné le besoin d'assurer un partage de données flexible au sein d'un groupe dynamique d'utilisateurs. Cet enjeu exige, par conséquence, un partage efficace des clés entre les membres du groupe.
Pour répondre à cette préoccupation, nous avons, d'une part, proposé une nouvelle méthode reposant sur l'utilisation de la cryptographie basée sur l'identité, où chaque client agit comme une entité génératrice de clés privées. Ainsi, ce client génère ses propres éléments publiques et s'en sert pour calculer sa clé privée correspondante. Grâce aux propriétés d'IBC, cette contribution a démontré sa résistance face aux accès non autorisés aux données au cours du processus de partage, tout en tenant compte de deux modèles de sécurité, à savoir un serveur de stockage honnête mais curieux et un utilisateur malveillant.
D'autre part, nous avons introduit CloudaSec, une solution à base de clé publique, qui propose la séparation de la gestion des clés et les techniques de chiffrement, sur deux couches. En effet, CloudaSec permet un déploiement flexible d'un scénario de partage de données ainsi que des garanties de sécurité solides pour les données externalisées sur les serveurs du cloud. Les résultats expérimentaux, sous OpenStack Swift, ont prouvé l'efficacité de CloudaSec, compte tenu de l'impact des opérations cryptographiques sur le terminal du client.

A.3.1 Cryptographie Basée sur l'Identité pour un stockage sécurisé des données

Dans cette section, nous présentons notre première contribution [KBL13], basée sur les avantages de la Cryptographie Basée sur l'Identité (IBC) pour un stockage sécurisé de données dans le cloud [Sha85, BF01].

A.3.1.1 Architecture

Figure A.1 illustre un schéma de l'architecture d'un service de stockage en nuage. Elle s'appuie sur les entités suivantes, permettant à un client de stocker, récupérer et partager des données avec plusieurs utilisateurs:

- *fournisseur de service Cloud (CSP)* – un CSP dispose de ressources importantes pour contrôler les serveurs de stockage. Il fournit également une infrastructure virtuelle pour héberger des services d'application. Ces services peuvent être utilisés par le client pour gérer ses données stockées dans les serveurs cloud.
- *Client (C)* – un client est le propriétaire de données. Il utilise les ressources du fournisseur de service pour stocker, récupérer et partager des données avec plusieurs utilisateurs. Chaque client a une identité unique et authentique, notée ID_C.
- *utilisateurs (U)* – les utilisateurs sont en mesure d'accéder au contenu stocké dans le nuage, en fonction de leurs droits d'accès s'agissant des autorisations accordées par le client, comme les droits de lecture et d'écriture. Ces droits d'accès permettent de déterminer plusieurs groupes d'utilisateurs. Chaque groupe est caractérisé par un identifiant ID_G et un ensemble de droits d'accès.

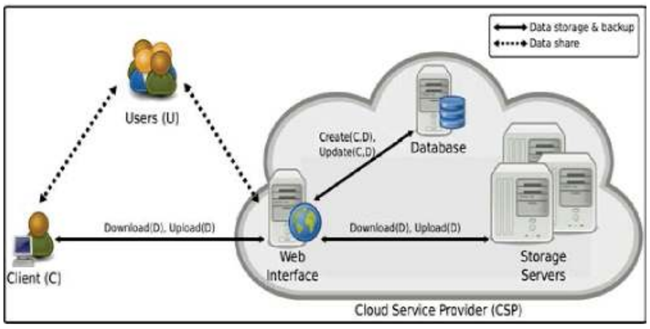

FIGURE A.1 - Architecture d'un service de stockage en nuage

Notre idée consiste à utiliser la cryptographie basée sur l'identité, pour générer une paire de clés pour chaque donnée. En fait, notre proposition bénéficie des avantages d'IBC, notamment une infrastructure sans certificat, une petite taille des clés utilisées, et une gestion légère des secrets.

Dans [KBL13], chaque client prend le rôle d'un générateur de clé privé (PKG) qui génère ses propres *ID-Based Cryptography Public Elements* (IBC–PE). Ces IBC–PE sont utilisés pour générer les clés, permettant ainsi de chiffrer les données avant de les stocker dans le cloud. Il est à noter que pour chaque donnée, le client calcule les clés publique et privée correspondantes en s'appuyant sur son secret s_C.

Le choix de l'IBC est motivé par plusieurs facteurs. Tout d'abord, nous bénéficions d'un mécanisme de gestion flexible grâce à une infrastructure sans certificat. En effet, le calcul des clés publiques, à partir des identifiants uniques de données ne nécessite pas le déploiement d'une infrastructure à clé publique (PKI) et la distribution de certificats. D'autre part, IBC permet de déduire les clés publiques sans avoir besoin d'un calcul préalable des clés privées correspondantes, contrairement aux schémas de chiffrement classiques. En effet, les utilisateurs n'ont qu'à générer les clés publiques pour chiffrer les données avant de les stocker. La génération de la clé privée sera exécutée au moment de la récupération des données. Troisièmement, IBC permet de générer une clé spécifique pour chaque donnée à partir d'un identifiant unique. La dérivation d'une clé par donnée est bien adaptée pour un processus de partage. Dans la section suivante, nous présentons les différents scénarios, pour le stockage, la récupération et le partage de données.

A.3.1.2 Scénario de stockage de données

Après s'être authentifié avec succès avec le CSP, le client démarre le processus de stockage. En effet, le client chiffre ses données à l'aide d'une clé publique, notée, Pub_D qui est dérivée à partir d'une concaténation de l'identité du client ID_C et l'identifiant des données de ID_D, comme suit:

$$Pub_D = Hash_{pub}(ID_C || ID_D) \tag{A.1}$$

ID_D est générée localement par le client et est dérivée des méta-données (MD) en utilisant une fonction de hachage $H()$ telle que $ID_D = H(MD)$.

Figure A.2 illustre les différents échanges entre le client et son fournisseur de cloud. Le scénario de stockage se base sur quatre messages:

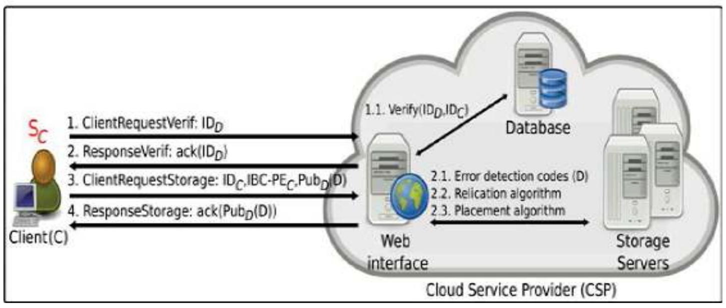
S_C
1. ClientRequestVerif: ID_D
2. ResponseVerif: ack(ID_D)
3. ClientRequestStorage: ID_C,IBC-PE_C,Pub_D(D)
4. ResponseStorage: ack(Pub_D(D))
Client(C)
1.1. Verify(ID_D,ID_C)
Database
2.1. Error detection codes (D)
2.2. Relication algorithm
2.3. Placement algorithm
Web interface
Storage Servers
Cloud Service Provider (CSP)

Figure A.2 - Scénario de stockage de données

- *ClientRequestVerif*: ce premier message contient l'identifiant de données ID_D. Ce message est une demande de vérification de l'unicité de ID_D. Le CSP répond par un message *ResponseVerif* pour valider ou rejeter l'identifiant revendiqué. Il est à noter que le processus de stockage de données doit être arrêté lorsque la vérification est échouée.
- *ResponseVerif*: ce message est un accusé de réception, généré par le CSP pour valider l'identifiant ID_D. Lors de la réception de ce message, le client concatène ID_C et ID_D pour dériver la clé publique Pub_D utilisée pour chiffrer ses données.
- *ClientRequestStorage*: ce message contient les éléments publiques générés par le client, ainsi que les données chiffrées $Pub_D(D)$.
- *ResponseStorage*: ce message est envoyé par le CSP, et est utilisé pour confirmer au client le succès de son stockage de données.

A.3.1.3 Scénario de récupération des données

La Figure A.3 présente les différents échanges entre un client et son fournisseur de service, pour récupérer ses données stockées sur des serveurs distants. Ce scénario se base sur deux messages: *ClientRequestBackup* envoyé par le client et *ResponseBackup*, envoyé par le fournisseur de cloud.

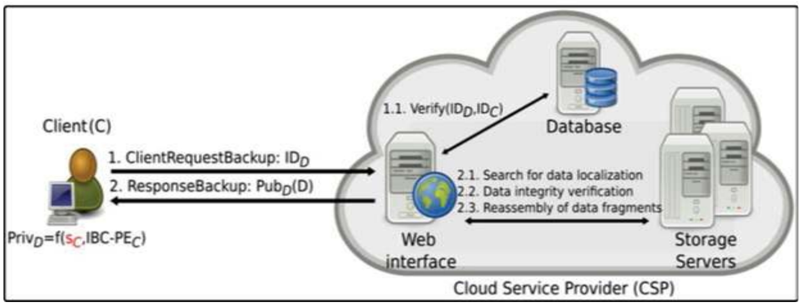
Client(C)
1. ClientRequestBackup: ID_D
2. ResponseBackup: $Pub_D(D)$
$Priv_D=f(s_C,IBC\text{-}PE_C)$
1.1. Verify(ID_D,ID_C)
Database
2.1. Search for data localization
2.2. Data integrity verification
2.3. Reassembly of data fragments
Web interface
Storage Servers
Cloud Service Provider (CSP)

FIGURE A.3 - Scénario de récupération des données

A.3.1.4 Scénario de partage des données

Notre proposition distingue deux scénarios de partage de données, selon le nombre de destinataires. Le premier scénario de partage *un à un* est défini lorsque le propriétaire de données (déposant) partage des contenus avec un seul utilisateur (récépteur). Le deuxième scénario de partage *un à plusieurs* est défini quand un client partage ses données avec un groupe d'utilisateurs. La figure A.4 présente les différents échanges entre un déposant et son récepteur, lors d'un scénario de partage de données *un à un*. Ce scénario s'établit en deux phases: une phase de dépot des données, exécutée par le propriétaire de données, et une phase de récupération des données stockées par le récépteur. Il est à noter que les deux phases peuvent s'effectuer séparement.

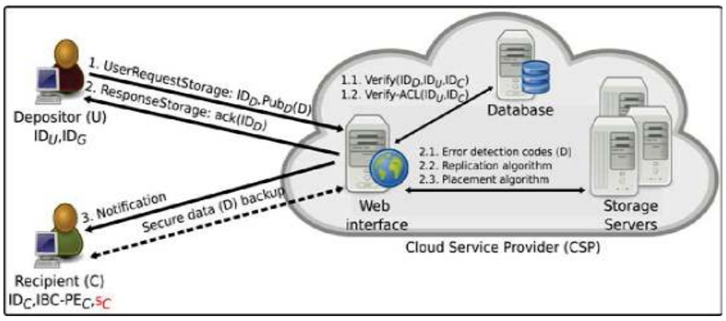
Depositor (U)
ID_U,ID_G
1. UserRequestStorage: ID_D,Pub_D(D)
2. ResponseStorage: ack(ID_D)
1.1. Verify(ID_D,ID_U,ID_C)
1.2. Verify-ACL(ID_U,ID_C)
Database
2.1. Error detection codes (D)
2.2. Replication algorithm
2.3. Placement algorithm
3. Notification
Secure data (D) backup
Web interface
Storage Servers
Cloud Service Provider (CSP)
Recipient (C)
ID_C,IBC-PE_C,s_C

FIGURE A.4 - Scénario de partage de données entre un déposant et un récépteur

La Figure A.5 présente les différents échanges entre un déposant et un groupe des récepteurs, lors d'un scénario de partage de données *un à plusieurs*.

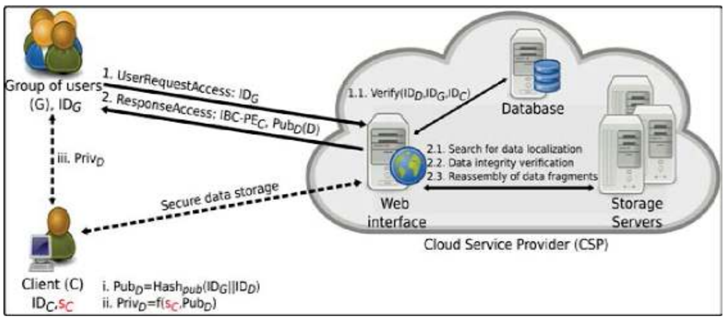
Group of users (G), ID_G
1. UserRequestAccess: ID_G
2. ResponseAccess: IBC-PE_C, Pub_D(D)
1.1. Verify(ID_D,ID_G,ID_C)
Database
iii. Priv_D
2.1. Search for data localization
2.2. Data integrity verification
2.3. Reassembly of data fragments
Secure data storage
Web interface
Storage Servers
Cloud Service Provider (CSP)
Client (C)
ID_C,s_C
i. Pub_D=Hash_pub(ID_G||ID_D)
ii. Priv_D=f(s_C,Pub_D)

FIGURE A.5 - Scénario de partage de données entre un déposant et plusieurs récepteurs

A.3.1.5 Conclusion

La montée de la demande des services de stockage sécurisés corrélée aux spécificités de la cryptographie basée sur l'identité nous ont permis de définir une solution innovante pour assurer la confidentialité des données stockées sur des serveurs distants.

Cette première contribution [KBL13] est basée sur une application d'IBC, permettant ainsi une meilleure protection de la vie privée et confidentialité des données. La solution proposée a fait preuve de résistance face aux accès non autorisés aux données et à toute divulgation de données au cours du processus de partage, tout en tenant compte de deux modèles de menaces réalistes, à savoir un serveur honnête mais curieux et un utilisateur

malveillant.
En outre, grâce au processus de calcul avantageux de l'IBC et contrairement aux schémas de chiffrement existants, notre proposition n'exige pas du déposant d'être connecté, lorsque les destinataires veulent récupérer les données partagées.
En outre, plusieurs expérimentations ont été réalisées, afin d'évaluer le coût de calcul de certains schémas de chiffrement basés sur l'identité du côté du client. Nous avons conclu que les algorithmes de chiffrement basés sur l'identité restent encore plus lents que les algorithmes de chiffrement symétriques tels que *AES*. En outre, nous affirmons que IBC doit être considéré comme un compromis intéressant entre les coûts de calcul et de mémoire.

Enfin, une discussion générale est présentée, tout en introduisant deux améliorations possibles à notre contribution. La première approche consiste à partager le rôle du PKG entre le serveur cloud et le client, afin d'alléger la complexité de calcul du côté du client. La seconde présente notre deuxième contribution qui détaille un système de déduplication pour les applications du cloud computing [KL14].

A.3.2 CloudaSec: Un protocle à clé publiques pour un partage sécurisé de données

Cette section présente notre troisième contribution, CloudaSec [KLEB14]. CloudaSec est un protocole à clé publique permettant un partage sécurisé des données.

A.3.2.1 Présentation de CloudaSec

Pour sécuriser le partage de données au sein d'un groupe d'utilisateurs, l'architecture CloudaSec introduit le rôle d'un chef de groupe, désigné par GM. Le chef de groupe prend en charge la constitution d'un groupe, la génération des paramètres publiques, l'enregistrement d'un nouveau utilisateur et la révocation d'un membre. La Figure A.6 illustre l'architecture de CloudaSec.
Ensuite, nous nous référons aux utilisateur(s) autorisé(s) comme récépteur(s) et au propriétaire des données comme déposant. Il est à noter que CloudaSec n'exige pas des destinataires d'être connectés au cours du processus de partage. En effet, les droits d'accès des bénéficiaires se sont accordés par le propriétaire des données et gérés par le CSP.

Pour protéger les données stockées sur des serveurs publiques de cloud, CloudaSec met à disposition du propriétaire des données plusieurs outils cryptographiques lui permettant de s'assurer que seuls les utilisateurs autorisés sont en mesure d'obtenir les clés de déchiffrement de données.
Notre protocole repose sur le chiffrement convergent [WQPW10], un mécanisme de chiffrement qui dépend du contenu des données. En effet, il présente deux niveaux: le chiffrement symétrique des données et le chiffrement asymétrique de la clé:

- ***chiffrement symétrique des données*** – avant de stocker les données sur les serveurs du cloud, le déposant chiffre le contenu des fichiers, en utilisant un algorithme symétrique. En effet, la clé de chiffrement de données est dérivée à partir du contenu du fichier en clair, en utilisant une fonction de hachage. Par conséquent,

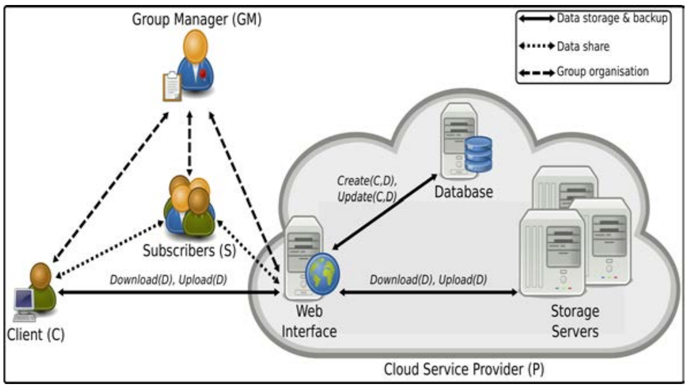

FIGURE A.6 - Architecture de CloudaSec

les capacité de stockage sont conservées puisque le même contenu sera chiffré de la même manière, produisant le même contenu.

- ***chiffrement asymétrique de la clé*** – le déposant chiffre la clé de déchiffrement k, en se basant sur un algorithme asymétrique, tout en utilisant la clé publique du destinataire. Puis, il intégre la clé cryptée dans les métadonnées du fichier, garantissant ainsi des politiques d'accès flexibles. En effet, n'importe quel destinataire autorisé peut accéder aux métadonnées du fichier, afin de déchiffrer la clé de données chiffrées à l'aide de sa clé privée.

Les procédures de CloudaSec définissent deux couches conjointement liées: une couche de données et une couche de gestion. Dans la couche de données, nous introduisons les opérations appliquées sur les données et les clés de chiffrement correspodantes, à savoir les procédures *GenerateParameters*, *EncryptData*, *DecryptData*, $EncryptKey_{OneToOne}$, $EncryptKey_{OneToMany}$ et *ShrinKey*. Dans la couche de gestion, CloudaSec introduit des procédures de *user revocation*, quand un ancien membre quitte le groupe ou est révoqué du groupe, et *user subscription*, lorsqu'un nouvel utilisateur rejoint le groupe.
CloudaSec permet un accès souple aux contenus chiffrés, en partageant de manière dynamique une clé secrète au sein du groupe. En effet, lorsque l'état du groupe est modifié en raison d'un nouveau abonnement ou d'une révocation, le GM diffuse la nouvelle constitution du groupe aux membres autorisés afin de générer la nouvelle clé secrète du groupe. La dérivation de cette clé secrète se base sur les éléments publiques publiés, sans mettre à jour les clés privées des autres utilisateurs.
CloudaSec distingue deux scénarios de partage de données différents. Premièrement, les données partagées *un à un*, où un propriétaire de données partage des contenus avec un seul utilisateur. Deuxièmement, les données partagées *un à plusieurs*, où un propriétaire de données partage des données au sein d'un groupe d'utilisateurs autorisés. Ces scénarios englobent deux algorithmes de chiffrement de clé différents, respectivement $EncryptKey_{OneToOne}$ et $EncryptKey_{OneToMany}$.

A.3.2.2 Procédures de CloudaSec sur le plan de données

CloudaSec définit deux scénarios de partage de données, à savoir le partage de données entre un déposant et un récépteur (partage *un à un*), et le partage au sein d'un groupe d'utilisateurs (partage *un à plusieurs*).

Le scénario de partage de données *un à un*, est défini, quand un propriétaire de données U_i partage des contenus avec un seul récepteur U_j. D'abord, le déposant U_i chiffre le fichier f, comme présenté dans l'algorithme 16, en se basant sur un algorithme de chiffrement symétrique *SymEnc*, et la clé de chiffrement k. Rappelons que k est généré à partir du contenu du fichier, en appliquant une fonction de hachage $H()$.

Algorithm 16 La procédure EncryptData

```
1: Input: $\{f, H, SymEnc\}$, w
2: Output: $< C_f, k >$
3: $k = H(f)$;
4: $C_f = SymEnc(f, k)$;
5: return $< C_f, k >$
```

Ensuite, U_i envoie le contenu chiffré C_f à son fournisseur de service afin de le stocker pour son correspondant U_j. Au vue d'accorder des droits d'accès au récépteur U_j, le propriétaire de données chiffre la clé de déchiffrement avec la clé publique du destinataire pk_j, comme présenté dans l'algorithme $EncryptKey_{OneToOne}$ (cf. Algorithm 17). En effet, CloudaSec introduit un nouveau mécanisme de chiffrement de la clé, permettant d'assurer un contrôle d'accès plus flexible, dans les environements cloud.
La clé de déchiffrement de données est encodée en deux éléments $< C_1, C_2 >$. C_1 est introduite par le déposant U_i, dans les méta-données. Par ailleurs, C_2 est envoyé au fournisseur de service, afin d'ajouter une couche de vérification d'accès supplémentaire et de permettre la génération de la clé re-dirigée.

Algorithm 17 La procédure EncryptKey$_{OneToOne}$

```
1: Input: $\{params, k, sk_i, pk_i, pk_j, pk_c\}$
2: Output: $< C_1, C_2 >$
3: $r \in_R \mathbb{Z}_n$;
4: $C_1 = k \oplus F(\hat{e}(pk_i, pk_j)^r)$;
5: $C_2 = \hat{e}(pk_c, r \cdot P)^{sk_i}$;
6: return $< C_1, C_2 >$
```

Nous supposons que le fournisseur de service détient une paire de clés publique et privée $< sk_c, pk_c >$, telque $sk_c = s_c \in_R \mathbb{Z}_n$ présente la clé privée, tandis que $pk_c = s_c \cdot P \in \mathbb{G}_1^*$ est la clé publique. Après la réceptiondu deuxième élement C_2, le fournisseur de service exécute l'algorithme *ShrinKey* pour générer la clé re-dirigée C_3, comme présenté dans l'algorithme 18. Ce dernier chiffre C_2, à l'aide de sa clé secrète sk_c et génère C_3.

La Figure A.7 présente un schéma représentatif des relations entre les différentes procéduces CloudaSec du scénario de partage *un à un*.

Algorithm 18 La procédure ShrinKey

```
1: Input: ${C_2, sk_c}$
2: Output: $C_3$
3: $C_3 = (C_2)^{\frac{1}{sk_c}}$;
4: return $C_3$
```

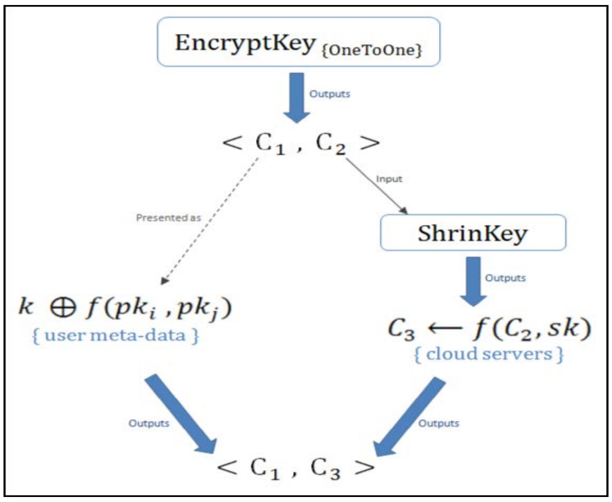

FIGURE A.7 - Schéma représentatif des relations entre les différentes procéduces CloudaSec du scénario de partage entre un déposant et un récépteur

Afin de récupérer les données stockées par le déposant U_i, le récépteur U_j, où $j \neq i$, doit tout d'abord retrouver la paire de clés $< C_1, C_3 >$. Par conséquent, U_j commence ainsi un scénario de récupération de données. En effet, après être authentifié, U_j demande C_3, à son fournisseur de service. Ensuite, en se basant sur les métadonnées du fichier, cet utilisateur autorisé U_j récupère l'élement C_1, chiffré à l'aide de sa clé publique pk_j par son correspondant U_i. Puis, le récépteur U_j utilise sa clé secrète sk_j pour exécuter la procédure *DecryptKey*$_{OneToOne}$ lui permettant de retrouver la clé de déchiffrement du contenu, comme présenté dans l'algorithme 19. Enfin, le récépteur peut retrouver les

Algorithm 19 La procédure DecryptKey$_{OneToOne}$

```
1: Input: ${params, < C_1, C_3 >, sk_j}$
2: Output: $k$
3: $C_1 \oplus F((C_3)^{sk_j})$;
4: return $k$
```

données qui lui ont été stockées par le déposant, en exécutant l'algorithme *DecryptData* à l'aide de la clé de déchiffrement k (cf. Algorithm 20).

Algorithm 20 La procédure DecryptData

```
1: Input: $\{C_f, k, SymEnc\}$
2: Output: $f$
3: $f = SymEnc(C_f, k)$ ;
4: return $f$
```

A.4 Intégrité des données dans le cloud

Le traitement des données est un sujet sensible. Les entreprises et particuliers exigent de plus en plus de transparence de la part des fournisseurs de solutions cloud quant aux moyens déployés pour exécuter leurs prestations. Le recours à des sous-traitants et le transfert des données à l'étranger peuvent constituer un risque quant à la garantie de confidentialité et d'intégrité des données.
Dans cette section, nous traitons une seconde préoccupation de sécurité dans les environnements de cloud: la preuve de possession de données. Des algorithmes de vérification ont été élaborés afin de satisfaire ce besoin de sécurité. Ces algorithmes permettent au client cloud de vérifier que son fournisseur de service possède les données dans l'état où il les a envoyées. En tenant compte d'une complexité d'autant plus importante, prenant en considération que le client ne dispose pas des données localement, il a fallu concevoir de nouveaux schémas adaptés aux environnements cloud.

A.4.1 Protocoles de Preuves de Possession des Données

La preuve de la possession de données (PDP) est un protocole *challenge-response* permettant à un client de vérifier si un fichier de données D stocké sur un serveur distant est disponible sous sa forme originale. Un système PDP se compose de quatre procédures: le pré-traitement, la requête, la preuve et la vérification (cf. Figure A.8). Pour la génération des méta-données d'un fichier, le client exécute la procédure du pré-traitement. Dans la plupart des cas, le client maintient les méta-données et envoie une version du fichier de données au serveur de stockage (par exemple, des données chiffrées). Pour vérifier l'authenticité du fichier, le client envoie une requête au serveur contenant un *challenge* aléatoire afin d'éviter les attaques par rejeu. Le serveur génère ainsi la preuve requise. cette preuve doit faire usage du *challenge* reçu et des données hébergées. De son côté, après avoir reçu la réponse de son fournisseur de service, le client compare la preuve avec les méta-données stockées en local.

La méthode naïve pour concevoir un système PDP est basée sur une fonction de hachage $H()$. En effet, le client pré-calcule k défis aléatoires c_i, tels que $i \in \{1, k\}$ et calcule les preuves correspondantes: $p_i = H(c_i||D)$. Dans sa requête, le client envoie c_i au serveur distant qui calcule $p'_i = H(c_i||D)$. Si la comparaison est exacte, le client suppose que le fournisseur de cloud héberge le bon fichier de données. Le plus grand inconvénient de ce système est le nombre pré-fixé des requêtes qui ont été calculées dans la procédure du pré-traitement. En effet, le client ne peut vérifier l'intégrité de ses données que k fois.

Les algorithms de preuve de possession doivent satisfaire plusieurs exigences, à savoir,

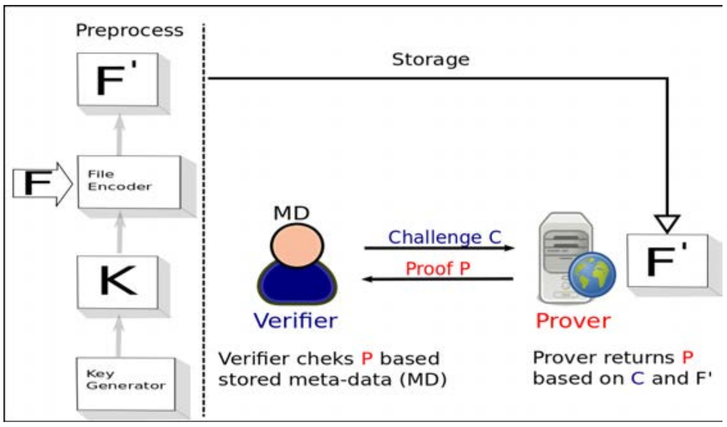

Figure A.8 - Schéma représentatif d'un prorocole PDP

le niveau de sécurité des données, la vérification publique, l'efficacité et la robustesse:

- **vérification publique:** la vérification publique de la possession des données est une exigence importante, permettant à toute entité autorisée de vérifier l'intégrité des données stockées sur des serveurs publiques distants. Par conséquent, le propriétaire des données peut être relevé de la charge du stockage et du calcul.
- **vérification sans état:** les preuves doivent être générées selon un *challenge* aléatoire. La vérification nécessite ainsi l'utilisation de valeurs imprévisibles.
- **coût de calcul faible:** d'une part, pour des raisons d'évolutivité, le coût de calcul sur le serveur de stockage devrait être réduit au minimum, puisque le serveur pourrait être impliqué dans plusieurs interactions simultanées. D'autre part, les algorithmes proposés doivent également avoir une faible complexité de traitement, du côté client, vu ses faibles capacités de calcul.
- **coût de communication faible:** un PDP efficace devrait réduire au minimum l'utilisation de la bande passante, en s'appuyant sur un faible coût de communication.
- **coût de stockage faible:** les capacités de stockage limitées des dispositifs de l'utilisateur ont une importance cruciale dans la conception de nos solutions. De ce fait, un faible coût de stockage est fortement recommandé.
- **challenges illimités:** le nombre de challenges devrait être illimité. Cette condition est considérée comme importante pour l'efficacité d'un système PDP.

A.4.2 Preuve de Possession de Données (PDP), sans apport de connaissance

Dans cette section, nous présentons, notre contribution définissant un nouveau protocole de preuve de possession, basé sur des mécanismes de preuve sans apport de connaissance [KEML14].

A.4.2.1 Présentation

Profitant des propriétés des systèmes interactifs de preuve, sans apport de connaissance [Gol00], nous présentons un nouvel algorithme PDP, basé sur le protocole GPS, proposé dans [GPS06]. Ainsi, nous étendons ce mécanisme à la vérification de l'authenticité des fichiers stockés sur des serveurs cloud.

Nous proposons deux variantes de preuve de possession de données, à savoir un schéma de vérification privée et un algorithme de vérification publique. La vérification privée fait usage d'un secret stocké en local chez le client, tandis que la preuve publique est basée sur des fonctions de couplage, permettant ainsi une entité autorisée à vérifier le contenu stocké par une autre entité.

Le choix d'étendre le protocole GPS est motivé par plusieurs facteurs. En premier lieu, les preuves, sans apport de connaissance, ajoutent des valeurs aléatoires aux messages échangés. En effet, pour chaque session de vérification, le prouveur et le vérifieur doivent générer de nouvelles valeurs aléatoires, ce qui rend chaque session différente des sessions précédentes. Par conséquent, le caractère aléatoire impliqué dans les réponses du serveur permet la préservation de la confidentialité des données. En deuxième lieu, le protocole GPS a été adapté aux capacités de stockage limitées des *tags*. Avec la prévalence de la communication sans fil, les appareils mobiles commencent à partager les avantages des services de stockage dans le cloud. Au vu des ressources limitées de ces appareils, notre protocole est basé sur un seul secret qui est nécessaire pour la vérification de toutes données externalisées. En outre, le principal avantage de notre approche est la vérification publique, permettant au vérifieur de se baser uniquement sur des éléments publiques, sans aucune intervention du propriétaire des données.

A.4.2.2 Vérification privée de la preuve de possession

Dans notre système, nous définissons une courbe elliptique EC définie sur un sous-groupe additif $\mathbb{G}_1$ d'ordre premier q. Soit P un générateur de $\mathbb{G}_1$.

Quand un client veut stocker un fichier de données D, il doit, d'abord, décomposer D en deux blocs s et n. n représente le quotient et s est le reste de la division euclidienne appliquée sur le fichier D à l'aide du diviseur b. Il est à noter que b est maintenu secret par le client et est utilisé dans la décomposition de plusieurs fichiers. Il représente l'unique information secrète qu'un client doit conserver pour l'ensemble de ses demandes de preuve de la possession de données. Notons que b est étroitement lié à la sécurité de notre système de vérification à distance. En effet, la définition de plusieurs diviseurs de données peut étendre notre proposition permettant le propriétaire des données de s'appuyer sur différents secrets, selon le niveau de sensibilité des données.
Ensuite, en se basant sur le *ECDLP*, les éléments publiés sont bP, nP, SP, notés respectivement pk, σ_1 et σ_2. pk représente la clé publique du propriétaire de données, tandis que les σ_1 et σ_2 sont les éléments publiques du fichier D.

Notre protocole consiste en deux phases. Au cours de la première phase, les algorithmes *KeyGen* et *Setup* sont exécutés. *KeyGen* est la procédure permettant de générer une paire de clés publique et privée du propriétaire de données, tandis que *Setup* applique la division

euclidienne sur le fichier concerné afin de générer σ_1 et σ_2. Cette phase n'est effectuée que lorsque le fichier est téléchargé sur les serveurs du cloud.
La deuxième phase se produit lorsque le client veut vérifier l'authenticité du fichier de données. En effet, il génère une nouvelle requête b' afin d'obtenir une preuve de la possession de données de son fournisseur de service. b' représente un diviseur aléatoire choisi par le propriétaire de données pour avoir une nouvelle décomposition du fichier.

A la réception de ce nouveau diviseur, le fournisseur de service exécute, ainsi, un algorithme *ChalProof* qui consiste à échanger trois messages avec le vérifieur, comme suit:

- Commitment $(CSP \to C)$: le serveur de stockage calcule la nouvelle décomposition en se basant sur b', telle que:
$$D = mb' + z$$
Ensuite, il choisit deux nombres aléatoires $(r, t) \in_R [0,\ B[^2$ et envoie les commitments correspondants au client $(x_1, x_2) = (r \cdot P, t \cdot P)$.

- Challenge $(C \to CSP)$: le client choisit un nombre aléatoire $c \in_R [0,\ K[$ qu'il envoie au prouveur.

- Response $(CSP \to C)$: le serveur calcule γ_1 et γ_2, en se basant sur (r, t), et le couple (m, z):
$$(\gamma_1, \gamma_2) = (r + cz, t + cm)$$
Après, le prouveur envoie (y_1, y_2), telque $y_1 = \gamma_1 \cdot P$ et $y_2 = \gamma_2 \cdot P$.

Une fois cette preuve reçue, le fournisseur de service exécute une procédure *Verify.* Il vérifie l'intégrité de son fichier stocké sur un serveur distant, en se basant sur l'unicité de la division euclidienne appliquée sur le même contenu, comme suit:

$$y_1 - x_1 - c \cdot \sigma_1 = c.sk \cdot \sigma_2 - b' \cdot (y_2 - x_2). \tag{A.2}$$

A.4.2.3 Vérification publique de la preuve de possession

Un utilisateur autorisé, différent du propriétaire de données, pourrait également vérifier l'authenticité des données. Toutefois, le protocole proposé, dans le paragraphe précédent, ne peut pas être utilisé à ce fin, car il suppose que le vérifieur possède la clé privée du client, ce qui n'est pas le cas de l'utilisateur. Dans ce qui suit, nous démontrons que les paramètres publiques du client peuvent également être utilisés afin de mettre en œuvre un système PDP entre un utilisateur, différent du propriétaire du fichier, et le serveur de stockage, en se basant sur les fonctions de couplage $\hat{e}$.

La vérification publique vérifie l'équation suivante:

$$\hat{e}(c \cdot \sigma_1, pk) \star \hat{e}(c \cdot \sigma_2, P) = \hat{e}(y, P) \star \hat{e}(x_1 + b'x_2, P)^{-1}. \tag{A.3}$$

où $y = (\gamma_1 + b'\gamma_2) \cdot P$.

A.4.3 SHoPS: Preuve de possession d'ensembles de données homomorphiques

Dans cette section, nous présentons SHoPS, un protocole de preuve de possession de données capable de traiter les trois relations d'ensembles homomorphiques. SHoPS permet ainsi au client non seulement d'obtenir une preuve de la possession du serveur distant, mais aussi de vérifier que le fichier, en question, est bien réparti sur plusieurs périphériques de stockage permettant d'atteindre un certain niveau de la tolérance aux pannes. En effet, nous présentons l'ensemble des propriétés homomorphiques, qui étend la malléabilité du procédé aux propriétés d'union, intersection et inclusion.

A.4.3.1 Présentation de SHoPS

Pour tolérer les pannes des disques, chaque fichier de données est stocké avec la redondance nécessaire. Ensuite, chaque fichier de données est divisé en *blocs*, et chaque bloc B est divisé en q *sous-blocs*, où q est un paramètre du système. Chaque sous-bloc est représenté par un seul élément d'un groupe multiplicatif $\mathbb{G}_2$. Notre protocole de preuve de possession de données, appliqué à un seul bloc de données, est constitué de cinq algorithmes aléatoires, sur la base de deux phases. Pendant la première phase, les procédures d'initialisation du système sont exécutées. Cette phase est effectuée une seule fois lorsque le fichier est stocké sur les serveurs cloud:

- `gen` : $\{1\}^{\lambda} \to \mathcal{K}_{pub}{}^2 \times \mathcal{K}_{pr} \times \mathbb{G}_2{}^{2q-1}$ – soit λ le paramètre de sécurité, cet algorithme génère une paire de clés publique et privée du propriétaire des données $(pk, \hat{pk}, sk)$, et un ensemble d'éléments publiques, basé sur le problème *Diffie-Hellman Exponent* [DH76].

- `stp` : $2^{\mathcal{M}} \times \mathbb{G}_2{}^q \to \mathbb{G}_2$ – Soient un bloc de données $B_i \in \{0,1\}^*$ et la clé publique pk, l'algorithme `stp` génère un identifiant du bloc B_i, noté ID_{B_i} et son accumulateur $\{B_i, \varpi_i\}$, tels que $i \in \{1, \cdots, n\}$ et n est le nombre de blocs.

La deuxième phase est exécutée, quand un propriétaire de données, veut vérifier l'authenticité d'un bloc de données:

- `clg` : $\mathbb{Z}_p{}^* \times \mathbb{Z}_p{}^* \to \mathcal{C}$ – cet algorithme probabiliste est exécuté par le client. Il prend comme entrée le nombre de blocs de données q. Il génère ainsi un *challenge* $c \in \mathcal{C}$ qui consiste à un index d'un sous-bloc choisi aléatoirement et un élément de la clé publique $\hat{pk}$ caché par un nonce η telque $c = (i, \hat{pk}^{\eta})$.

- `prf` : $\mathcal{K}_{pub} \times 2^{\mathcal{M}} \times \mathcal{C} \to \mathcal{P}$ – l'algorithme `prf` calcule la réponse du serveur $P = (\sigma_1, \sigma_2)$ suite au challenge reçu par le client, en utilisant les blocs de données stockés dans ses serveurs.

- `vrf` : $\mathcal{P} \times \mathcal{K}_{pub}{}^2 \to \{0,1\}$ – il s'agit d'une fonction de vérification des réponses reçues P du serveur de stockage, où 1 signifie *succès* (i.e., le client a vérifié avec succès l'intégrité de ses données stockées sur le serveur cloud). Par ailleurs, 0 signifie un *rejet*.

A.4.3.2 Vérification publique d'un bloc de données SHoPS

La preuve de possession de données, présentée dans cette section, est limitée à un seul bloc. Comme présenté ci-dessus, une première phase traitant les procédures d'initialisation du système est exécutée. Elle comporte deux procédures, à savoir l'algorithme `gen` (cf, Algorithm 21) et l'algorithme `stp` (cf, Algorithm 22).

Algorithm 21 La procédure `gen` de SHoPS

```
1: Input: paramètre de sécurité $(\xi)$
2: Output: $(pk, \hat{pk})$, $pr$, $param = \{g_i\}_{1\leq i\leq 2q; i\neq q+1}$
3: Choisir un groupe multiplicatif $\mathbb{G}_1$ d'ordre premier $q$;
4: Choisir un générateur $g$ du groupe $\mathbb{G}_1$;
5: $\alpha \overset{R}{\leftarrow} \mathbb{Z}_p{}^*$;
6: $param = \{g\}$
7: for all $j \in [1 \dots 2q]$ do
8:     $param \leftarrow param \cup \{g^{\alpha^j}\}$
9: end for
10: Générer $s \overset{R}{\leftarrow} \mathbb{Z}_p$;
11: $pr \leftarrow s$;
12: $pk \leftarrow g^s$;
13: $\hat{pk} \leftarrow g_{q+1}^s$;
14: return $(pk, \hat{pk}, pr, \{g_i\}_{1\leq i\leq 2q; i\neq q+1})$
```

Algorithm 22 La Procédure `stp` de SHoPS

```
1: Input: $(B_i)$, $pr$, $param$
2: Output: $ID_{B_i}$, $\varpi$
3: Générer un identifiant du bloc $ID_{B_i}$;
4: $\varpi_i = 1$;
5: for all $j \in [1 \dots q]$ do
6:     $\varpi_i \leftarrow \varpi_i * {g_{q+1-j}^{\pi_{i,j}}}^{pr}$;
7: end for
8: return $(ID_{B_i}, \varpi_i)$
```

La deuxième phase est un protocole *challenge-response* entre un vérifieur autorisé et le fournisseur de service.

D'abord, la procédure `clg` est exécutée par le vérifieur pour générer un *challenge* qu'il envoie au serveur de stockage. Le client choisit, aléatoirement, un index $k \in \{1, q\}$ qui représente une position d'un sous bloc de données et un nonce η. Le *challenge* $c \in \mathcal{C}$ consiste ainsi en un index d'un sous bloc et un élément publique $\hat{pk}$, caché à l'aide d'un nonce η telque $c = (k, \hat{pk}^\eta)$.

A la réception du *challenge*, le serveur de stockage exécute une procédure `prf` pour générer la preuve requise. En effet, le prouveur doit fournir un nouveau accumulateur, en utilisant le nonce η envoyé par le client. Dans notre construction, la procédure `prf` est définie par l'algorithm 23. Par souci de consistence, nous supposons que le serveur possède une version du du bloc de données qui est potentiellement altérée. Au-delà, cette version est désignée par $\hat{B_i}$. Le but des étapes suivantes est de vérifier si $\hat{B_i} = B_i$.

Algorithm 23 La procédure `prf` de SHoPS

```
1:  **Input:** $(B_i)$, $(pk, \hat{pk})$, $param$ and the challenge $c = (k, \hat{pk}^{\eta})$
2:  **Output:** $P = (\sigma_1, \sigma_2)$
3:  $\sigma_1 \leftarrow (\hat{pk}^{\eta})^{\pi_{i,k}}$;
4:  $\hat{\varpi}_i = 1$;
5:  for all $j \in [1 \ldots q]$ do
6:      if $j \neq k$ then
7:          $\hat{\varpi}_i \leftarrow \hat{\varpi}_i * g_{q+1-j+k}^{\pi_{i,j}}$;
8:      end if
9:  end for
10: $\sigma_2 \leftarrow \hat{\varpi}_i$;
11: return $(\sigma_1, \sigma_2)$
```

Une fois la preuve (σ_1, σ_2) reçue, le vérifieur publique exécute une procédure `vrf`, en se basant sur les paramètres publiques *param*. Le vérifieur valide l'authenticité du bloc de données, en vérifiant l'égalité suivante, en se basant sur le nonce η, le *challenge* c, et la réponse $P = (\sigma_1, \sigma_2)$ reçue du fournisseur de service, comme suit:.

$$[\hat{e}(g_k, \varpi_i)\hat{e}(pk, \sigma_2)^{-1}]^{\eta}\hat{e}(g, \sigma_1)^{-1} = 1 \tag{A.4}$$

A.5 Conclusion

Nous avons présenté dans ce travail quatre contributions, pour pallier à deux besoins de sécurité dans des environnements de stockage en nuage, à savoir la confidentialité des données et l'intégrité des données.

Notre *Objectif A* consiste à définir de nouvelles méthodes pour améliorer la confidentialité des données dans des applications cloud, tout en améliorant le partage dynamique entre les utilisateurs. En réponse à cet objectif, nous avons proposé deux approches différentes, basées sur l'utilisation de la cryptographie basée sur l'identité et la cryptographie convergente, respectivement.

Afin de remplir l'*Objectif C* qui consiste à implémenter les techniques proposés pour valider leur faisabilité et leur impact sur les performances des équipements, des expérimentations ont été menées dans le cadre d'un *testbed OpenStack*. Les résultats de l'implémentation de nos solutions, notamment CloudaSec et les schémas basés sur l'identité, ont affirmé que ces propositions représentent des alternatives intéressantes, particulièrement pour des dispositifs, ayant des capacités de calcul et de stockage limitées.

les résultats expérimentaux ont montré l'efficacité de notre solution basée sur l'ID et de CloudaSec, compte tenu de l'impact de l'exécution des opérations cryptographiques du côté du client. Des expérimentations ont été menées dans un .

L'*Objectif B* consiste à aborder la preuve de possession des données stockées sur des serveurs distants dans les environnements de stockage en nuage. Considérant trois exigences de conception: *niveau de sécurité, vérifiabilité publique*, *performances* et des capacités de stockage limitées des dispositifs de l'utilisateur, nous avons proposé deux systèmes de vérification publiques. Le premier algorithme se base sur des mécanismes de preuve sans apport de connaissance et consiste à générer à chaque session une composition différente du fichier. Le deuxième algorithme, désigné SHoPS, permet de vérifier l'intégrité d'un fichier de données, ainsi que sa répartition sur plusieurs périphériques de stockage permettant d'atteindre un certain niveau de tolérance aux pannes.

L'*Objectif D* consiste à fournir des preuves mathématiques de l'exactitude des mécanismes proposés. D'un côté, afin de s'acquitter de ce quatrième objectif, nous avons démontré l'exactitude des procédures de CloudaSec, à savoir les procédures de chiffrement et de déchiffrement de la clé, pour les deux scénarios de partage. En outre, nous avons également montré l'exactitude de la vérification publique et privée de SHoPS et de notre protocole à base de preuves sans apport de connaissance.
D'un autre côté, basé sur un jeu de possession de données, nous avons montré que nos algorithmes PDP, sans apport de connaissance, ainsi que SHoPS, étaient résistants aux attaques de divulgation de données.

Nos perspectives de recherche comprennent:

- une évaluation de la sécurité de CloudaSec et de SHoPS, à l'aide de la sécurité prouvée et le modèle de l'oracle aléatoire.
- avec la tendance forte des appareils mobiles (c'est à dire, les *smartphones*, les tablettes, *cdots*) à faire appel aux services de stockage de données en nuage, il serait intéressant d'étudier les mécanismes de sécurité des données de faible coût. En effet, notre protocole de preuve de possession de données, à apport nul de connaissance, mérite d'être mis en œuvre sur un matériel mobile réel pour évaluer les coûts de calcul. Ces tests serviraient également à prédire l'efficacité énergétique des appareils mobiles, tout en appliquant les fonctions cryptographiques proposées par notre protocole.
- une évaluation des coûts de communication de CloudaSec, en faisant varier les paramètres influant sur la consommation de la bande passante, telles que la localisation des utilisateurs finaux, et la localisation des capacités de stockage.
- une évaluation de SHoPS dans un environnement des clouds imbriqués. Nous avons montré l'efficacité de SHoPS, au sein du même CSP. Par conséquent, il serait intéressant d'évaluer ses performances, quand ce stockage de données est distribué sur plusieurs sites. En tant que tel, SHoPS serait une alternative intéressante pour de nombreux fournisseurs qui préfèrent stocker des données sur différents sites.

Pour conclure, ce mémoire a été l'occasion d'examiner un large éventail de concepts, de modèles et de technologies dans les domaines de la sécurité de l'information et des réseaux. Notre objectif était d'étudier les problèmes de sécurité des données stockées en nuage, tout en se concentrant sur la confidentialité des données et les problèmes de vérification de l'intégrité des données à distance. Nous avons fourni de nouvelles approches

cryptographiques en réponse à quatre objectifs, et nous avons montré que notre travail proposé rejoint une thématique de recherche riche et encourageante.
Enfin, nous affirmons que la sécurité de stockage de données en nuage est toujours pleine de défis et d'une importance majeure, et de nombreux problèmes de recherche restent à identifier et à étudier.