CloudaSec : a novel public-key based framework to handle data sharing security in clouds

Nesrine Kaaniche, Maryline Laurent, Mohammed El Barbori

CloudaSec: A Novel Public-key based Framework to handle Data Sharing Security in Clouds

Nesrine Kaaniche[1], Maryline Laurent[1], Mohammed El Barbori[1]
[1]*Institute Mines Telecom, Telecom-SudParis*
{*nesrine.kaaniche, maryline.laurent, mohammed.el_barbori*}*@telecom-sudparis.eu*

Keywords: cloud storage systems, data security, access control, algorithm verification

Abstract: Recent years have witnessed the trend of leveraging cloud-based services for large scale content storage, processing, and distribution. Data security and privacy are among top concerns for the public cloud environments. Towards these security challenges, we propose and implement CloudaSec framework for securely sharing outsourced data via the public cloud. CloudaSec ensures the confidentiality of content in the public cloud environments with flexible access control policies for subscribers and efficient revocation mechanisms. CloudaSec proposes several cryptographic tools for data owners, based on a novel content hash keying system, by leveraging the Elliptic Curve Cryptography (ECC). The separation of subscription-based key management and confidentiality-oriented asymmetric encryption policies uniquely enables flexible and scalable deployment of the solution as well as strong security for outsourced data in cloud servers. Through experimental evaluation, we demonstrate the efficiency and scalability of CloudaSec, build upon OpenStack Swift testbed.

1 INTRODUCTION

Recently, 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 (Gantz and Reinsel, 2012). This explosive growth of data continues to rise the demand for new storage and network capacities, along with an increasing need for more cost effective architectures. As such, recent years have witnessed the trend of leveraging cloud data storage, since it provides efficient remote storage services in pay per use business model.

However, these promising data storage services have brought many challenging design issues, considerably due to the loss of control on outsourced data. One of the biggest concerns is data confidentiality provisioning which remains a fundamental security requirement in cloud storage services.

It is commonly agreed that data encryption at the client side is a good alternative to mitigate such concerns of data confidentiality. Thus, the client preserves the decrypting keys out of reach of the cloud provider. Nonetheless, 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, 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.

In this paper, we propose CloudaSec, a public key based solution for improving data confidentiality in cloud storage environments and enhancing dynamic sharing between users. CloudaSec applies the convergent encryption concept (Wang et al., 2010) 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 (Burmester and Desmedt, 2005). 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.

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.

Paper Organization– The remainder of this work is organized as follows: Section 2 presents security considerations and design goals. Then, Section 3 describes the system model, reviews some preliminaries and cryptographic primitives, details the framework design, and describes the prototype and its different procedures. In Section 4, rigorous security discussions are given, and implementation results are discussed in Section 5. Finally, we review the related work in Section 6, before concluding in Section 7.

2 PROBLEM STATEMENT

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). 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. Nonetheless, the confidentiality preservation becomes more complex with resilient data sharing among dynamic groups. Hence, secure data sharing should support flexible security policies including forward and backward secrecy.

- *Forward secrecy* – this property requires that the confidentiality of previous 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. As such, a revoked group member is unable to access data that were outsourced after he leaves the group.

Therefore, the design of our protocol 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 requires 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.

3 CLOUDASEC FRAMEWORK

This section presents CloudaSec architecture with four different types of players. Then, it introduces CloudaSec, a public key based framework to handle data sharing security, and it highlights the cryptographic assumptions that should be fulfilled by our proposed framework.

3.1 System Model

Figure 1 illustrates a descriptive network architecture for CloudaSec framework. It relies on the following entities for the good management of client data:

- *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.
- *Data owner*: a data owner makes use of provider's resources to store, retrieve and share data with multiple users. A data owner can be either an individual or an enterprise.
- *Group Manager (GM)*: a group manager takes charge of construction of a group, system parameters generation, user registration and user revocation. Therefore, we assume that the group manager is trusted by the other entities.
- *Users*: the users are able to access the content stored in the cloud, depending on their access

rights which are authorizations granted by the data owner, 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.

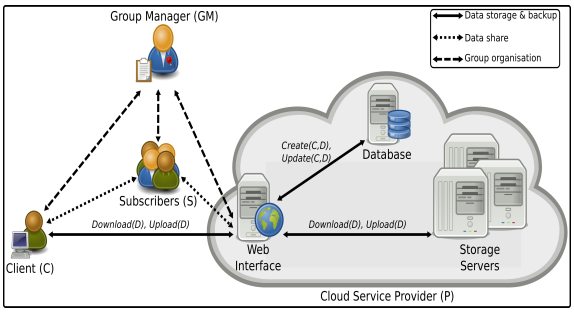

Figure 1: CloudaSec architecture

In practice, the CSP provides a web interface for data depositors 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. We assume that there is an established secure channel between the cloud user and the CSP. This secure channel supports mutual authentication and data confidentiality and integrity. It can be implemented through the Transport Layer Security protocol (TLS) (Dierks and Rescorla, 2008), where the client can authenticate with a certificate or password.

Next, we refer to these authorized user(s) as the recipient(s) and to the data owner as the depositor. 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 a redirected access key element.

3.2 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.

Our framework relies on the convergent encryption (Wang et al., 2010) 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*. 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 3.5.

CloudaSec distinguishes two different data sharing scenarios. First, the data sharing one to one, presented in Section 3.4.1, where a data owner stores for one CloudaSec user. Second, the data sharing one to many, described in Section 3.4.2, 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 this paper are listed in Table 1.

Table 1: Our 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

3.3 Cryptographic Background

This section reviews a straightforward cryptographic background, used in the design of our CloudaSec framework.

3.3.1 Preliminaries

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

Collision resistant hash functions(Boneh and Boyen, 2006) – 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 { (Regan,), (Ratna et al., 2004)} – 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.

3.3.2 Cryptographic Assumptions

Our proposal is based on two cryptographic assumptions, namely the Elliptic Curve Discrete Logarithm Problem and the Computational Diffie Hellman Problem.

Elliptic Curve Discrete Logarithm Problem (ECDLP) – given an additive group $\mathbb{G}$, a subgroup of $E(\mathbb{F}_p)$, which is generated by the point P of prime order n, it is intractable to find a, where $Q = aP$, and P are known.

Computational Diffie Hellman problem (CDH) – given a cyclic group $\mathbb{G}$ of order p and generator g, there is no efficient algorithm to calculate g^{ab} , where (g, g^a, g^b) are known.

3.3.3 Group Key Distribution (GKD)

Burmester and Desmedt propose an unauthorized key exchange protocol (Burmester and Desmedt, 2005). It is a two round protocol that extends the concept of the Diffie Hellman assumption.
Let $G = \{U_1, ...U_m\}$, be a group of m users arranged into a cycle. To generate a group key, 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 generator of a multiplicative group $\mathbb{G}$. Afterwards, this latter 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 K, as $K \equiv g^{b_1 b_2 + b_2 b_3 + ... + b_m b_1} \mod(p)$.

3.4 CloudaSec Data Layer Procedures

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 initializes the system and generates the public parameters *params*, according to a required security parameter ξ, as presented in Algorithm 1. That is, the system setup procedure generates the groups $\mathbb{G}_1$ and $\mathbb{G}_2$ and the pairing function $\hat{e}$ from $\mathbb{G}_1 \times \mathbb{G}_1$ in $\mathbb{G}_2$. $\mathbb{G}_1$ is an additive subgroup of the group of points of an Elliptic Curve (EC), where $\mathbb{G}_2$ is a multiplicative subgroup of a finite field. $\mathbb{G}_1$ and $\mathbb{G}_2$ have the same order n and are generated by P and $g = \hat{e}(P,P)$, respectively.
After the specification of the groups, CloudaSec *GenerateParameters* procedure defines a secure one way hash function $H : \mathbb{E} \rightarrow \{0,1\}^l$, 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

Algorithm 1 GenerateParameters

1: **Input:** Security parameter ξ
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** $params = \{\mathbb{G}_1, \mathbb{G}_2, \hat{e}, P, g, H, F, n\}$

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 $pk_t = s_t \cdot P$.

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

In the 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.

3.4.1 CloudaSec One to One Sharing Scenario

The One to One 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, U_i stores the encrypted content 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 (Algorithm 3). That

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** $< C_f, k >$

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 in our ap-

Algorithm 3 $\text{EncryptKey}_{OneToOne}$

1: **Input:** $\{params, k, sk_i, pk_i, pk_j, pk_c\}$
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: $C_1 = k \oplus F(\hat{e}(pk_i, pk_j)^r)$;
5: $C_2 = \hat{e}(pk_c, r \cdot P)^{sk_i}$;
6: **return** $< C_1, C_2 >$

proach that all key elements belong to a finite domain space D. We denote each key element by ***key.elt*** as defined in Equation 1, where D can be either a user metadata element space or a CSP metadata element space.

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

We must 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. 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 enciphers C_2, using his secret key sk_c and generates the corresponding C_3. Afterwards, when the CloudaSec recipient U_j, where $j \neq i$, 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 as follows.
After successfully authenticating with the CSP, U_j gets the redirected key element C_3. Then, based on the outsourced user metadata, the authorized recipi-

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

ent 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, based 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

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

retrieves the 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

3.4.2 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 (Algorithm **??**), based on parallel Diffie Hellman instantiations (Burmester and Desmedt, 2005), as explained in Section 3.5.
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 (Algorithm 4). When an au-

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** $< C_1, C_2 >$

thorized 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 3.4.1. Then, he runs the CloudaSec *DecryptKey*$_{OneToMany}$ procedure (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

3.5 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 2.

$$pubelts_i =< id_i, pk_i > \quad (2)$$

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

Algorithm 9 GenerateGroup

1: **Input:** n, p, ξ
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** $< \mathbb{G}, h >$

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 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

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** $< id_i, h_i >$

group member U_i. Then, he updates 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 3.

$$(h_i, X_i) = (h^{b_i}, (\frac{h_{i+1}}{h_{i-1}})^{b_i}) \quad (3)$$

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 2. Then, based on Algorithm **??**, 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 (Burmester and Desmedt, 2005).

Table 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})

3.5.1 User Subscription

When a new user $\{U_N, id_N\}$ wants to join the group Gr, presented by $Gr = \{\{U_0, id_0\}, ..., \{U_{N-1}, id_{N-1}\}\}$, where $i \notin \{0,...,N-1\}$, 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.

3.5.2 User Revocation

When a group member U_j leaves or is revoked from the group $Gr = \{id_0, id_1, id_2, ..., id_k\}$, where $j \in \{0, 1, ..., k\}$, 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_N, X_N) 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 based on a selection protocol, 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 based on a sharing One To One scenario.

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.

4.1 Threat Model

For designing the most suitable security solutions for cloud storage, we have to consider realistic threat models. That is, we point out two adversaries: malicious cloud user and *honest but curious* cloud server.

- *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.
- *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.

4.2 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 4.1. ***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 these two lemmas. □

Lemma 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 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:

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

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

  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 (Burmester and Desmedt, 2005). 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 and Equation 5, an attacker obtains indistinguishable data key elements $key.elt_1$, as follows.

$$C_1^{(f_1)} = k_1 \oplus F(\hat{e}(pk_i, r \cdot P)^d) \quad (4)$$

$$C_1^{(f_2)} = k_2 \oplus F(\hat{e}(pk_i, r \cdot P)^d) \quad (5)$$

  Hence, we notice that there is no polynomial-time algorithm that can deduce secret information from Equation 4$\oplus$Equation 5 = $k_1 \oplus k_2$, due 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 key. Thus, based on two successive sessions (α) and (β), the malicious cloud user gets the following key elements $key.elt_1$.

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

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

  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:

$$C_1^{(A)} = k \oplus F(\hat{e}(pk_i, r_A \cdot P)^{d_A}) \quad (6)$$

$$C_1^{(B)} = k \oplus F(\hat{e}(pk_i, r_B \cdot P)^{d_B}) \quad (7)$$

  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 G_B.
  From Equation 6, the recipient U_M extracts the data deciphering key k. In the sequel, from $C_1^{(B)}$, U_M computes Equation 8 as follows:

$$C_1^{(B)} \oplus k = k \oplus F(\hat{e}(pk_i, r_B \cdot P)^{d_B}) \oplus k \quad (8)$$

$$= F(\hat{e}(pk_i, r_B \cdot P)^{d_B}) \quad (9)$$

  From Equation 7 and Equation 8, 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 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 4.3. *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 10 and Equation 11 as follows:

$$C_2 \star [\frac{1}{\hat{e}(pk_c, pk_i)}] = \frac{g^{sk_i sk_c r}}{g^{sk_i sk_c}} = g^r \qquad (10)$$

$$C_3 \star [\frac{1}{\hat{e}(pk_i, pk_i)}] = \frac{g^{sk_i r}}{g^{sk_i{}^2}} = \frac{g^r}{g^{sk_i}} \qquad (11)$$

From Equation 10 and Equation 11, 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, based on the hardness of the CDH assumption. □

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. ***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 $k^* = C_1 \oplus F((C_3)^{sk_j})$.
  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 redi-

rected 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 4.5. *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 4.2 and Lemma 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. □

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.

5.1 Context

In order to evaluate the performances of our proposal, we build a simulated CloudaSec framework, based on OpenStack Storage system (Swift) (swi,). 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 (The OpenSSL Project, 2003), the GMP library (et al., 2002) and the Pairing Based Cryptography (PBC) library (Ben, 2007), 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.

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 2 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 can note that this computa-

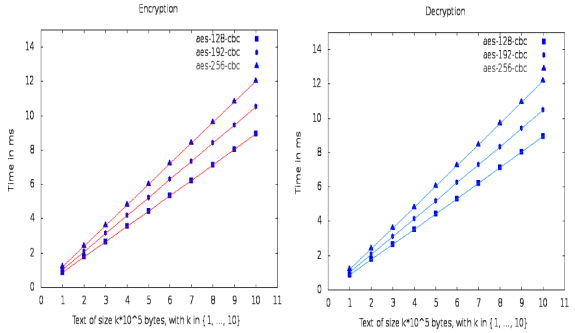

Figure 2: Computation overhead of data encryption and decryption at the client side with different data size (from 10^5 to 10^6 bytes) (ms)

tion cost 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 (Ben, 2007), 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 3).

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 in use $\hat{e}$, 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 3, 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. In fact, we examine the encryption operations vs Swift upload procedure and the decryption operations vs 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

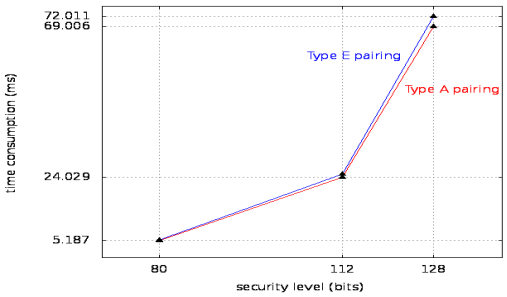

Figure 3: Computation duration of Type A vs Type E pairing functions (ms)

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.

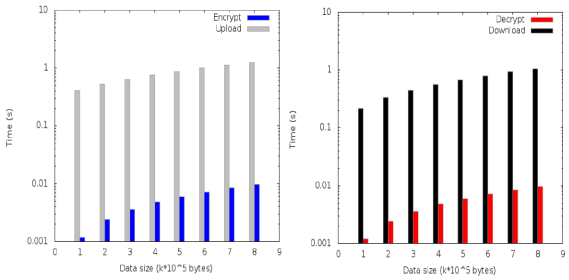

Figure 4: Impact of cryptographic operations on CloudaSec at the client side ($log_{10}(ms)$)

5.3 Communication Cost Evaluation

We investigate the communication overhead, when a client stores his data file to remote 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 5.

We can 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. 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 simulta-

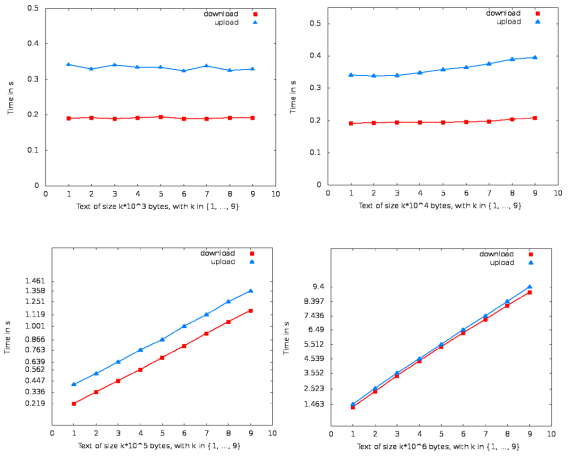

Figure 5: OpenStack upload and download overhead with different data size (ms)

neously, 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 3.3.

As depicted in Figure 6, 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.

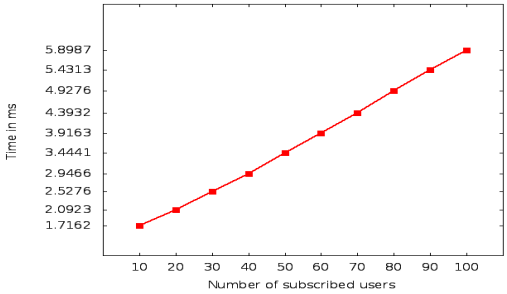

Figure 6: Computation complexity of a group update (ms)

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.

6 RELATED WORK

Several security solutions have been recently developed, in order to provide data confidentiality in public cloud storage environments { (Xiong et al., 2012),(Zarandioon et al., 2011), (Yu et al., 2010), (Zhou et al., 2011), (Liu et al., 2013),(Fugkeaw, 2012)}, while considering the group access control issues.

In (Yu et al., 2010), Yu et al. proposed an attribute based access control policy to securely outsource sensitive client data to untrusted cloud servers. In this approach, data are encrypted using a symmetric encryption algorithm, while the enciphering key is protected by a Key-Policy-Attribute Based Encryption scheme (Goyal et al., 2006). In order 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. However, CloudaSec defines a new revocation system based on (Burmester and Desmedt, 2005), 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.

In addition, several storage systems are based on the proxy re-encryption algorithms, in order to achieve fine grained access control {(Xiong et al., 2012; **?**),(Ateniese et al.,)}. 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. (Ateniese et al.,) propose a proxy re-encryption scheme to secure distributed storage systems and achieve efficient access control among dynamic groups. 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 (Xiong et al., 2012), 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. To mitigate to such concern, CloudaSec presents a flexible revocation procedure, while considering a restricted member list, adapted to resource constrained devices.

Recently, in order to achieve efficient membership revocation system, (Liu et al., 2013) 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 (Seo et al., 2013), 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, similarly to a proxy re-encryption scheme, 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.

7 CONCLUSIONS

The growing need for secure cloud sharing services and the attractive properties of the convergent cryptography lead us to combine them, thus, defining an innovative solution to the data outsourcing security and efficiency issues. In this paper, we design a secure data sharing scheme CloudaSec, for dynamic groups in untrusted cloud storage environments. Our approach ensures the confidentiality of outsourced data in public untrusted cloud servers and defines a smooth group revocation mechanisms. That is, flexible access control policies are enforced among users belonging to separate groups with different privileges.

Our experimental results show the efficiency of CloudaSec in scalable data sharing, while considering the impact of the cryptographic operations at the client side.

REFERENCES

https://github.com/openstack/swift.

Ateniese, G., Fu, K., Green, M., and Hohenberger, S. Improved proxy re-encryption schemes with applications to secure distributed storage. *ACM Trans. Inf. Syst. Secur.*, 9:1–30.

Ben, L. (2007). On the implementation of pairing-based cryptosystems.

Boneh, D. and Boyen, X. (2006). On the impossibility of efficiently combining collision resistant hash functions. In *In Proc. Crypto 06*, pages 570–583.

Burmester, M. and Desmedt, Y. (2005). A secure and scalable group key exchange system. *Inf. Process. Lett.*, 94(3).

Dierks, T. and Rescorla, E. (2008). RFC 5246 - The Transport Layer Security (TLS) Protocol Version 1.2. Technical report.

et al., T. G. (2002). GNU multiple precision arithmetic library 4.1.2.

Fugkeaw, S. (2012). Achieving privacy and security in multi-owner data outsourcing. pages 239–244. IEEE.

Gantz, B. J. and Reinsel, D. (2012). The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east. *IDC iView*, (December):1–16.

Goyal, V., Pandey, O., Sahai, A., and Waters, B. (2006). Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM conference on Computer and communications security*, CCS '06, pages 89–98. ACM.

Liu, X., Zhang, Y., Wang, B., and Yan, J. (2013). Mona: Secure multi-owner data sharing for dynamic groups in the cloud. *IEEE Trans. Parallel Distrib. Syst.*, 24(6).

Ratna, D., Rana, B., and Palash, S. (2004). Pairing-based cryptographic protocols : A survey.

Regan, K. W. Minimum-complexity pairing functions.

Seo, S.-H., Nabeel, M., Ding, X., and Bertino, E. (2013). An efficient certificateless encryption for secure data sharing in public clouds. *IEEE Transactions on Knowledge and Data Engineering*, 99:1.

The OpenSSL Project (2003).

Wang, C., guang Qin, Z., Peng, J., and Wang, J. (2010). A novel encryption scheme for data deduplication system. pages 265–269.

Xiong, H., Zhang, X., Yao, D., Wu, X., and Wen, Y. (2012). Towards end-to-end secure content storage and delivery with public cloud. CODASPY '12, pages 257–266. ACM.

Yu, S., Wang, C., Ren, K., and Lou, W. (2010). Achieving secure, scalable, and fine-grained data access control in cloud computing. INFOCOM'10, pages 534–542.

Zarandioon, S., Yao, D. D., and Ganapathy, V. (2011). K2c: Cryptographic cloud storage with lazy revocation and anonymous access. In *SecureComm*, volume 96, pages 59–76. Springer.

Zhou, L., Varadharajan, V., and Hitchens, M. (2011). Enforcing role-based access control for secure data storage in the cloud. *Comput. J.*, 54.