POMS : (P)roxy (O)ffloading for (M)ulticloud Storage with Keyword (S)earch

Adam Oumar Abdel-Rahman[1], Sofiane Azogagh[2], Zelma Aubin Birba[2], and Arthur Tran Van[1]

[1] Samovar, Télécom SudParis, Institut Polytechnique de Paris, Palaiseau, France
adam_oumar.abdel_rahman@telecom-sudparis.eu
arthur_tran-van@telecom-sudparis.eu
[2] Université du Québec à Montréal, Montréal, Canada
azogagh.sofiane@courrier.uqam.ca birba.zelma_aubin@courrier.uqam.ca

Abstract. Cloud storage offers convenient data access and sharing, but security concerns remain. Existing secure cloud storage solutions often lack essential features like data integrity, multi-cloud support, user-friendly file sharing, and efficient search. This paper proposes a novel secure cloud storage system that addresses these limitations. Our system uses distributed storage and attribute-based encryption to enhance data availability, access control, and user experience. It also enables private and efficient file search and data retrievability verification. This approach overcomes the trade-offs present in prior work, offering a secure and user-friendly solution for cloud data management.

Keywords: Multi-cloud Storage · Attribute Based Encryption · Searchable Encryption · Proof of Retrievability.

1 Introduction

The IT industry has witnessed a drastic rise in the reliance on cloud computing. The cloud offers a wide array of meaningful services to individuals and organisations, ranging from computing and networking to file storage and data processing. In this constantly evolving landscape of cloud services, cloud storage has emerged as one of the key solutions. It allows users to store their data and access it from anywhere at any time.

Classical cloud storage services offer data sharing features, empowering data owners to grant access to other users on their stored data. While traditional cloud storage services provide essential features to their costumers, they often lack the security-centric approach necessary to provide meaningful protection of users' data. Namely, cloud storage service providers potentially have direct access to such data when users rely on their services, which raises serious privacy and confidentiality concerns. Also, various incidents such as hardware failures, network congestion and malicious actions can lead to data loss or corruption.

To overcome limitations in data security, recent research has proposed several secure cloud storage systems. Despite offering data privacy, many of these secure

cloud storage solutions lack essential features like guaranteeing data integrity, working seamlessly with multiple cloud storages, or enabling user file sharing. As cloud usage grows tremendously, allowing users to search for particular files also becomes essential for efficient management of their data. This is particularly important in the context of multi-user systems, where some parties might need the ability to search for files shared by others. Ease of use is another crucial, yet often neglected, factor. Ideally, users shouldn't need to manage a secret key until they want to retrieve their data. Unfortunately, a limited number of solutions address these critical needs for secure, user-friendly cloud storage. Furthermore, they often come with limitations such as requiring a trusted server or a single cloud storage.

Our Contributions. In this paper, we propose a secure cloud storage system that brings back together the essential features of the native cloud storage, while efficiently addressing the security issues outlined above. Our solution abstracts the storage of data across multiple cloud storages by relying on a proxy that interacts with the cloud storages on behalf of the data owner. By strategically distributing the data across multiple cloud storages, we enhance retrievability and ensure resilience against data loss, corruption, or deletion. Our protocol provides a secure way to share data among multiple users, while ensuring that only authorized parties can access the data. It frees the data owners from managing secret keys, allowing for a keyless and more user-friendly experience. Our construction also provides the ability to privately and efficiently search for files stored in the cloud. Additionally, we provide a mechanism to periodically check the retrievability of the stored data. We outline the main contributions of this paper as follows:

Enhanced security of data. We build on various primitives to provide a secure cloud storage system that ensures the confidentiality, integrity, and retrievability of the stored data. Our scheme is secure against honest-but-curious proxies and dishonest cloud storages trying to break privacy. In addition, we provide efficient protection against data corruption attempts by the providers.

Focus on user experience. To bring back the user experience of native cloud storage, we provide a keyless solution, where users do not need to manage secret keys. Data-sharing is made easy to allow efficient collaboration among users. By providing a private and efficient search mechanism, we offer an additional feature that facilitates data management, especially in multi-user systems.

The rest of this paper is organized as follows. In Section 2, we start by introducing the primitives used in our solution. Then, we present the proposed secure cloud storage protocol in Section 3. In Section 4, we provide the necessary information to understand the security of our scheme. In Section 5, we present our proof of concept of our protocol and evaluate its performance. Finally, we provide an overview of related solutions in the literature in Section 6 before concluding the paper.

2 Preliminaries

In this section, we review the main cryptographic primitives used in our protocol. We begin with *attribute-based encryption*, which enables fine-grained access control over encrypted data. Next, we discuss *identity-based signature*, which streamlines the process of verifying digital signatures. We then introduce *searchable encryption*, allowing users to store encrypted data on a server while enabling keyword searches over it. Finally, we cover *proof of retrievability*, a technique that ensures users can retrieve and verify the retrievability of their data stored remotely. The notations used in this paper are summarized in Table 2.

2.1 Attribute-Based Encryption

Attribute-Based Encryption (ABE) is a cryptographic primitive that allows fine-grained access control on encrypted data. In ABE, decryption is possible only if the user's attributes satisfy the access policy.

We focus on Ciphertext-Policy Attribute-Based Encryption (CP-ABE) because the access policy is embedded in the ciphertext, while attributes are associated with a decryption key. This eases data sharing, as a user can specify whom they share the file with directly within the file. Previous work, such as [12, 26], has also highlighted the advantages of this approach.

We first define an *access structure* (a.k.a access policy) and then we present a CP-ABE scheme with decryption outsourcing.

Definition 1 (Access Structure). *Let* $\mathsf{Att} = \{\mathsf{att}_1, \mathsf{att}_2, \ldots, \mathsf{att}_m\}$ *be a finite set of attributes. An access structure over* Att *is a family* $\mathbb{A} \subseteq 2^{\mathsf{Att}} \setminus \{\varnothing\}$*. A set in* $\mathbb{A}$ *is said to be* authorized*; otherwise it is* unauthorized.

Given a set of attributes $\Gamma \subseteq \mathsf{Att}$, we write $\Gamma \in \mathbb{A}$ if and only if there exists $A \subseteq \Gamma$ such that A is authorized.

Ciphertext-Policy Attribute-Based Encryption (CP-ABE). We describe a CP-ABE with decryption outsourcing, such as the one introduced by the scheme of Green *et al.* [11]. The scheme consists of the following algorithms:

$\mathsf{Setup}(1^\lambda) \to (\mathsf{mpk}_{\mathsf{ABE}}, \mathsf{msk}_{\mathsf{ABE}})$: The algorithm takes as input the security parameter λ, it outputs the public parameters $\mathsf{mpk}_{\mathsf{ABE}}$ and a master secret key $\mathsf{msk}_{\mathsf{ABE}}$.

$\mathsf{Encrypt}(\mathsf{mpk}_{\mathsf{ABE}}, M, \mathbb{A}) \to \mathsf{CT}$: The algorithm takes as input the public parameters $\mathsf{mpk}_{\mathsf{ABE}}$, a message M and an access structure $\mathbb{A}$. It outputs a ciphertext CT.

$\mathsf{KeyGen}(\mathsf{msk}_{\mathsf{ABE}}, \Gamma) \to (\mathsf{SK}, \mathsf{TK})$: The algorithm takes as input the master secret key $\mathsf{msk}_{\mathsf{ABE}}$ and an attribute set Γ. It outputs a private key SK and a transformation key TK.

$\mathsf{Transform}(\mathsf{TK}, \mathsf{CT}) \to \mathsf{CT}'$ `or` $\perp$: The algorithm takes as input a transformation key TK for a set of attributes Γ and a ciphertext CT that was encrypted under an access structure $\mathbb{A}$. It outputs the partially decrypted ciphertext CT' if and only if $\Gamma \in \mathbb{A}$ and $\perp$ otherwise.

$\mathsf{Decrypt}(\mathsf{SK}, \mathsf{CT}') \rightarrow M$: The decryption algorithm takes as input a private key SK and a partially decrypted ciphertext CT' that was returned by the $\mathsf{Transform}$ algorithm. It outputs the message M.

2.2 Identity-Based Signature

Shamir first introduced the concept of identity-based cryptography (IBC) in [24]. In an IBC system, a user's public key is derived directly from their identity (e.g., an email address or name), eliminating the need for traditional certificates used in public key infrastructure (PKI). A trusted entity, known as a private key generator (PKG), computes private keys using a master secret and distributes them to users. Since the introduction of IBC, numerous identity-based signature (IBS) schemes have been proposed [10], offering various approaches to build efficient and secure IBS systems.

An IBS scheme typically consists of the following four algorithms:

$\mathsf{Setup}(1^\lambda) \rightarrow (\mathsf{mpk}_{\mathsf{IBS}}, \mathsf{msk}_{\mathsf{IBS}})$: This algorithm takes a security parameter λ and outputs a master public key $\mathsf{mpk}_{\mathsf{IBS}}$ and a master secret key $\mathsf{msk}_{\mathsf{IBS}}$.

$\mathsf{Extract}(\mathsf{msk}_{\mathsf{IBS}}, ID) \rightarrow \mathsf{sk}$: Given the master secret key $\mathsf{msk}_{\mathsf{IBS}}$ and a user identity ID, this algorithm generates a private key sk associated with the identity ID.

$\mathsf{Sign}(\mathsf{sk}, ID, m) \rightarrow \Sigma = (\sigma, ID)$: Using the private key sk, the identity ID, and the message m, this algorithm produces a signature $\Sigma = (\sigma, ID)$.

$\mathsf{Verif}(\mathsf{mpk}_{\mathsf{IBS}}, m, \Sigma = (\sigma, ID)) \rightarrow \{0, 1\}$: This algorithm takes the master public key $\mathsf{mpk}_{\mathsf{IBS}}$, a message m, and a signature Σ. It returns 1 if the signature is `valid` and 0 otherwise.

2.3 Searchable Encryption

Searchable encryption is cryptographic primitive that allows a user to store its data in an encrypted form on a server, while enabling keyword search over the encrypted data. There are two branches of searchable encryption: *Symmetric Searchable Encryption (SSE)* and *Public key Encryption with Keyword Search (PEKS)*. In SSE, only the secret key holder can encrypt data and perform searches, while in PEKS, anyone with the public key can encrypt, but only the private key holder can search. This work focuses on SSE.

Index based SSE scheme. In an index-based SSE scheme, the user first indexes the data by extracting relevant keywords that can be searched later. The user then creates an encrypted index and encrypts the data before sending it to the server. There are two index types: (i) a direct index, which links documents to keywords and is efficient for updates, and (ii) an inverted index, which links keywords to documents and is efficient for search. Since, in our case, users gradually upload files to the server, the direct index is better suited to efficient updates without reindexing.

When searching for a keyword w, the user generates a trapdoor with their private key and sends it to the server. The server then uses the trapdoor to

search and retrieve pointers to the relevant encrypted documents. Formally, a searchable encryption scheme is defined by the following algorithms:

$\mathsf{Setup}(1^\lambda) \to K_{\text{sse}}$: The algorithm takes a security parameter λ and outputs a private key K_{sse}.

$\mathsf{Build}(M, I, K_{\text{sse}}) \to \mathcal{C}_I$: The algorithm takes a message M indexed by a set of keywords I and the private key K_{sse}. It outputs the searchable ciphertext $\mathcal{C}_I$.[3]

$\mathsf{TrapGen}(w, K_{\text{sse}}) \to T_w$: The algorithm takes a keyword w and the private key K. It outputs the trapdoor T_w for w.

$\mathsf{Search}(\mathcal{C}_I, T_w) \to M$ or $\perp$: The algorithm takes a searchable ciphertext $\mathcal{C}_I$ and a trapdoor T_w. It outputs the message M if $w \in I$, and $\perp$ otherwise.

The private key K_{sse} is essential for constructing searchable ciphertext and generating trapdoors. Therefore, in a multi-user setting, this key must be shared among all users [7].

Security of SSE. The security of searchable encryption is typically characterized as the requirement that nothing be leaked beyond the *access pattern* and the *search pattern* [7, 9]. The *access pattern* refers to the result of a query search, which is the set of documents that contain a keyword. If the server returns the same document set for the same query search, then the *access pattern* is leaked. The *search pattern*, in other hand, reveals whether the same search was performed in the past or not.

2.4 Proof of Retrievability

The Proof of Retrievability (PoR) is a cryptographic primitive that allows an entity to verify the retrievability and retrieve data stored in the cloud over time without requiring a data download. It does this by challenging servers to prove that the data is retrievable. PoR schemes are resilient to partial data corruption or deletion. PoR schemes encompass a variety of techniques and contexts, as outlined in the comprehensive survey by Tan *et al.* [27].

To enhance data retrievability, we focus only on PoR schemes using several cloud storages. In our protocol, all responsibilities for the PoR scheme are delegated to the proxy. Thus, only the proxy has to verify data retrievability. Schemes where only one party is allowed to verify retrievability are referred to as private PoR schemes.

The delegation is motivated by the fact that the user may not always have regular access to the cloud. It also helps reduce the user's computational load. Since the proxy is assumed to be honest but curious, it poses no security threats.

A private PoR scheme run by the proxy can be described by the primitives:

$\mathsf{KeyGen}(1^\lambda) \to sk$: Using the security parameters λ the proxy generates a secret key sk.

[3] The message M corresponds to the unique identifier of the document containing the keyword w.

$\mathsf{Encode}(sk, f) \to s_1, \ldots, s_\ell$: The proxy processes the original file f into f'[4]. Then f' is split into shares $s_1, \ldots, s_\ell$ where ℓ is the number of servers.

$\mathsf{Check}(sk, \mathrm{id}) \to (\top, \bot)$: The proxy generates a challenge c for file identifier id using sk and sends it to the servers. Each server computes a response r and sends it to the the proxy. Finally, the proxy can verify if the file identified by id is retrievable.

$\mathsf{Retrieve}(\mathrm{id}) \to f$: The proxy retrieves the file f identified by id.

3 Our Construction

3.1 Protocol Overview

The protocol involves four key entities: a *trusted authority*, a *proxy*, *cloud storages* and *users*. The trusted authority is responsible for generating and distributing public keys, while users authenticate with the authority to obtain their private keys. The proxy manages data storage across cloud storages and verifies data retrievability. The cloud storages store the data and respond to the proxy's challenges during the proof of retrievability phase. Users include both data owners, who upload their data, and data users, who download it. Figure 1 illustrates the interactions and roles of these entities within the protocol.

Our protocol securely stores data across ℓ cloud storages, enabling authorized users to download it, even if they did not initially upload it. Users are not required to store cryptographic material to upload or retrieve data.

The protocol consists of the following algorithms:

$\mathsf{Setup}(1^\lambda)$: The algorithm takes a security parameter λ and generates the cryptographic parameters for the system.

$\mathsf{KeyGen}(ID, \Gamma)$: The trusted authority generates a signature key sk using the user's ID, and a decryption key dk using the user's attributes Γ. The signature key is required for the upload operation, while the decryption key is needed for the download operation. The user requests the corresponding key before uploading or downloading a file.

$\mathsf{Upload}(f, \mathcal{I}, \mathbb{A})$: The user uploads a file f, associated with a set of keywords $\mathcal{I}$ and an access policy $\mathbb{A}$. The file is uploaded to a proxy, which forwards it to cloud storages. The file can be retrieved using the keywords in $\mathcal{I}$, and only users whose attributes satisfy the access policy $\mathbb{A}$ are allowed to download it.

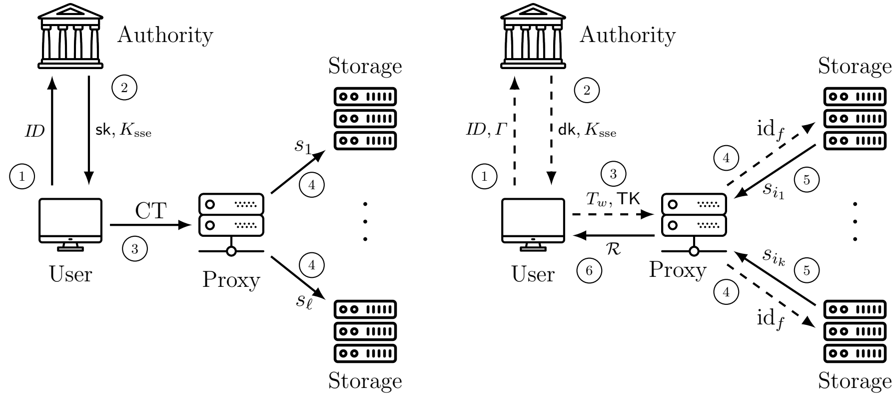
$\mathsf{Download}(w)$: This algorithm takes a keyword w and retrieves all files associated with that keyword that have been shared with the current user.

$\mathsf{RetrievabilityCheck}(\mathrm{id}_f)$: The proxy verifies if the file f identified by id_f is retrievable.

3.2 Detailed Protocol

In the following sections, we describe in details the setup, key generation, upload, download, and proof of retrievability phases of the protocol.

[4] This can be done using an error-correcting code such as Reed-Solomon codes.



(a) Upload: When uploading a file, the user authenticates himself to the authority to obtain their private key. Using this key, they generate the ciphertext and send it to the proxy, which processes the data and distributes shares to cloud storages.

(b) Download: To download files matching keyword w, the user authenticates with the authority to obtain their private key $dk = (\mathsf{SK}, \mathsf{TK}), K_{\text{sse}}$. The user sends a trapdoor for w and transformation key (T_w, TK) to the proxy, which handles the search, partial decryption, and file retrieval from cloud storages. The user finalizes the decryption of the files using SK.

Fig. 1: Information exchange during the upload and download phases of the protocol.

Setup. This algorithm is executed by the trusted authority and proceeds as follows:

- Execute the setup algorithms for both the ABE and IBS schemes to generate the public parameters and master secret keys, $(\mathsf{mpk}_{\mathsf{ABE}}, \mathsf{msk}_{\mathsf{ABE}})$ and $(\mathsf{mpk}_{\mathsf{IBS}}, \mathsf{msk}_{\mathsf{IBS}})$.
- Generate a random symmetric key K_{sse} for the searchable encryption scheme.
- Select a collision-resistant hash function H.
- Select a symmetric encryption scheme SKE.

The authority keeps the master secret keys $\mathsf{msk}_{\mathsf{ABE}}$ and $\mathsf{msk}_{\mathsf{IBS}}$ secret. It shares the symmetric key K_{sse} with users upon their request for private keys. The public parameters $\{\mathsf{mpk}_{\mathsf{ABE}}, \mathsf{mpk}_{\mathsf{IBS}}, H, \mathsf{SKE}\}$ are made publicly available.

Key Generation. The authority runs this algorithm to generate the private keys for a user $\mathcal{U}$. The algorithm requires the master secret key $\mathsf{msk}_{\mathsf{ABE}}$, the user's identity ID, and the user's attributes Γ. The process involves the following steps:

- Generate the decryption key $\mathsf{dk} = \{\mathsf{SK}, \mathsf{TK}\}$ by invoking the algorithm $\mathsf{ABE.KeyGen}(\mathsf{msk}_{\mathsf{ABE}}, \Gamma)$ of the ABE scheme.
- Generate the user private key sk by invoking the $\mathsf{IBS.Extract}(\mathsf{msk}_{\mathsf{IBS}}, ID)$ algorithm of the IBS scheme.

The authority outputs the user's private key as $\{\mathsf{dk}, \mathsf{sk}, K_{\text{sse}}\}$ and shares the searchable encryption key K_{sse} with the user.

Note that users are not required to store their private keys. They can generate them on-demand based on their needs. Additionally, they can request only their signature keys or decryption key from the authority, depending on whether they want to upload or download a file.

Upload. The algorithm consists of two steps. First, on the user side, the user encrypts his data and generates the secure indexes. Then, the user sends the encrypted data to the proxy for storage. The second step is executed by the proxy. It invokes the PoR protocol to encode the encrypted data into shares and stores them among the cloud storages. The proxy stores some relevant informations in its database.

User Side. Consider a user $\mathcal{U}$ with a unique identifier ID, who wishes to upload a file f associated with a set of keywords I. We denote id_f the unique identifier for the file f. The user shares this file with a list of users which is specified by an access policy $\mathbb{A}$. Then the user proceeds with the following steps:

- Requests the authority for their signature key sk and the searchable encryption key K_{sse} (① and ② in Figure 1a).
- Generates a random symmetric key K and computes the encryption $C_f = \mathsf{SKE.Encrypt}(f, K)$.
- Encrypts the symmetric key K using ABE encryption algorithm and the access policy $\mathbb{A}$: $\mathsf{CT}_{\mathbb{A}} = \mathsf{ABE.Encrypt}(\mathsf{mpk}_{\mathsf{ABE}}, K, \mathbb{A})$.
- Computes the searchable ciphertext $\mathcal{C}_I = \mathsf{SSE.Build}(M, I, K_{\mathrm{sse}})$, where the message $M = r \| \mathrm{id}_f$, is a concatenation of a random bitstring r and the file identifier id_f.
- Computes the signature $\Sigma = \mathsf{IBS.Sign}(\mathsf{sk}, ID, m)$, where the message m is obtained by first computing $m_0 = H(C_f \| r)$ and then $m = H(m_0 \| K)$.

The user sends the ciphertext $\mathsf{CT} = \left\{(\mathsf{CT}_{\mathbb{A}}, \mathcal{C}_I, \Sigma), C_f\right\}$ to the proxy for storage (③ in Figure 1a).

Proxy Side. Upon receiving a ciphertext CT, the proxy performs the following actions:

- Parses the ciphertext $\mathsf{CT} = \left\{(\mathsf{CT}_{\mathbb{A}}, \mathcal{C}_I, \Sigma), C_f\right\}$ and stores in its database the ABE ciphertext, the searchable ciphertext and the signature $(\mathsf{CT}_{\mathbb{A}}, \mathcal{C}_I, \Sigma)$.
- Executes the setup algorithm of PoR, if not executed before, to generate the public and secret parameters: $sk_{\mathsf{PoR}} = \mathsf{PoR.KeyGen}(1^{\lambda})$.
- Applies the encode algorithm of PoR to convert the encrypted file into shares: $s_1, \ldots, s_\ell = \mathsf{PoR.Encode}(sk_{\mathsf{PoR}}, C_f)$.

Subsequently, the proxy dispatches the share s_i to the corresponding i-th cloud storage (④ in Figure 1a).

Download. Similar to the upload phase, the download phase also consists of two steps. Initially, the user executes a query search for files containing a specific keyword, ensuring they have the necessary access rights. Subsequently, the proxy retrieves the encrypted files from the cloud storages and forwards them to the user. The user then verifies the signature and decrypts the files if the signature is valid.

User Side. Consider a user who wishes to download files containing a specific keyword w and possesses the necessary access rights. The user proceeds as follows:

– Requests the symmetric key for searchable encryption K_{sse} and their decryption key $\mathsf{dk} = \{\mathsf{SK}, \mathsf{TK}\}$ from the authority (① and ② in Figure 1b).
– Generates the trapdoor $T_w = \mathsf{SSE.TrapGen}(w, K_{\mathrm{sse}})$.

The user then keeps SK secret and submits a query search by sending $Q = (T_w, \mathsf{TK})$, the trapdoor and the transformation key, to the proxy (③ in Figure 1b).

Proxy Side. Upon receiving a search query $Q = (T_w, \mathsf{TK})$, the proxy processes each entry $(\mathsf{CT}_{\mathbb{A}}, \mathcal{C}_I, \Sigma)$ in the database as follows:

– Computes $M = \mathsf{SSE.Search}(\mathcal{C}_I, T_w)$. If $M \neq \perp$, it retrieves the random bitstring and the identifier of the file containing the word w, denoted by $M = r \| \mathrm{id}_f$. Otherwise, it proceeds to the next entry.
– Invokes the ABE transformation algorithm to partially decrypt the ciphertext $\mathsf{CT}_{\mathbb{A}}$. If $\mathsf{CT}'_{\mathbb{A}} = \mathsf{ABE.Transform}(\mathsf{TK}, \mathsf{CT}_{\mathbb{A}})$ is successful, then it executes the PoR protocol to retrieve the corresponding encrypted file $C_f = \mathsf{PoR.Retrieve}(\mathrm{id}_f)$ (④ and ⑤ in Figure 1b). Otherwise, it continues to the next entry.
– Computes the first part of the signature message $m_0 = H(C_f \| r)$ and appends the tuple $(\mathsf{CT}'_{\mathbb{A}}, m_0, \Sigma, C_f)$ to the set of results $\mathcal{R}$.

Finally the proxy sends the result set $\mathcal{R}$ to the user (⑥ in Figure 1b).

User Side. Upon receiving the set of results $\mathcal{R}$, the user processes each tuple $(\mathsf{CT}'_{\mathbb{A}}, m_0, \Sigma, C_f)$ in the following manner:

– Decrypts the transformation ciphertext CT' to retrieve the encapsulated key: $K = \mathsf{ABE.Decrypt}(\mathsf{CT}', \mathsf{SK})$.
– Constructs the signature message $m = H(m_0 \| K)$ and verifies the signature Σ using $\mathsf{IBS.Verif}(\mathsf{mpk}_{\mathsf{IBS}}, m, \Sigma)$.
– If the signature is `valid`, the user decrypts the encrypted file C_f to obtain $f = \mathsf{SKE.Decrypt}(C_f, K)$.

This completes the user's process of retrieving and validating the desired files.

Retrievability Check. This phase focuses on verifying the retrievability of stored data, without involving the user. The proxy runs the Check algorithm on the file identified by id_f using its PoR private key sk_{PoR}.

Proxy Side. The proxy uses Check with the given id_f.

- It uses sk_{PoR} and id_f to generate a challenge c.
- Sends the challenge and file identifier to the cloud storages.
- Receive the responses $(r_1, \ldots, r_\ell)$ from the cloud storages.
- Verifies whether the file is retrievable based on the responses.

Storage Side. The cloud storages wait for challenges and respond accordingly.

- Receive a challenge c along with the file identifier id_f
- Reply to the challenge with r.

The proxy can run this algorithm at any time and must perform it regularly to ensure data retrievability.

4 Security Proofs

4.1 Threat Model

In this paper, we assume two types of adversaries: an honest-but-curious proxy and dishonest cloud storages. The honest-but-curious proxy is assumed to follow the protocol, but tries to learn as much information as possible from the data they handle. This is a reasonable assumption in real world, as a proxy, often a company, risks its reputation if caught acting maliciously. Some cloud storage providers may be dishonest and are assumed to deviate from the protocol by altering the data they store. If all of them do so, the privacy property remains intact, but the user won't be able to retrieve their data. We also consider the case of a colluding adversary, where a set of dishonest cloud storages and an honest-but-curious proxy work together to learn more information about the stored data. Users, on the other hand, are assumed to be honest.

Communications between the different entities are assumed to be performed over secure channels, so that external adversaries cannot eavesdrop on the data.

4.2 Security Analysis

In this section, we provide a security analysis of our proposed secure cloud storage system. We first define the security properties we aim to achieve, and then we provide a proof of security for our system.

Security Properties Our secure cloud storage system aims to provide the following security properties:

Privacy: The data stored in the cloud should remain confidential. Only authorized users should be able to access the data. Moreover, the search mechanism should not reveal any information about the stored data to unauthorized users.

Integrity: The data stored in the cloud should remain intact. Attempts to alter the data should be detected.

Retrievability: The definition of retrievability property varies among PoR schemes. In general, there are two types of security properties: the first focuses on resilience against pollution or corruption attacks, as in Network Coding - Dispersal Coding PoR (ND-POR) [19], while the second ensures that data can be retrieved as long as a certain fraction of verification challenges are successful, as introduced in the Compact Proof of Retrievability scheme [22].

Proof of Security We prove the security of our system by showing that it satisfies the security properties defined above.

Privacy: when a user outsources the storage of a file to the multi-cloud using our system, a search index is first built on the file, and the file is then encrypted using the symmetric encapsulated key K. The generic construction provided in [31] is such that no polynomial time adversary can decrypt a ciphertext without satisfying the access policy (see Theorems 3.1, 5.2, 6.2). This ensures that the proxy (or colluding cloud storages) cannot learn the stored data.

When the proxy receives the ciphertext, it parses it to extract the ABE ciphertext CT, the symmetric encryption of the file C_f, the secure search index $\mathcal{C}_I$ and the signature Σ. By the security of the ABE scheme, the proxy cannot learn anything about the file from CT. The symmetric encryption and the signature also do not reveal any informations about the file. Therefore, the data handled by the proxy remains confidential.

After the proxy invokes the encoding algorithm of the PoR scheme, the generated shares are send to the cloud storages. This encoded data is a ciphertext created under a key unknown to both the cloud storages and the proxy. Therefore, recovering it, even through collusion, would be pointless for the cloud storages as they cannot decrypt it. Even if all the cloud storages are dishonest, they would not be able to access the data. In such a case, the user would not be able to retrieve the original file, but the data would remain confidential.

A similar analysis can be made for the operations performed during the download phase, to conclude that our scheme provides privacy.

Integrity/Authenticity: The integrity of the stored data is ensured by the IBS scheme used to sign the data. The signature Σ is computed on the file f, and verified by the user when downloading the file, to ensure that it has not been altered by the cloud storages. Morevover, the secure keyword search scheme garantees that the results returned to the user are indeed the files that match the search query, providing authenticity.

Retrievability: The proxy is assumed to be honest-but-curious. This implies that, although the proxy may observe and collect data, it follows the protocol correctly. Consequently, if the proxy retrieves the encrypted file, the user will successfully retrieve the file as well.

Thus, to ensure retrievability, it is sufficient to demonstrate that the proxy, when following the Proof of Retrievability (PoR) scheme, adheres to the PoR security properties. This guarantee is inherent to the PoR scheme itself. There-

fore, the level of retrievability we achieve is equivalent to that guaranteed by the underlying PoR scheme used.

The proxy remains a single point of failure in our protocol. However, only minimal information is stored on the proxy. Therefore, regular backups of the metadata stored by the proxy can help prevent its failure.

5 Implementation

We implemented a Proof of Concept (PoC) of our protocol. The project is available at https://anonymous.4open.science/r/poms-8040/README.md[5].

5.1 Proof of Concept

This implementation is built upon the widely recognized RELIC cryptography library [1], which provides the cryptographic primitives necessary for secure and efficient operation. Symmetric primitives come from OpenSSL. The PoC is a library written in C++. Moreover, our project includes a comprehensive example of the protocol.

We use Docker [17] to enhance portability and ease to reproduce. Actors of the protocol are deployed with a script that handles all different parameters. Then we can run as many users as wanted. Each user is a CLI program allowing upload and download on the file. The example uses the user's name as an attribute. While our library supports various types of attributes, using the name simplifies the example.

This approach not only enhances the reproducibility of our results but also simplifies the setup process for users who wish to explore or build upon our work. The use of Docker ensures that all dependencies are encapsulated, allowing for a consistent execution environment across various systems.

5.2 Cryptographic Scheme Selection

To create the PoC, we had to select one scheme for each requirement of the protocol.

The PoR scheme we chose is the private version of Compact Proof of Retrievability (CPoR) [22]. Tan et al. [27] propose several private schemes utilizing various storage methods, with CPoR highlighted as a reference model. However, public implementations were lacking, except for the CPoR scheme. It was helpful to have some sources of inspiration, although we could not use the available code as a dependency.

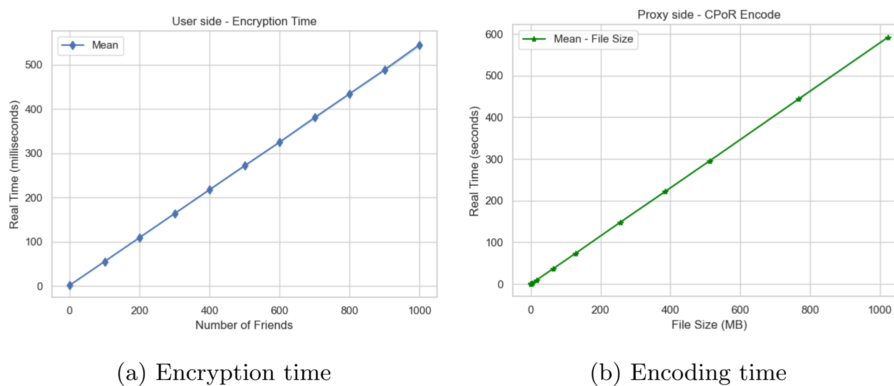
Regarding the Attribute-Based Encryption (ABE) scheme, there are several alternatives available [38]. We chose the CP-ABE scheme with outsourcing introduced by Green *et al.* [11]. We instancied the scheme using an asymmetric pairing-friendly setting.

[5] Anonymous git is employed for the review. If the paper is accepted, the final project will be released publicly on GitHub.

For the Identity-Based Signature (IBS) scheme, many existing constructions rely on bilinear pairing [10], which are computationally expensive for the user. We opted for the scheme proposed by Bellare *et al.* [3], which is also based on elliptic curves and avoids the need for pairings.

Finally, for the searchable encryption scheme, we prioritize simplicity and efficiency while the only restriction was a scheme using forward indexes. Consequently, we selected the symmetric scheme proposed by Waters *et al.* [32].

We prioritize simplicity in implementation as our goal was to deliver a Proof of Concept and minimize development time. As a result, performance could be further optimized.



(a) Encryption time

(b) Encoding time

Fig. 2: Upload phase: coefficient of determination for linear regression R^2 are above 0.9999

5.3 Performance

We evaluated the performance of our proof of concept (PoC) in terms of storage and computation. We conducted a series of microbenchmarks using Google benchmark library to measure the time taken by the different operations. The experiments were conducted on an Ubuntu 22.04.05 system with a 12-core Intel(R) Core(TM) i5-11500H CPU running at 2.90GHz and 32GB of RAM.

Implementation details. For our implementation, we use the BLS12-381 elliptic curve [2] for both the ABE and IBS schemes. We chose the BLS12-381 curve (which is pairing-friendly) due to its widespread use in projects like Zcash and Ethereum 2.0, its proven security, and its alignment with IETF recommendations for 128-bit security [21]. For symmetric primitives, we use AES-256-GCM for encryption and SHA-256 as the hash function.

Computation. The computations on the storage involve simple calculations. Given the computational power of cloud storages, we focus on the client and proxy. As Figure 2a shows, the client's computation grows linearly with the

number of friends the file is shared with. Furthermore, the computations are very fast and can be performed on any computer in an acceptable amount of time, even when sharing the file with an entire organization.

In contrast, the proxy performs heavier computations due to the PoR scheme, which scales linearly with file size (see Figure 2b). Performance can be improved by using more modern PoR schemes that do not rely on the error-correcting codes responsible for the slowdown.

The proxy also processes the search operation, and we assessed its performance using a dataset of 10,300 files. When no files match the trapdoor, the average search time is about 4754 ms. If some match but none meet the policy, it increases slightly to 5151 ms. If at least one file satisfies the policy, the time increases to 5196 ms due to partial decryption.

Finally, the user's overall process, including ABE decryption and signature verification, is highly efficient, with a mean time of only 3.66 ms. These results underscore the system's ability to manage search and decryption tasks effectively, even with large datasets. Delegating computation to the proxy significantly reduces the user's computational load, allowing users to remain lightweight while only the proxy needs to be powerful.

Storage. We focus on the proxy and cloud storages because the client does not store any information. The proxy's storage requirement increases linearly with the number of people the file is shared with and it does not depend on the file size as highlighted by Figure 3a. Therefore, storage on the proxy side remains lightweight.

Table 1: Upload – Proxy side (PoR)

File Size in MB	1	256	1024
Storage overhead in MB	0.214 (21%)	59 (23%)	238 (23%)
Time	0.58 s	2 min 28s	9 min 52s

Regarding storage, the required storage size grows linearly with the file size as highlighted by Table 1, which was made for resilience to up to 9% data deletion or corruption. Most of the storage overhead is due to the error-correcting codes used by the PoR scheme. While the overhead in the current benchmark is not too significant, it could increase substantially if we aim to recover the file despite heavy corruption or errors. More recent PoR schemes reduce this overhead but they do not fully resolve the issue.

The performance aligns well with the presented use case, enabling a lightweight user experience supported by a more powerful proxy. Although the PoR scheme adds time for protocol operations and storage overhead on cloud servers, these can be seen as opportunities for optimization.

6 Related Work

Shen et al. [25] proposed a protocol involving three parties (Data Owner (DO), Cloud Service Provider (CSP), and Trusted-Party Auditor (TPA)), enabling the data owner to verify the integrity of a part of the file while maintaining privacy.

Yet, Li et al. [14] identified flaws in Shen et al.'s protocol, questioning its privacy guarantees. Similarly, Zhan et al. [37] presented an identity-based protocol with a Trusted-Party Auditor, claiming resistance to three types of attacks introduced by Yang et al. [34]. The *replace attack*, where adversaries attempt to replace a data block with another valid, uncorrupted block to pass TPA's audit. The *forgery attack*, where adversaries forge TPA's audit proof targeting the soundness of the auditing process. The *replay attack*, where adversaries replay a previously valid proof in an attempt to pass TPA's audit.

Regarding searchable encryption, a solution based on Searchable Symmetric Encryption is proposed in [30], capable of performing keyword search and controlled updates in a multi-user context without a shared secret key. However, a key feature absent in their solution is the controlled updates with partial data recovery while maintaining integrity. Sun et al. [26] introduced attribute-based keyword search with efficient revocation. However, their protocol is designed for a single server scenario where multiple data owners permit multiple users to search their outsourced data. In contrast, Li et al. [13] proposed a mechanism using erasure code to enhance the reliability of outsourced searchable encrypted data, which is more relevant to our multi-cloud setting.

For the multi-cloud environment side, significant research has been conducted to address its unique challenges [4,5,8,15,18,28,29,33,35]. DepSky, introduced by [4], mitigates the limitations of individual clouds through a combination of Byzantine quorum protocols [16], secret sharing [23] composed with erasure codes [20], and the use of multiple clouds to ensure diversity and robustness. However, DepSky requires a maximum of two communication round-trips per operation and typically stores only approximately half of the data in each cloud. There are several approach like [8,29] focused on the encryption algorithm of the data storage in the multi-cloud environment, but the integrity of the outsourced data is not guaranteed. In [15], the authors introduced an enhanced data fragmentation framework for multi-cloud storage environments, focusing on data privacy and consistency. This framework efficiently manages data fragments by dynamically updating and resolving conflicts when a data is already existing in the multi-cloud.

One of the initial schemes for cloud storage utilizing attribute-based encryption was introduced in [6]. In this approach, a trusted authority generates the master keys for the system and issues private keys to each user. This scheme allows semi-authorized users to collaborate and access encrypted data. However, it is not appropriate for multi-cloud environments as it only considers a single cloud in the protocol. Additionally, in [36], the authors proposed a fine-grained authorized keyword secure search scheme using ciphertext policy attribute-based encryption (CP-ABE). This scheme supports privacy-preserving keyword searches over encrypted data. Specifically, the data owner build secure indexes for enabling the data users to search over encrypted data. However, in addition to the fact that it's not suitable for multi-cloud, communication between the data owner and the data user is required for authentication.

7 Conclusion

In this paper, we designed and implemented a secure multi-cloud storage system that incorporates keyword search. This system enables data owners to upload files and authorizes data users to download files that match their attributes in a secure and verifiable manner.

To achieve this, we leveraged several advanced cryptographic techniques, such as attribute-based encryption to control access to the files, symmetric searchable encryption to allow data users to search for the files, proof of retrievability to ensure the files' retrievability to the delegated proxy, and identity-based encryption to verify the integrity of the files.

To the best of our knowledge, ours is the first to provide all these features together in a unified framework compared to existing systems. Our system is not only effective and robust but also prioritizes user privacy. We have implemented a user-friendly proof-of-concept that demonstrates the system's capabilities under the honest-but-curious adversary model. Since our protocol is built on top of well-established cryptographic primitives, the security of our system is as strong as the security of these underlying cryptographic primitives. Therefore, it guarantees a comfortable user experience.

In the future, we plan to extend our system to support additional features, such as handling file modifications, enabling verifiable deletions, and preventing file duplication across cloud storages.

References

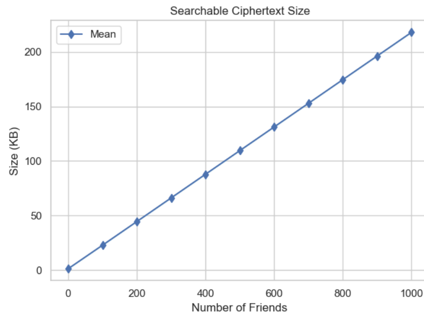
1. Aranha, D.F., Gouvêa, C.P.L., Markmann, T., Wahby, R.S., Liao, K.: RELIC is an Efficient LIbrary for Cryptography. `https://github.com/relic-toolkit/relic`
2. Barreto, P.S., Lynn, B., Scott, M.: Constructing elliptic curves with prescribed embedding degrees. In: Security in Communication Networks: Third International Conference, SCN 2002 Amalfi, Italy, September 11–13, 2002 Revised Papers 3. pp. 257–267. Springer (2003)
3. Bellare, M., Namprempre, C., Neven, G.: Security proofs for identity-based identification and signature schemes. Journal of Cryptology **22**(1), 1–61 (2009)
4. Bessani, A., Correia, M., Quaresma, B., André, F., Sousa, P.: Depsky: dependable and secure storage in a cloud-of-clouds. ACM Transactions on Storage (TOS) **9**(4), 1–33 (2013)
5. Chang, J., Shao, B., Ji, Y., Bian, G.: Efficient identity-based provable multi-copy data possession in multi-cloud storage, revisited. IEEE Communications Letters **24**(12), 2723–2727 (2020). `https://doi.org/10.1109/LCOMM.2020.3013280`
6. Chen, N., Li, J., Zhang, Y., Guo, Y.: Efficient cp-abe scheme with shared decryption in cloud storage. IEEE Transactions on Computers **71**(1), 175–184 (2020)
7. Curtmola, R., Garay, J., Kamara, S., Ostrovsky, R.: Searchable symmetric encryption: improved definitions and efficient constructions. In: Proceedings of the 13th ACM conference on Computer and communications security. pp. 79–88 (2006)
8. Ghanmi, H., Hajlaoui, N., Touati, H., Hadded, M., Muhlethaler, P.: A secure data storage in multi-cloud architecture using blowfish encryption algorithm. In: International Conference on Advanced Information Networking and Applications. pp. 398–408. Springer (2022)

9. Goh, E.J.: Secure indexes. Cryptology ePrint Archive, Paper 2003/216 (2003), https://eprint.iacr.org/2003/216, https://eprint.iacr.org/2003/216
10. Gorantla, M.C., Gangishetti, R., Saxena, A.: A survey on id-based cryptographic primitives. Cryptology ePrint Archive (2005)
11. Green, M., Hohenberger, S., Waters, B.: Outsourcing the decryption of {ABE} ciphertexts. In: 20th USENIX security symposium (USENIX Security 11) (2011)
12. Hur, J.: Attribute-based secure data sharing with hidden policies in smart grid. IEEE Transactions on Parallel and Distributed Systems **24**(11), 2171–2180 (2013)
13. Li, J., Lin, D., Squicciarini, A.C., Li, J., Jia, C.: Towards privacy-preserving storage and retrieval in multiple clouds. IEEE Transactions on Cloud Co mputing **5**(3), 499–509 (2015)
14. Li, X., Liu, S., Lu, R.: Comments on 'a public auditing protocol with novel dynamic structure for cloud data' (2020)
15. Loh, R., Thing, V.L.L.: Data privacy in multi-cloud: An enhanced data fragmentation framework. In: 2021 18th International Conference on Privacy, Security and Trust (PST). pp. 1–5 (2021). https://doi.org/10.1109/PST52912.2021.9647746
16. Malkhi, D., Reiter, M.: Byzantine quorum systems. Distributed computing **11**(4), 203–213 (1998)
17. Merkel, D.: Docker: lightweight linux containers for consistent development and deployment. Linux J. **2014**(239) (mar 2014)
18. Miao, Y., Huang, Q., Xiao, M., Susilo, W.: Blockchain assisted multi-copy provable data possession with faults localization in multi-cloud storage. IEEE Transactions on Information Forensics and Security **17**, 3663–3676 (2022). https://doi.org/10.1109/TIFS.2022.3211642
19. Omote, K., Tran, P.T.: ND-POR: A POR Based on Network Coding and Dispersal Coding. IEICE Trans. Inf. & Syst. **E98.D**(8), 1465–1476 (2015). https://doi.org/10.1587/transinf.2015EDP7011, https://www.jstage.jst.go.jp/article/transinf/E98.D/8/E98.D_2015EDP7011/_article
20. Rabin, M.O.: Efficient dispersal of information for security, load balancing, and fault tolerance. Journal of the ACM (JACM) **36**(2), 335–348 (1989)
21. Sakemi, Y., Kobayashi, T., Saito, T., Wahby, R.: Pairing-friendly curves, 2021. IETF draft
22. Shacham, H., Waters, B.: Compact Proofs of Retrievability. In: Pieprzyk, J. (ed.) Advances in Cryptology - ASIACRYPT 2008, vol. 5350, pp. 90–107. Springer Berlin Heidelberg, Berlin, Heidelberg (2008), http://link.springer.com/10.1007/978-3-540-89255-7{_}7, series Title: Lecture Notes in Computer Science
23. Shamir, A.: How to share a secret. Communications of the ACM **22**(11), 612–613 (1979)
24. Shamir, A.: Identity-based cryptosystems and signature schemes. In: Advances in Cryptology: Proceedings of CRYPTO 84 4. pp. 47–53. Springer (1985)
25. Shen, J., Shen, J., Chen, X., Huang, X., Susilo, W.: An efficient public auditing protocol with novel dynamic structure for cloud data. IEEE Transactions on Information Forensics and Security **12**, 2402–2415 (2017)
26. Sun, W., Yu, S., Lou, W., Hou, Y.T., Li, H.: Protecting your right: Verifiable attribute-based keyword search with fine-grained owner-enforced search authorization in the cloud. IEEE Transactions on Parallel and Distributed Systems **27**(4), 1187–1198 (2014)
27. Tan, C.B., Hijazi, M.H.A., Lim, Y., Gani, A.: A survey on Proof of Retrievability for cloud data integrity and availability: Cloud storage state-of-the-art,

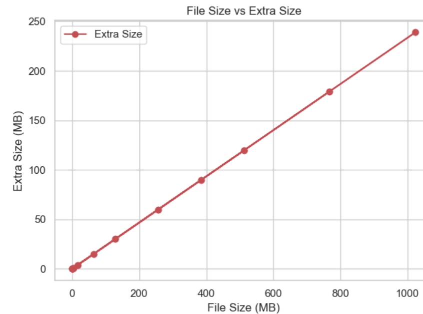
issues, solutions and future trends. Journal of Network and Computer Applications **110**, 75–86 (May 2018). https://doi.org/10.1016/j.jnca.2018.03.017, https://linkinghub.elsevier.com/retrieve/pii/S1084804518301048
28. Tchernykh, A., Miranda-López, V., Babenko, M., Armenta-Cano, F., Radchenko, G., Drozdov, A.Y., Avetisyan, A.: Performance evaluation of secret sharing schemes with data recovery in secured and reliable heterogeneous multi-cloud storage. Cluster Computing **22**(4), 1173–1185 (2019)
29. Viswanath, G., Krishna, P.V.: Hybrid encryption framework for securing big data storage in multi-cloud environment. Evolutionary Intelligence **14**(2), 691–698 (2021)
30. Wang, J., Zhang, R., Li, J., Xiao, Y., Ma, H.: Seupdate: Secure encrypted data update for multi-user environments. IEEE Transactions on Dependable and Secure Computing **19**(6), 3592–3606 (2022)
31. Waters, B.: Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. In: International workshop on public key cryptography. pp. 53–70. Springer (2011)
32. Waters, B.R., Balfanz, D., Durfee, G., Smetters, D.K.: Building an encrypted and searchable audit log. In: NDSS. vol. 4, pp. 5–6 (2004)
33. Yang, J., Zhu, H., Liu, T.: Secure and economical multi-cloud storage policy with nsga-ii-c. Applied Soft Computing **83**, 105649 (2019). https://doi.org/10.1016/j.asoc.2019.105649, https://www.sciencedirect.com/science/article/pii/S1568494619304296
34. Yang, K., Jia, X.: An efficient and secure dynamic auditing protocol for data storage in cloud computing. IEEE transactions on parallel and distributed systems **24**(9), 1717–1726 (2012)
35. Yang, X., Pei, X., Wang, M., Li, T., Wang, C.: Multi-replica and multi-cloud data public audit scheme based on blockchain. IEEE Access **8**, 144809–144822 (2020). https://doi.org/10.1109/ACCESS.2020.3014510
36. Yin, H., Qin, Z., Zhang, J., Deng, H., Li, F., Li, K.: A fine-grained authorized keyword secure search scheme with efficient search permission update in cloud computing. Journal of Parallel and Distributed Computing **135**, 56–69 (2020). https://doi.org/10.1016/j.jpdc.2019.09.011, https://www.sciencedirect.com/science/article/pii/S0743731519303004
37. Zhang, H., Yuan, Y., Xin, X., Qu, Y.: Identity-based integrity verification and public auditing scheme in cloud storage system against malicious auditors. Tehnički vjesnik **30**(2), 408–415 (2023)
38. Zhang, Y., Deng, R.H., Xu, S., Sun, J., Li, Q., Zheng, D.: Attribute-based encryption for cloud computing access control: A survey. ACM Computing Surveys (CSUR) **53**(4), 1–41 (2020)

A Appendices

A.1 Benchmarking results

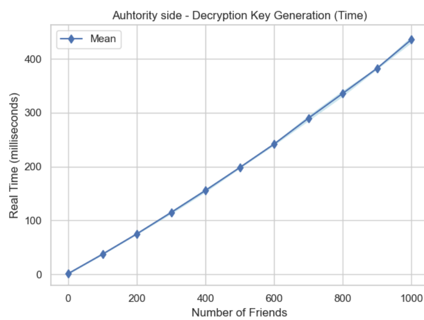


(a) Searchable ciphertext size. This is the size of the ciphertext stored on the proxy, and used for secure keyword search. It grows linearly with the number of friends in the access structure.

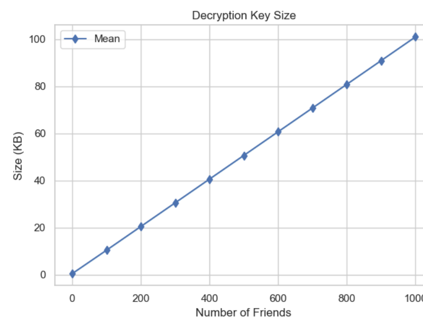


(b) Storage overhead. As explained in Section 5, the storage overhead is primarily due to the CPoR scheme that ensures retrievability by using error-correcting codes for resilience against data loss. It grows linearly with the size of the original file.

Fig. 3: Storage cost. We report the storage cost of the system, in terms of the size of the searchable ciphertext stored on the proxy and the storage overhead on the cloud providers.



(a) Time taken by the authority to generate decryption keys for users, which grows with the number of users the file is shared with.



(b) Size of the decryption key generated by the authority and sent to the user. Like the generation time, this size grows with the number of users the file is shared with.

Fig. 4: Decryption key generation. The authority generates decryption keys which are then sent to the users. We present the time taken by the authority to generate these keys and their size with respect to the number of users the file is shared with.

A.2 Notations

Table 2: Notations and descriptions.

Notation	Description
λ	Security parameter
ℓ	number of storages
mpk	Master public key
msk	Master secret key
$\mathbb{A}$	Access structure
$\varGamma$	Set of attributes
$\mathsf{CT}_{\mathbb{A}}$	Ciphertext with access structure $\mathbb{A}$
$\mathcal{C}_I$	Searchable ciphertext
TK	Transformation key of ABE scheme
SK	Secret key of ABE scheme
dk	Decryption key associated with a user, consisting of SK and TK
sk	Private key of signature scheme
K_{sse}	SSE Private Key
T_w	Trapdoor associated with w
id_f	Id of the file f