박사 학위논문

Ph. D. Dissertation

클라우드 스토리지 시스템에서 안전하고 효율적인 데이터 중복 제거 기법에 관한 연구

Secure and Efficient Data Deduplication Techniques for Cloud Storage Systems

신 영 주 (申 榮 柱 Shin, Youngjoo)

전산학과

Department of Computer Science

KAIST

2014

클라우드 스토리지 시스템에서 안전하고 효율적인 데이터 중복 제거 기법에 관한 연구

Secure and Efficient Data Deduplication Techniques
for Cloud Storage Systems

Secure and Efficient Data Deduplication Techniques for Cloud Storage Systems

Advisor : Professor Kim, Kwangjo

by

Shin, Youngjoo
Department of Computer Science
KAIST

A thesis submitted to the faculty of KAIST in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science . The study was conducted in accordance with Code of Research Ethics[1].

2014. 5. 15.

Approved by

Professor Kim, Kwangjo

[Advisor]

[1]Declaration of Ethical Conduct in Research: I, as a graduate student of KAIST, hereby declare that I have not committed any acts that may damage the credibility of my research. These include, but are not limited to: falsification, thesis written by someone else, distortion of research findings or plagiarism. I affirm that my thesis contains honest conclusions based on my own careful research under the guidance of my thesis advisor.

클라우드 스토리지 시스템에서 안전하고 효율적인 데이터 중복 제거 기법에 관한 연구

신 영 주

DCS 20115153

신 영 주. Shin, Youngjoo. Secure and Efficient Data Deduplication Techniques for Cloud Storage Systems. 클라우드 스토리지 시스템에서 안전하고 효율적인 데이터 중복 제거 기법에 관한 연구. Department of Computer Science . 2014. 80p. Advisor Prof. Kim, Kwangjo. Text in English.

ABSTRACT

Data outsourcing to a cloud storage brings forth new challenges for efficient utilization of computing resources such as storage space and network bandwidth in the cloud computing infrastructure. Data deduplication refers to a technique that eliminates the redundant data on the storage and transmitting on the network, and is considered to be one of the the most-enabling storage technologies that offers efficient resource utilization in the cloud computing. However, applying data deduplication incurs security vulnerabilities in the cloud storage system so that untrusted entities including a cloud server or unauthorized users may break data confidentiality, privacy and integrity on the outsourced data. It is challenging to solve the problems of data security and privacy with respect to data deduplication, but certainly necessary for offering a mature and stable cloud storage service.

In the dissertation, we study the security implications of data deduplication in the cloud computing environment. We analyze the issues of security and efficiency in various aspects, and propose novel solutions for secure and efficient data deduplication in the cloud storage system.

First, we propose a secure and efficient file deduplication scheme that keeps data confidentiality from a cloud server and unauthorized users. For implementing the deduplication scheme, we construct two symmetric-key equality predicate encryption algorithms, which are cryptographic primitives in the symmetric-key setting that allow to know only equivalence relations among ciphertexts without leaking any other information about that plaintexts. By applying the constructions, the cloud server is able to perform deduplication over encrypted files without any knowledge of their content. This offers data confidentiality against the cloud server while still preserving the desired storage efficiency. In addition, the proposed deduplication scheme adopts randomized approach in hybrid manner. That is, deduplication will occur either of at server side or at client side with some probability, which is dependent on a security parameter. This randomized strategy greatly increases attack overhead of online-guessing adversaries, hence reduces the risk of information leakage on the stored data. The constructed equality predicate encryptions offer provable security, and the required data security of the proposed deduplication scheme is also strongly enforced.

Second, we address a problem of information leakage through the side channel in client-side data deduplication, and discuss inefficiency and security weakness of previously proposed solutions. For satisfying the desired privacy requirements, we propose a differentially private client-side data deduplication protocol. Differential privacy is a security notion that guarantees enhanced privacy against an adversary who is capable of performing statistical analysis using sufficient computing resources. By exploiting a differential private approach, the proposed protocol strongly guarantees that it is hard for the side channel adversaries to infer the existence of individual data in the cloud storage. To implement the proposed scheme, we utilize a storage gateway, which is a network appliance server that provides access to the remote cloud storage over the Internet. In the proposed scheme, a storage gateway mainly handles user requests for data uploads and performs data deduplication on them on behalf of users. This storage gateway based approach prevents generating unnecessary network traffic, thus elevates the efficiency in terms of network bandwidth utilization, while eliminating the side channel by weakening the link between the deduplication event and the amount of actually transferred. In order to validate the effectiveness of the proposed solution, we make the analysis of security and performance, as well as some experiments.

Third, we address a problem of security in a proof of storage with deduplication, which is an approach that combines proof of ownership protocol with reliable data auditing schemes in a cloud storage system. We discuss a security weakness of a previous scheme under a newly proposed attack model, in which users are allowed to manipulate their own keys. More concretely, we show that the previous scheme fails to satisfy the desired security requirements if malicious users initiate the protocol with dishonestly manipulated keys. We present a solution that mitigates such an attack and improves security by modifying the original scheme such that the user keys are blended with the random values contributed by the cloud storage server. This approach weakens the adversary's capability to manipulate their keys. We minimize the modification, hence the proposed scheme preserves the efficiency while providing more robust security.

All of the schemes that are proposed in the dissertation achieve high level of efficiency in terms of utilizing the resources of storage space and network bandwidth. These schemes have the advantage with regard to strong data security and privacy against an untrusted cloud server and unauthorized users. The proposed schemes are expected to contribute to the advance of secure deduplication techniques for cloud storage services.

Contents

List of Tables

List of Figures

Chapter 1. Introduction

Cloud computing is a promising technology that economically enables data outsourcing as a service using Internet technologies with elastic provisioning and usage-based pricing [1]. Many cloud service vendors provide remote data outsourcing and backup services by utilizing storage and network resources on cloud storage infrastructures. Since they promise to offer users a convenient and efficient storage service that is available anywhere and anytime over the Internet, cloud storage services such as DropBox, Mozy and Memopal are gaining popularity.

As the fast growth of data volumes increases demand for data outsourcing on cloud storage services, pay-as-you-use cloud paradigm drives the need for cost-efficient storage, specifically for reducing storage space and network bandwidth overhead, which is directly related to the financial cost savings. In order to reduce the overheads on storage and network, commercial cloud storage service providers utilize their resources efficiently through data deduplication, which refers to a technique that finds redundant data units across users, eliminates duplicate copies of them, and provides links to the remaining data instead of storing the copies. By storing and transferring only a single copy of redundant data, deduplication provides savings of both storage space and network bandwidth. Data deduplication technique is considered to be one of the most-impactful storage technology, and it is estimated that the ratio of applying deduplication will increase steadily among the storage service providers [2].

Data deduplication techniques can be classified according to some criteria. There are two strategies of data deduplication based on the granularity of data units that they handle. In file-level deduplication, a file is the basic data unit, and only a single copy of each file is stored instead of storing duplicate copies of the file. In block-level deduplication, which segments a file into multiple fixed-sized blocks (or variable-sized chunks), each block is compared to other blocks to determine whether the both blocks are identical. Both strategies share the same redundancy elimination method that calculates the hash value of the data unit, and find identical data by comparing its hash value.

In terms of the deduplication architecture, there are two basic approaches. In the server-side (*i.e.,* target-based) deduplication, the cloud storage server mainly handles deduplication. The process of removing duplicates of data is occurred on the server, hence data deduplication is transparent to the client (the user). On the other hand, client-side (*i.e.,* source-based) deduplication occurs at the client before a file is transferred to the server. That is, in client-side duplication, the client checks with the cloud server, usually by calculating the hash value of the file and sending it to the server, whether a file has already been stored before uploading. Because the client does not need to transfer the whole file over the network if the file already exists on the storage, client-side deduplication improves not only storage space but also network bandwidth utilization. As noted in [3], this data deduplication technique achieves disk and bandwidth savings of more than 90%, and this brings huge financial cost savings to the cloud storage service providers as well as their customers.

1.1 Security Issues on Cloud Storage with Data Deduplication

Although the data deduplication technique is considered to be effective and useful in storage systems, there are several challenging issues of data security and privacy in the cloud storage services where the data deduplication technique is applied. These issues of security and privacy originate from the following facts:

- In the cloud computing environments, cloud servers are usually outside of the trust domain of the data owners (*i.e.,* users). In fact, a wide range of the users are more than willing to put their data outsourcing task to a cloud storage provider.
- Cloud storage services are typically based on multi-tenant architecture, where there is no trust relationship among users. Chasing efficiency in terms of utilizing resources such as the storage space and the network bandwidth leads to applying client-side data deduplication across multiple (untrusted) users.

In the cloud storage system with data deduplication, untrusted entities including a cloud server and users may cause security threats to the storage system. By exploiting some vulnerabilities in data deduplication, both an inside adversary, which act as a cloud server, and an outside adversary, which act as an user, will attempt to break data confidentiality, privacy and integrity on the outsourced data. More concretely, for cloud storage system with deduplication, we are concerned with several security issues that are raised by the adversaries: 1) sacrificing data security for deduplication, 2) information leakage through side channel, and 3) unauthorized arbitrary data access.

1) Sacrificing Data Confidentiality for Deduplication

Many security solutions for data outsourcing on cloud storage were proposed to keep data confidentiality and privacy from an untrusted cloud server, mainly focusing on access control [4]-[8] and searchability over the encrypted data [9]-[13]. These proposed solutions are usually attained through a sort of encryption techniques. Cloud storage service providers that are using deduplication, however, are typically reluctant to apply encryption on the stored data, because encrypting on data impedes executing data deduplication [3][14]. They may be unlikely to stop using deduplication due to the high cost savings offered by the technique. This eventually incurs the loss of confidentiality for the data stored on the cloud storage.

Several approaches [15]-[18] proposed some solutions to enable data deduplication over the encrypted data. Their solutions commonly exploit a deterministic encryption primitive, so called convergent encryption. This encryption algorithm takes a hashed value of a data as an encryption key, hence will always output the identical ciphertexts on the same plaintexts. Using this technique, a cloud server can perform deduplication over encrypted data. Convergent encryption, however, inherits deterministic property that is not as secure as the semantically secure and randomized encryption algorithms [19].

2) Information Leakage through Side Channel

In a storage system using client-side data deduplication, losing confidentiality of the outsourced data is not the only security problem. As shown in [3], client-side deduplication commonly incurs a side channel through which a malicious user (*i.e.,* an adversary) may get sensitive information about the other user's data. The side channel is caused by two inherent properties; 1) data transmission over the network is visible to an adversary, and 2) a small-sized hashed value of a file is used to determine the existence of the same file on the cloud server.

Using the side channel, an adversary can easily identify the existence of a file by uploading the file and monitoring its network activity. If the whole file is not transmitted over the network, the adversary learns that the file already exists on the server. Furthermore, the adversary is even able to learn the content of the file by mounting an online-guessing attack. That is, an adversary trying to figure out the content of the file will build a dictionary that consists of guessed versions of that file and repeat uploading each guess to the server until finding out that a deduplication event occurs.

3) Unauthorized Arbitrary Data Access

Besides an information leakage through the side channel, a cloud storage system that applies the client-side deduplication technique is also vulnerable to another type of attack [20]. This new security threat originates from the fact that client-side deduplication systems typically adopt a hashing based strategy, in which a small-sized hash value is calculated for each file (or block) and is used to find duplicate copies of the same file. In such a storage system, by accepting the hash value for the file, the cloud server allows anyone who possesses the hash value to download the entire file.

An adversary, who does not have a whole file except its hash value, will exploit this vulnerability to get the ownership of that file. The adversary can convince the cloud storage server that it owns that file by presenting just the hash value, hence can download the entire file from the server.

1.2 Contribution

It is challenging to solve the problems of data security and privacy with respect to data deduplication, but certainly necessary for enhancing security in the cloud storage services. Unfortunately, the issues of efficient resource utilization using data deduplication while preserving security in the cloud computing have not been considered together and well addressed yet in either of academia or industries. It actually still remains open to achieve security and privacy against an untrusted server as well as unauthorized and malicious users, while enabling data deduplication. For offering a mature and stable cloud storage service, both problems of cost efficiency and data security should be addressed well and resolved simultaneously.

In the dissertation, we study the security implications of data deduplication in the cloud computing environment. We analyze the issues of security and efficiency in various aspects, and propose novel solutions for secure and efficient data deduplication in the cloud storage system.

First, we propose a secure and efficient file deduplication scheme that preserves data security against an untrusted cloud server and unauthorized users. Concretely, we construct equality predicate encryption schemes in the symmetric-key setting that are suitable for our applications. Equality predicate encryption is a cryptographic primitive that allows to know only equivalence relations among ciphertexts without leaking any other information about that plaintexts. Utilizing our constructions, the cloud server is able to perform deduplication over encrypted files without learning their content. This offers data confidentiality against the cloud server while still preserving the storage efficiency. In addition, the proposed scheme allows data deduplication in a hybrid manner. That is, deduplication will occur randomly either of at server side or at client side. This randomized strategy greatly increases attack overhead of online-guessing adversaries, hence reduces the risk of information leakage on the stored data. The constructed equality predicate encryptions provide provable security, hence the required data security of the proposed deduplication protocol, which is built upon these constructions, is strongly enforced. The proposed solution achieves high level of efficiency in terms of storage space and network bandwidth utilization, while also ensuring strong data security.

Second, we address an information leakage problem caused by the side channel in client-side deduplication, and discuss inefficiency and security weakness of previously proposed solutions. In order to achieve desired privacy requirements, we propose a differentially private client-side data deduplication protocol. Differential privacy is a security notion that guarantees enhanced privacy against an adversary who is capable of doing statistical analysis using sufficient computing resources. By applying a differential private mechanism, the proposed protocol strongly guarantees that the presence or absence of individual data in the cloud storage is hard to infer from any side channel attacks. For the implementation, our proposed solution utilizes a storage gateway, which is a network appliance server that provides access to the remote cloud storage over the Internet. In the proposed solution, a storage gateway mainly handles user requests for data uploads and performs data deduplication on them on behalf of users. The approach using a storage gateway avoids generating unnecessary network traffic, thus improves the efficiency in terms of network bandwidth utilization, while preventing the side channel by weakening the link between the deduplication event and the amount of actually transferred. For the validation of the effectiveness of the proposed solution, we make the analysis of security and performance, as well as some experiments.

Third, we address the security problem of a proof of storage with deduplication, which is an approach that combines proof of ownership protocol with reliable data auditing schemes in a cloud storage system. We propose a new attack model that users are allowed to manipulate their own keys, and discuss security weakness of a previous scheme under the attack model. More concretely, it is shown that the previous scheme fails to satisfy desired security requirements if malicious users run the protocol with dishonestly manipulated keys. We present a solution that mitigates this attack and enhances the security by modifying the original scheme such that the clients-created keys are blended with the random values contributed by the storage server. This approach weakens the adversary's capability to manipulate their keys. We minimize the modification so that the proposed scheme preserves the efficiency while providing more robust security.

1.3 Organization

This paper is organized as follows. In Chapter 2, we describe several related works for secure data deduplication in the cloud computing environment. In Chapter 3, we propose a secure data deduplication protocol using equality predicate encryption. We present two constructions of the equality predicate encryption scheme, and describe the proposed protocol. The security analysis of the protocol and the encryption algorithms as well as the performance analysis are also given. In Chapter 4, we address the issue of information leakage through the side channel in client-side deduplication, and discuss the inefficiency and security weakness of previous solutions. We also present the proposed differentially private client-side deduplication protocol that is resistant against such an attack, as well as its security and performance analysis. In Chapter 5, we present a new attack model in proof of storage with deduplication, and discuss the security weakness of a previous scheme under our attack model. We propose a scheme that mitigates the attack while keeping all the desired properties. We also present the security analysis of the proposed scheme. In Chapter 6, we remark the conclusion and future work of the dissertation.

Chapter 2. Related Work

2.1 Secure Data Outsourcing Techniques

2.1.1 Access Control over the Outsourced Data

Most previous works addressed the issue of secure data outsourcing in terms of cryptographic access control. In order to achieve a goal of fine-grained access control and efficient revocation over the outsourced data, various techniques are proposed built on several cryptographic primitives that ensure access control over encrypted data. Attribute-based encryption (ABE) [21][22] is one of their building tools. ABE is a type of public key encryption primitive in which the private key given to a user and the corresponding ciphertext are dependent upon attributes. In ABE, a user can decrypt the ciphertext if and only if the set of attributes given to the user matches the attributes of the ciphertext (*i.e.,* the access policy of the ciphertext).

Yu *et al.* addressed an issue of secure outsourcing to cloud storage servers with attribute revocation, and proposed two solutions [4] and [5] in KP (Key Policy)-ABE and CP (Ciphertext Policy)-ABE, respectively. In KP-ABE based scheme [4], outsourced data are associated with a set of attributes, and a data owner associates the set of attributes to the data by encrypting it with the corresponding public key. Each client who is assigned to attributes in the access tree over the outsourced data can decrypt it according to the given attributes. In CP-ABE based scheme [5], each user is associated with attributes and data is encrypted with access structures according to attributes. Any user whose attributes satisfy the access policy of the encrypted data can decrypt it.

Hur *et al.* [6] addressed a problem of key management in ABE based access control system, specifically when a user is revoked from a group of users with same attributes (*i.e.,* an attribute group). Their main approach is to combine both CP-ABE and broadcast encryption technique [23] together for enabling more fine-grained access control with efficient and scalable revocation capability. Utilizing selective group key distribution mechanism, their solution provides backward/forward secrecy of outsourced data on any membership changes in the attribute group, hence achieves high degree of security compared to previous attribute revocation schemes.

Zhou *et al.* [7] and Green *et al.* [8] considered ABE encryption and decryption overhead at client side, which grows with the complexity of the access formula. They proposed solutions that outsource the computation burden to the cloud server while preserving privacy of outsourced data. Zhou *et al.*'s work was focused on constructing a CP-ABE scheme which has a privacy preserving property, namely PP-CP-ABE scheme. The main idea of Green *et al.*'s solution is to generate a single transformation key that allows a cloud server to translate any ABE ciphertext that satisfies the access policy into a ciphertext that can be decrypted by the user.

2.1.2 Proof of Retrievability/Provable Data Possession

Outsourcing data to untrusted remote servers in the cloud storage service brings a new challenge: ensuring the authenticity of outsourced data on storage. A straightforward solution for this problem is that a client downloads the whole of files from storage and tests the integrity of downloaded files itself. This approach, however, incurs a large number of disk I/Os and wasteful network bandwidth consumption. Juels *et al.* [24] and Ateniese *et al.* [25] considered this problem and proposed their solutions that enable verifying the integrity of outsourced data in efficient way. Both works were presented almost at the same time and were followed by several works extending their techniques in various ways.

Juels *et al.* [24] presented a notion of proof of retrievability (POR), which is similar to a kind of cryptographic proof of knowledge. Unlike a proof of knowledge, however, in a POR neither a prover (untrusted storage server) nor a verifier (a client) need actually to have knowledge of a file. Hence, the POR scheme is applicable in some environments where the outsourced data should be kept private from the remote storage server. Briefly, the POR scheme encrypts a file F and randomly embeds a set of sentinels, which consists of small blocks that are filled with random value, to the file. The sentinel value is indistinguishable from other data of file blocks. During the protocol, the verifier sends challenges that specify the positions of some sentinels to the prover, and asks for returning the associated sentinel values back This protocol will detect the case that the prover has removed or modified a substantial portion of F with high probability. The POR scheme has a preprocessing phase in which a file F is encoded to F' via error correcting codes. Hence, this scheme also provides a feature that corrupted portions of F can be recovered through the protocol.

Shacham *et al.* [26] addressed on improving previous schemes in terms of executing in more efficient way. The basic idea is to store homomorphic block integrity values so that in Shacham *et al.*'s protocol, instead of sending out $O(l)$ messages (l is a security parameter), these message blocks are aggregated into just one message. This approach gives an advantage that clients (*i.e.,* verifiers) can generate and verify an unlimited number of challenges. Dodis *et al.* [27] identified several variants of the problem in POR model, and presented some improvements of the previous scheme as well. They gave an optimal version of Jeuls *et al.*'s scheme, where the communication cost is linear, and proved its security without any assumption of adversary's behaviors as well as not relying on the random oracle model. Bowers *et al.* [28] addressed the limitations of previous theoretical models, and proposed a new theoretical framework for designing POR scheme which supports various adversarial models. The aforementioned POR schemes do not support dynamic storages in which frequent file update operations are permitted, hence are very inefficient in practical applications. Shi *et al.* [29] proposed dynamic POR scheme that achieves efficiency in dynamic storage environments in terms of communication bandwidth overhead and computation cost at client side.

Ateniese *et al.* [25] focused on drawbacks of Juels *et al.*'s POR scheme [24] that it is applicable to only encrypted data storage and the number of available challenge queries is quite limited in that model. They introduced another model of data verifiability, provable data possession (PDP), which can be applied to large and public databases with no restriction on the number of possible challenges. PDP provides probabilistic proof that a client who has outsourced data on an untrusted storage server can verify that the data is not removed or modified without retrieving it. In this model, a client just needs holding a small sized data that is generated from a file instead of downloading the whole of file data, and the storage serer is allowed to access small portions of the file for generating the proof. In Ateniese *et al.*'s PDP constructions, homomorphic verifiable tags, which have a homomorphic property that tags computed from multiple data blocks can be aggregated into a single value, are used. Before uploading a file, a client computes tags for each block of the file and stores them with the file in the storage server. At later, the client can initiate the verification protocol with the storage server by generating a challenge chosen at random from a set of file blocks. Upon receipt of the challenge,the server responds a proof of possession that is computed by the queried blocks and their corresponding tags. This scheme helps the client detect the case that the file has been modified or deleted with high probability.

Following the previous work [25], Ateniese *et al.* [30] presented another version of PDP scheme in the symmetric-key setting, which is also an improvement of both the efficiency and the security of the original PDP. Furthermore, their improved version supports dynamic storage environments, where dynamic operations such as modification, deletion, and append on the outsourced data are permitted. Erway *et al.* [31] also addressed the problem of provable data possession in dynamic storage environments. They introduced a framework for dynamic provable data possession that extends the PDP model to support updates on the outsourced data. Their solution is based on a variant of authenticated dictionaries, where rank information is used for the dictionary organization. Zhang *et al.* [32] focused on lowering the overhead during update operations and proposed a solution that utilized a data structure termed a balanced update tree.

2.2 Techniques for Search over Encrypted Data

2.2.1 Predicate Encryption

Besides an issue of secure data outsourcing, it is also important to address the problem of searching over encrypted storage in the cloud computing environments. Predicate encryption techniques [9][33][34] are among the solutions for searching over encrypted data. A predicate encryption is a kind of functional encryption [35], which has a feature that a master secret key allows a user to evaluate a function (*i.e.,* a predicate) of encrypted data without knowing the plaintext. In predicate encryption scheme, an owner of a master secret key generates a token, which is associated with predicates, and then other users can evaluate predicates over the owner's encrypted data using the token without knowing any information of the plaintext.

Specifically, users rather than the owner can learn whether C, an encryption of data M, satisfies a predicate P by evaluating a token TK_P associated with P over C.

Boneh *et al.* [9] focused on the problem of keyword search over encrypted data in the public-key setting, and proposed constructions of PEKS (Public-key Encryption of Keyword Search). PEKS is a public-key equality predicate encryption, which is a predicate encryption scheme that allows to evaluate an equality predicate over a ciphertext of M with a token of M' to learn whether $M = M'$ in the public-key environment. PEKS consists of four probabilistic polynomial time (PPT) algorithms. The one of constructions of PEKS is built on bilinear maps. Let a bilinear map $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$, where the order of $\mathbb{G}$ is p, and let two hash functions $H_1 : \{0,1\}^* \rightarrow \mathbb{G}$ and $H_2 : \mathbb{G}_T \rightarrow \{0,1\}^{\log p}$. The four PPT algorithms of PEKS work as follows:

- KeyGen(1^λ): Given a security parameter 1^λ, this algorithm picks a random value $\alpha \in \mathbb{Z}_p^*$, where the length of p depends on λ, and a generator $g \in \mathbb{G}$, then outputs a public key $A_{pub} =< g, h = g^\alpha >$ and a private key $A_{priv} = \alpha$.
- PEKS(A_{pub}, M): This algorithm picks a random $r \in \mathbb{Z}_p^*$, and computes $t = e(H_1(M), h^r) \in \mathbb{G}_T$. Then, it outputs $S =< g^r, H_2(t) >$.
- Trapdoor(A_{priv}, M): This algorithm outputs $T_M = H_1(M)^\alpha \in \mathbb{G}$.
- Test(A_{pub}, S, T_M): S is a tuple $< A, B >$, where $A = g^r$ and $B = H_2(t)$. This algorithm tests whether if $H_2(e(T_M, A)) = B$. If the equation holds, then it outputs 'yes'; otherwise, outputs 'no'.

PEKS is suitable for several applications including a secure e-mail server which is possible for the untrusted server to retrieve encrypted mails. The security of PEKS was proved such that it is semantically secure against a chosen keyword (*i.e.,* a ciphertext) attack in the random oracle model assuming the Bilinear Diffie-Hellman (BDH) problem [36][37] is hard.

Shen *et al.* [33] considers a new notion called predicate privacy, which is a property that tokens reveal no information about the predicate. Since this property is inherently unnecessary in the public-key setting, they focused on predicate encryption in the symmetric-key setting and presented a symmetric-key predicate encryption system. Shen *et al.*'s scheme also supports inner product queries, hence multiple types of predicates possible in the scheme. Their construction is based on bilinear groups of composite order $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$, where $\mathbb{G}$ and $\mathbb{G}_T$ are two cyclic groups of order $N = pqrs$ (p,q,r,s are distinct primes). Let the class of plaintexts be $\Sigma = \mathbb{Z}_N^n$ and the class of predicates be $\mathcal{F} = \{f_{\vec{v}} | \vec{v} \in \mathbb{Z}_N^n\}$ with $f_{\vec{x}}(\vec{v}) = 1$ if and only if $< \vec{x}, \vec{y} >= 0$. The construction is described as follows:

- Setup(1^λ): This algorithm picks four generators g_p, g_q, g_r, g_s of $\mathbb{G}_p, \mathbb{G}_q, \mathbb{G}_r, \mathbb{G}_s$, respectively, where the size of each group depends on the security parameter λ. It also picks $h_{1,i}, h_{2,i}, u_{1,i}, u_{2,i} \in \mathbb{G}_p$ at random for $1 \leq i \leq n$, and outputs a secret key $SK = \left(g_p, g_q, g_r, g_s, \{h_{1,i}, h_{2,i}, u_{1,i}, u_{2,i}\}_{i=1}^n\right)$.

- Encrypt(SK, $\vec{x}$): Let $\vec{x} = (x_1, \dots, x_n) \in \mathbb{Z}_N^n$. This algorithm chooses at random $y, z, \alpha, \beta \in \mathbb{Z}_N$, $S, S_0 \in \mathbb{G}_s$, and $R_{1,i}, R_{2,i} \in \mathbb{G}_r$ for $1 \leq i \leq n$. Then, it outputs the ciphertext $CT = \big(C, C_0, \{C_{1,i}, C_{2,i}\}_{i=1}^n\big)$, where $C = S \cdot g_p^y$, $C_0 = S_0 \cdot g_p^z$, $C_{1,i} = h_{1,i}^y \cdot u_{1,i}^z \cdot g_q^{\alpha x_i} \cdot R_{1,i}$ and $C_{2,i} = h_{2,i}^y \cdot u_{2,i}^z \cdot g_q^{\beta x_i} \cdot R_{2,i}$.

- GenToken(SK, $\vec{y}$): Let $\vec{v} = (v_1, \dots, v_n) \in \mathbb{Z}_N^n$. This algorithm chooses at random $f_1, f_2 \in \mathbb{Z}_N$, $r_{1,i}, r_{2,i} \in \mathbb{Z}_N$ for $1 \leq i \leq n$, $R, R_0 \in \mathbb{G}_r$, and $S_{1,i}, S_{2,i} \in \mathbb{G}_s$ for $1 \leq i \leq n$. Then, it outputs the token $TK_{\vec{v}} = \big(K, K_0, \{K_{1,i}, K_{2,i}\}_{i=1}^n\big)$, where $K = R \cdot \prod_{i=1}^n h_{1,i}^{-r_{1,i}} \cdot h_{2,i}^{-r_{2,i}}$, $K_0 = R_0 \cdot \prod_{i=1}^n u_{1,i}^{-r_{1,i}} \cdot u_{2,i}^{-r_{2,i}}$, $K_{1,i} = g_p^{r_{1,i}} \cdot g_q^{f_1 v_i} \cdot S_{1,i}$, and $K_{2,i} = g_p^{r_{2,i}} \cdot g_q^{f_2 v_i} \cdot S_{2,i}$.

- Query($TK_{\vec{v}}$, CT): Given $CT = \big(C, C_0, \{C_{1,i}, C_{2,i}\}_{i=1}^n\big)$ and $TK_{\vec{v}} = \big(K, K_0, \{K_{1,i}, K_{2,i}\}_{i=1}^n\big)$, this algorithm checks the following equation

$$e(C, K) \cdot e(C_0, K_0) \cdot \prod_{i=1}^{n} e(C_{1,i}, K_{1,i}) \cdot e(C_{2,i}, K_{2,i}) = 1.$$

  If the above equation holds, then it outputs 'yes'; otherwise, it outputs 'no'.

The security of Shen *et al.*'s scheme was proved under Generalized 3-Party Diffie-Hellman Assumption (C3DH) [38] in the context of groups whose order is the product of four distinct primes.

Blundo *et al.* [34] improved Shen *et al.*'s scheme in terms of efficiency with respect to the computation and the ciphertext size. Their construction is based on asymmetric bilinear groups of prime order $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, where $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ are cyclic multiplicative groups of order p. Let the plaintext vector be $\vec{x}$ and the predicate vector be $\vec{k}$. The algorithms are described as follows:

- MasterKeyGen (1^λ,l): Let l be the number of attributes, which is dependent of the security parameter 1^λ, and let g_1,g_2 be generators of $\mathbb{G}_1$, $\mathbb{G}_2$, respectively. This algorithm randomly picks $y \in \mathbb{Z}_p$ and sets $Y = e(g_1, g_2)^y$. Then, for $1 \leq i \leq l$, it chooses at random $t_{i,0}, t_{i,1}, v_{i,0}, v_{i,1} \in \mathbb{Z}_p$, and set $SK_i = (T_{i,0} = g_1^{t_{i,0}}, T_{i,1} = g_1^{t_{i,1}}, V_{i,0} = g_1^{v_{i,0}}, V_{i,1} = g_1^{v_{i,1}}, \bar{T}_{i,0} = g_2^{1/t_{i,0}}, \bar{T}_{i,1} = g_2^{1/t_{i,1}}, \bar{V}_{i,0} = g_2^{1/v_{i,0}}, \bar{V}_{i,1} = g_2^{1/v_{i,1}})$. The output is $SK = (Y, y, SK_1, \dots, SK_l)$.

- Encrypt (SK, $\vec{x}$): This algorithm picks at random $s \in \mathbb{Z}_p$ and set $\Omega = Y^{-s}$. Then, for $1 \leq i \leq l$, it computes $X_i = T_{i,x_i}^{s-s_i}$ and $Z_i = V_{i,x_i}^{s_i}$, where $s_i \in \mathbb{Z}_p$ is chosen at random. The output is $\tilde{X} = \Big(\Omega, (X_i, Z_i)_{i=1}^l\Big)$.

- KeyGen (SK, $\vec{k}$): Let $S_{\vec{k}}$ be the set of positions in which k_i is not empty. This algorithm chooses randomly $(a_i)_{i \in S_{\vec{k}}}$ in $\mathbb{Z}_p$ under the constraint that their sum is y. Then, for $i \in S_{\vec{k}}$, it computes $R_i = \bar{T}_{i,k_i}^{a_i}$ and $W_i = \bar{V}_{i,k_i}^{a_i}$. The output is $\tilde{K} = (i, R_i, W_i)_{i \in S_{\vec{k}}}$.

- Test ($\tilde{X}$,$\tilde{K}$): On input a ciphertext vector $\tilde{X} = \big(\Omega, (X_i, Z_i)_{i=1}^{l}\big)$ and a predicate key vector $\tilde{K} = \big(i_j, R_{i_j}, W_{i_j}\big)_{1 \leq j \leq m}$, this algorithm checks the equation $\Omega \cdot \prod_{j=1}^{m} e(X_{i_j}, R_{i_j}) e(Z_{i_j}, W_{i_j}) = 1$. If this equation holds, then it outputs 'yes'; otherwise, it outputs 'no'.

Another version for larger alphabets also can be easily extended from the above construction. The semantic security of this scheme was proved under the Bilinear Decisional Diffie-Hellman (BDDH) assumption.

2.2.2 Deterministic Searchable Encryption

Predicate encryption-based solutions provide strong privacy guarantee such as semantic security against untrusted cloud storage server, but their search operation takes linear time with respect to the size of storage. For some applications that require low searching complexity (*i.e.,* sub-linear searching time) while tolerating the loss of some privacy guarantee, deterministic searchable encryption techniques [10][11][12][13] can be another option for the solution.

The basic idea of deterministic searchable encryptions is to eliminate any randomness in running algorithms and to generate the ciphertexts in deterministic manner. This approach enables tree-based indexing over encrypted data, hence it can achieve high degree of efficiency. Due to the deterministic property of ciphertexts, however, there are several inherent limitations of deterministic encryption as follows.

- No privacy can be guaranteed if the space of plaintext is too small, or an adversary has some knowledge over it so that the adversary can mount guessing attack with a given ciphertext to find out the information of the corresponding plaintext.
- Since a ciphertext is deterministically computed from the corresponding plaintext, the ciphertext itself can be partial information about the plaintext.

In order to achieve desired level of privacy in constructing deterministic encryptions, the above limitations lead a requirement that the plaintext space should have a large min-entropy. Considering this limitations and requirement, Bellare *et al.* [11] presented PRIV, a new notion of privacy and security that is suitable for deterministic encryption, and gave some constructions in the public-key setting that preserve this privacy notion. Their constructions can be used as a tool for building various deterministic searchable encryption schemes utilizing existing encryption primitives and cryptographic hash functions. Searchable encryption schemes that satisfy the definition of PRIV can provide strong privacy in the random oracle model assuming that the underlying encryption primitive is semantically secure and the plaintext space has high min-entropy.

Besides the public-key based constructions of searchable encryption, Sedghi *et al.* [12] and Curtmola *et al.* [10] proposed deterministic searchable encryption schemes in the symmetric-key setting. Sedghi *et al.* [12]'s scheme supports efficient searching over encrypted data in logarithmic time, as well as a new feature that the user can update the content of outsourced data without giving any information about the modification to an adversary. Security of this scheme was proved under adaptive attack model. Curtmola *et al.* [10] extended previous searchable encryptions to the multi user setting, where users can query for search over other user's encrypted data. They formally defined symmetric searchable encryption in the multi user setting, and presented a construction. C. Dong *et al.* [13] also proposed another searchable encryption scheme and focused on enhancing security notions of previously proposed solutions.

2.3 Secure Data Deduplication Techniques

2.3.1 Convergent Encryption based Schemes

Implementing cross user data deduplication over outsourced data which is encrypted with conventional cryptosystems, where the encryption keys are supplied by each user, is not straightforward, since a cryptosystem prevents the storage server from knowing the equivalence relations among encrypted data. To overcome this, many solutions have been proposed using so called convergent encryption.

The notion of convergent encryption was first presented in [16] as a rough idea. Convergent encryption is a deterministic cryptosystem that generates a ciphertext using a hashed value computed from a plaintext (*i.e.,* a file) as an encryption key of the plaintext itself. Convergent encryption can be constructed from any symmetric encryption cryptosystems (*e.g.,* AES, DES or Blowfish, etc) and cryptographic hash algorithms (*e.g.,* MD5, SHA1, etc). Let us denote a cryptographic hash algorithm to H, and a symmetric encryption cryptosystem to a tuple (E, D), where E is an encryption algorithm and D is a decryption algorithm. Convergent encryption is defined formally as a tuple of the following three algorithms:

- Key generation: it takes a plaintext $M \in \{0,1\}^*$ as an input and computes $H(M)$, which is a hashed value of M. Then, it outputs an encryption key $k = H(M)$.
- Encryption: it takes an encryption key k and a plaintext M as inputs, and computes a ciphertext $C = E(k, M)$.
- Decryption: it takes an encryption key k and a ciphertext C as inputs, and computes a plaintext $M = D(k, C)$.

Convergent encryption has a property that any client encrypting a given data chunk (or a file) will use the same encryption key to do so, thus identical plaintexts will be encrypted to identical ciphertexts, regardless of who encrypts them. This property helps the storage server perform data deduplication on encrypted outsourced data. Several solutions after [16] fundamentally follow this strategy.

M. Storer *et al.* [15] further extended [16] and implemented data deduplication framework on the encrypted data storage. Their approach, which aims at achieving both storage efficiency and data privacy, provides a solution for secure data deduplication in authenticated and anonymous way. The solution can be applied to single server storage environments, where both data and metadata reside together, as well as to distributed server storage environments, where metadata is stored on an independent server.

Li *et al.* [39] observed that secure deduplication schemes based on convergent encryption suffer from managing an enormous number of convergent keys, since both encrypted data chunks and their convergent keys should be handled together. To address this key management problem, they proposed *Dekey*, which is an efficient and reliable key management scheme in secure deduplication. This construction makes cloud storage users to distribute their convergent key shares across multiple cloud storage servers instead of managing the keys on their own.

DuPLESS [40] is a real deduplication system, which is built on commercial cloud storage services, that provides security against brute-force attacks launched by malicious clients or an untrusted server. In order to achieve the desired goals while satisfying the required security, a Oblivious Pseudo Random Function (OPRF) protocol and message-locked encryption (MLE) [41] were utilized for their construction. OPRF is a randomized protocol between clients and the key server, which ensures that the key server learns nothing about the inputs and the resulting outputs, and the clients learn nothing about the key. MLE is a generalized version of convergent encryption designed to give restrictive semantic-security for unpredictable messages. Duan *et al.* [42] provided a rigorous security proof of DuPLESS in the random oracle model, and proposed a distributed protocol that eliminates the need for a key server in DuPLESS system.

Bellare *et al.* [41][43] gave security analysis on some variants of convergent encryptions in detail, and discussed a tag consistency, which is a security property that prevents malicious adversaries from faking deduplication, on them. According to [41][43], a deterministic convergent encryption is the only encryption that preserves strong tag consistency. However, data confidentiality such as semantic security cannot be strongly guaranteed due to its inherent deterministic property.

Aside from applying convergent encryption to cloud storage systems, L. Marques *et al.* [17] and P. Anderson *et al.* [18] utilized convergent encryption to address various problems in different applications such as mobile devices.

2.3.2 Side Channel Resistant Schemes

Performing data deduplication across multiple users' data at client side incurs a side channel, which can be exploited by adversaries who try to infer sensitive information of other user's data such as the existence of the data or even its content. Several works [3][44] studied various security problems caused by the side channel in client-side deduplication, and classified three possible attack scenarios on cloud storage systems as follows:

- Identifying the existence of files: Suppose that an adversary wants to know the existence of interesting file F on the storage. The adversary can easily identify the existence of F

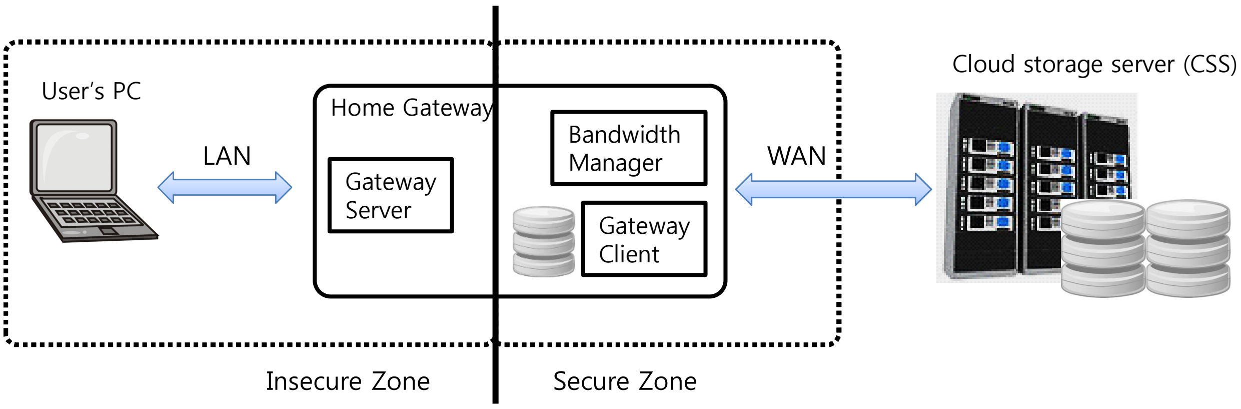


Figure 2.1: Architecture of Olivier *et al.*'s deduplication system

by uploading the file and monitoring its network activity. If actual content of F is sent over the network, then F has not previously uploaded. Otherwise, F already exists on the storage.

- Learning the content of files: An adversary who tries to learn the content of interesting file F can mount an online-guessing attack by utilizing the side channel. Initially, the adversary builds a dictionary which consists of candidates of F as many as possible. After that, the adversary repeats data deduplication protocol over each guess in the dictionary with the cloud storage server until a deduplication event occurs.

- Establishing a covert channel: Suppose that two adversaries $\mathcal{A}$ and $\mathcal{B}$ seek to establish a covert communication channel using deduplication. Two distinct random files F_0 and F_1 are used for transferring one-bit data. In order to send '1' to $\mathcal{B}$, the adversary $\mathcal{A}$ just uploads F_1 to the storage. By uploading F_1 and monitoring its deduplication event, $\mathcal{B}$ will eventually learn that '1' is sent from $\mathcal{A}$.

To prevent adversaries from utilizing side channels during deduplication, several solutions have been proposed. Harnik *et al.* [3] aimed at weakening the correlation between deduplication and the existence of files in the storage, and proposed a randomized approach that obfuscates the occurrence of deduplication. Given a security parameter d, a threshold value is randomly generated and assigned for each file in the cloud storage. When a client uploads a new copy of the file, the cloud server checks whether the accumulated number of file uploads exceeds the threshold. If the number is at least as high as the threshold, the server performs data deduplication at client side and the copy of that file is not sent over the network. Otherwise, deduplication is performed at server side, in which the actual file content is sent to the server.

The security of Harnik *et al.*'s solution is defined such that the occurrence of client-side deduplication does not reveal any information of files with some probability. However, Lee *et* *al.* [45] observed that the success probability of attack against Harnik *et al.*'s scheme is non-negligible in a security parameter, and thus user's privacy is still at risk with high probability.

Olivier *et al.* [46] also addressed the problem of the side channel in deduplication, and proposed another solution for new service deployment model, in which a network service provider offers cloud storage service through a home gateway appliance that resides at customer's network. The overall architecture of Olivier *et al.*'s deduplication system is given in Fig. 2.1. Since a home gateway resides at the border between a local home network (LAN), which is insecure zone that is easily controlled by an adversary, and a wide area network (WAN or the Internet), which is secure zone that is hardly controlled by an adversary, it plays a crucial role in this architecture. When a user sends an upload request for some file to the cloud server, the home gateway (the gateway server module) handles this request and receives a whole content of the file from the user. Upon receiving the file, the gateway (the gateway client module) performs deduplication on behalf of the client with the cloud server, and uploads the file, if necessary. Since the operation of deduplication always happens only at the WAN (*i.e.,* the secure zone) and the bandwidth manager in the gateway mixes the deduplication with other service traffic, there is no way for the adversary to get information of the interesting file from monitoring the network traffic.

2.3.3 Proof of Ownership

Aside from the side channel, Halevi *et al.* [20] observed new security problems in a cloud storage system with client-side deduplication. The security problems stems from the fact that in typical client-side deduplication, very small hash signatures of files are used for identifying the existence of same file in the storage. With knowledge of this hash value, an adversary who does not have a file can convince the cloud storage server that it owns that file by presenting just the hash value, hence can download the entire file from the server. To prevent such an attack, they introduced the notion of proof-of-ownership (POW), in which a client proves a cloud server that the client actually holds a whole of a file, rather than the file's hash value.

The security of the POW can be stated such that as long as the min-entropy in the file (from the adversary's perspective) is more than the sum of the number of bits given to the adversary and the security parameter (in bits), the adversary should not be able to convince the server that it possesses the file with non-negligible probability. In order to achieve this security, the POW adapted the notion proof of retrievability, which is based on of Merkle hash tree [47]. Once a whole content of the file is received from a client in the first time, the cloud server encodes the file using an erasure code, such that it is able to recover the entire file from any part of the encoded bits. Then, the server builds a Merkle hash tree over the encoded file and generates shorter verification information. Later, any client attempting to outsource the file runs the proof-of-ownership protocol with the server to prove that it really has the file. Interacting with the client, the server verifies materials which are sent from the client with the verification information generated from the file.

We present more concrete description of one of Halevi *et al.*'s several solutions. Let $E : \{0,1\}^M \to \{0,1\}^{M'}$ be an α-erasure code, which is able to recover upto α fraction of corrupted bits, and let H be a collision resistant cryptographic hash function. A binary Merkle tree over data buffer X with leaves of b-bits size and the hash function H is denoted to $MT_{H,b}(X)$. On an input file $F \in \{0,1\}^M$ of M bits, the verifier (*i.e.,* the cloud server) generates the encoding $X = E(F)$ as well as the Merkle tree $MT_{H,b}(X)$, and then stores the root of the tree and the leaves as verification information of F. When the proof protocol is initiated, the verifier chooses at random sufficient leafs, $l_1, l_2, \ldots, l_u$, where u depends on the security parameter, and sends the indexes of these leafs to the prover (*i.e.,* the client). Then, the prover responds the sibling-paths of all these leaves (*i.e.,* a set of the siblings of nodes along the path from leaf to root) to the verifier. After verifying the prover's response with respect to $MT_{H,b}(X)$, the verifier returns to the prover Accept if it is valid, or Fail, otherwise.

The notion of POW were extended in various ways by several works [48][49][50]. Ng *et al.* [48] proposed a private data deduplication protocol built upon Halevi *et al.*'s POW scheme. A private data deduplication protocol is a zero-knowledge proof based two-party computation protocol between a client and a cloud server. By executing the protocol, the cloud server, without knowing any information of files, can verify client's ownership of files with some materials presented by the client.

The security of their scheme was shown such that it is provably secure in terms of the simulation-based proof assuming that solving the discrete logarithm problem (DLP) is infeasible and the underlying hash algorithm is collision-resistant.

Pietro *et al.* [50] addressed the inefficiency of the original POW scheme and proposed s-POW, an efficiency-enhanced version of POW protocol in terms of the computation power and I/O capacity. The basic idea of s-POW is from the fact that the adversary's success probability to learn some portion of K bits of the file is negligible in the security parameter, hence a response to a challenge will be a K-bit string, which is constructed as the concatenation of K bits of the original file. Their constructions achieve high efficiency level that I/O and computational costs at server side do not depend on the input file size.

Xu *et al.* [49] constructed a non-linear and pairwise-independent hash function in the random oracle model, and applied it to build a POW scheme that is provably secure with respect to any distribution of input files with sufficient min-entropy. Xu *et al.*'s scheme supplements the original POW which is provably secure only in certain types of file distribution. They also addressed the problem of deduplication over encrypted data, and presented a solution that supports cross-user client-side deduplication utilizing a convergent encryption technique.

Chapter 3. Data Deduplication Protocol using Equality Predicate Encryption

3.1 Motivation and Contributions

Cloud storage service providers take advantage of efficient resource utilization by applying data deduplication techniques over data which are outsourced from customers (*i.e.,* clients). In order to keep security of outsourced data from an untrusted cloud server, several previous approaches [15]-[18] proposed some ideas to enable data deduplication over the encrypted data. Their solutions are commonly based on so-called convergent encryption, which takes a hashed value of a file as an encryption key. Using this technique, identical ciphertexts are always generated from identical plaintexts in a deterministic manner. Hence, the cloud server can perform deduplication over encrypted data, without knowing the exact content of the file. From the view of cryptography, however, it is understood that deterministic encryption, including convergent encryption naturally, is not as secure as randomized one [19].

Keeping confidentiality of the outsourced data from the untrusted cloud server is not the only security problem for the system utilizing data deduplication. It has been shown that most of storage services using client-side deduplication commonly incur a side channel through which an adversary may learn information about the contents of files of other users [3]. This is because these services share two inherent properties; 1) data transmission in a network can be visible to an adversary, and 2) deterministic and small-size data, such as a hashed value of a file, is queried to the cloud server before file uploading. A user who has not access to but curious about some file might mount an online-guessing attack exploiting the side channel. The attack is as follows: an adversary first constructs a search domain comprising possible versions of the target file, and then queries a hash of each version to the server until a deduplication event occurs. (See Fig.3.1) This attack is very relevant for a corporate environment where the file content usually has low-entropy. That is, most of files are small variations of standard templates and the number of possible versions of the target file is moderate. It is challenging but certainly necessary to prevent such an information leakage in the client-side deduplication system.

In this chapter, we address the issue of efficiency and security in cloud storage, and propose a novel solution to enable efficient resource utilization using data deduplication while preserving data security. The proposed solution raises efficiency up to a level of practicality while also achieving strong security. Concretely, the solution provides a data deduplication scheme which is secure against the untrusted server as well as unauthorized users. The proposed scheme utilizes primarily a novel cryptographic primitive, namely equality predicate encryption, which allows cloud server to know only equivalence relations among ciphertexts without leaking any

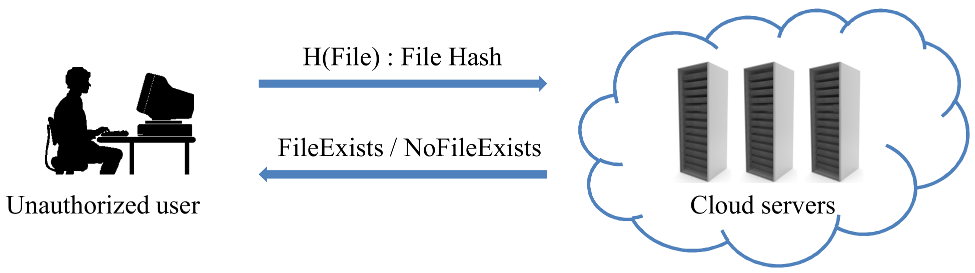


Figure 3.1: Online Guessing Attack Scenario: for each possible versions of target file unauthorized user continues querying its ashed value until receiving 'FileExists' message from the cloud server.

other information of their plaintexts. Thus data privacy can be kept from the cloud server while data deduplication is still enabled over the encrypted files. In addition, the proposed scheme allows the cloud server to perform data deduplication in a hybrid manner. That is, deduplication will occurs either of at server side or at client side with some probability. This strategy greatly reduces the risk of information leakage by increasing attack overhead of online-guessing adversaries. In order to achieve the design goals, two equality predicate encryption schemes in the symmetric-key setting are constructed to be suitable for the application. The proposed scheme is built upon these constructions, and the required data security is strongly enforced through the underlying provable security of the constructions.

Main contributions of this chapter can be summarized as follows: 1) This is the first work that simultaneously resolved two issues of the data security and the efficient resource utilization in the cloud computing; 2) The proposed scheme also gives data owners a guarantee that adding their data to the cloud storage has a very limited effect on what an adversary exploiting side channel may learn about the data; 3) The proposed scheme as well as the constructed equality predicate encryptions are proved to be secure under the standard cryptographic assumption.

The rest of this chapter is organized as follows: Section 3.2 presents the system models and an overview of the proposed solution. Section 3.3 describes the constructions of equality predicate encryption scheme. In Section 3.4, the proposed secure data deduplication scheme based on the constructions of the encryption scheme is described in detail. In Section 3.5, the security analysis and the discussion of the performance of the proposed scheme is presented. Third, this chapter is summarized in Section 3.6.

3.2 Models and Solution Overview

3.2.1 System and Attack Model

The system consists of the three entities: Data owners, users and a cloud server. A data owner wishes to outsource data (or file) in the external storage server and is in charge of encrypting the data before outsourcing it. In order to access the outsourced data, a user should be authorized by the data owner. An authorized user possess the secret key of the data file and can read the data by decrypting it. The cloud server, which stores outsourced data from data owners, has abundant but limited storage and network capacity. The cloud server is operated by the cloud service provider who is interested in cost savings by improving disk space and network utilization.

Similar to the previous approaches, the cloud server is assumed to be honest-but-curious. That is, the cloud server will honestly execute the assigned tasks in the system but may try to learn some information of stored data as much as possible. Unauthorized users would try to access stored data which is not authorized to them. To achieve this goal, they may perform an online-guessing attack using the side channel incurred by a client-side duplication as an oracle. In addition, users and other data owners may also collude to compute secret information that is necessary to access data with any information that the colluding adversaries have. Both of the cloud server and any unauthorized users mounting these attacks should be prevented from getting any information of unauthorized data on the storage.

In the attack model, malicious data owners, who mount some kind of attacks targeting a cloud server or other data owners, are also considered. In uploading a file, they would try to disturb deduplication or corrupt data owned by others by doing dishonest behaviors such as computing with a manipulated secret key and uploading a fake file. Reliability of the proposed scheme against such attacks should be retained as well.

3.2.2 Solution Overview

The goal is to help the cloud server enjoy the cost savings offered by data deduplication, while also giving data owners a guarantee that their data is kept confidential against the untrusted cloud server and the unauthorized users. Specifically, our motivation is to design an efficient and secure cross-user data deduplication scheme for cloud storage services. For this, we have primarily constructed, based on PEKS (Public-key Encryption with Keyword Search) [9], two equality predicate encryptions schemes in the symmetric-key setting: 1) SEPE (Symmetric-key Equality Predicate Encryption) which is a basic equality predicate encryption scheme and 2) SEPE$_n$ which is an extended scheme of SEPE that has one-to-n tokens mapping ($n \geq 1$). That is, SEPE$_n$ allows n possible tokens to be matched the corresponding encrypted file. Our constructions allow the cloud server in possession of a token associated with each file to know the equivalence relations between the encrypted files without knowledge of the file content.

In the proposed scheme, a data owner generates ciphertexts of a file and the corresponding tokens using both of SEPE and SEPE$_n$ before uploading the file to the cloud server. Then the cloud server retrieves an identical file in the storage by comparing these tokens with stored ciphertexts. If an identical file has existed in the storage, using SEPE, the cloud server will always perform server-side deduplication even without knowing the file content. On the other hand, SEPE$_n$ will enables client-side deduplication with some probability less than 1. This is because just one out of n possible tokens will be correctly verified with the stored file. Hence, using SEPE$_n$ gives online-guessing adversaries substantial search complexity so that no adversary with probabilistic and polynomially bounded computational resources can recover the information. With help of SEPE and SEPE$_n$, the proposed scheme provides data security while also enabling deduplication either of at server side or at client side in a hybrid manner. Some network transmission overhead may be introduced due to the randomized occurrence of client-side deduplication, but the overhead can be minimized while ensuring the required security as will be discussed later.

3.3 Symmetric-key Equality Predicate Encryption

In this section, symmetric-key equality predicate encryption and its security requirement are formally defined. Then, the constructions of two encryption schemes, SEPE and SEPE$_n$, which fulfill the definition and security requirement are presented. The security of the constructed schemes is also proved under the cryptographic complexity assumptions rigorously.

3.3.1 Definition

We first describe the concept of predicate encryption before giving the definition of symmetric-key equality predicate encryption. By definitions in [35] and [33], predicate encryption is a kind of functional encryption system in which a token associated with a function allows a user to evaluate the function over the encrypted data. In predicate encryption scheme, an encryption of a plaintext M can be evaluated using a token associated with a predicate to learn whether M satisfies the predicate. Symmetric-key equality predicate encryption is a predicate encryption scheme in the symmetric-key setting that allows to evaluate an equality predicate over a ciphertext of M given a token of another plaintext M' to learn whether M and M' are equal. Symmetric-key setting means that both of encryption and token generation are computed with the same secret key. Note that PEKS [9] is an equality predicate encryption scheme in the public-key setting.

Definition 1. *A symmetric-key equality predicate encryption scheme consists of the following probabilistic and polynomial time algorithms:*

Initialize(1^λ): Take as input a security parameter 1^λ and output a global parameter Param. For brevity, the global parameter Param output by $Initialize$ algorithm is omitted below.

$KeyGen(aux)$: Output a secret key SK. An auxiliary input aux may be used in generating the secret key.

Encrypt(SK, M): Take as input a secret key SK and a plaintext M and output a ciphertext CT.

GenToken(SK, M): Take as input a secret key SK and a plaintext M and output a token TK.

Test(TK, CT): Take as input TK=GenToken(SK, M') and CT=Encrypt(SK, M), and output 'Yes' if $M = M'$ and 'No' otherwise.

3.3.2 Security Requirement

The required security of symmetric-key equality predicate encryption scheme is an adaptive chosen plaintext security. We define the security against an active adversary who can access an encryption oracle and also obtain tokens TK for any plaintext M of his choice. Even under such an attack, the adversary is not allowed to distinguish an encryption of a plaintext M_0 from an encryption of a plaintext M_1. Now we define the security game between a challenger and the adversary $\mathcal{A}$, as below:

Setup: Challenger runs the *Initialize* algorithm to generate a global parameter Param and the *KeyGen* algorithm to generate a secret key SK. It gives the global parameter to the adversary and keeps SK to itself.

Phase 1: The adversary can adaptively ask the challenger for ciphertext CT or token TK for any plaintext $M \in \{0,1\}^*$ of his choice.

Challenge: Once $\mathcal{A}$ decides that Phase 1 is over, $\mathcal{A}$ sends two challenging plaintexts M_0, M_1 to the challenger. The restriction is that $\mathcal{A}$ did not previously ask for the ciphertexts or the tokens of M_0 and M_1. The challenger picks a random $b \in \{0,1\}$ and gives $\mathcal{A}$ a CT of M_b.

Phase 2: $\mathcal{A}$ continues to ask for ciphertexts CT or tokens TK for any plaintext M of his choice except M_0, M_1.

Guess: Finally, $\mathcal{A}$ outputs $b' \in \{0,1\}$ and wins the game if $b = b'$.

We define the adversary $\mathcal{A}$'s advantage in breaking the symmetric-key equality predicate encryption scheme as

$$\mathrm{Adv}_{\mathcal{A}} = \left| \Pr\{b = b'\} - \frac{1}{2} \right| .$$

Definition 2. *A symmetric-key equality predicate encryption scheme is semantically secure against an adaptive chosen plaintext attack if for any probabilistic and polynomial time (PPT) adversary $\mathcal{A}$, the advantage of $\mathcal{A}$ in winning the game is negligible in λ.*

3.3.3 Cryptographic Background

In this section, the bilinear maps is briefly reviewed and the complexity assumption, on which the proposed constructions are built, is presented.

Bilinear Maps

The proposed constructions are based on some facts about groups with efficiently computable bilinear maps. Let $\mathbb{G}$ and $\mathbb{G}_T$ be two multiplicative cyclic groups of prime order p. Let g be a generator of $\mathbb{G}$. A bilinear map is an injective function $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ with following properties:

1. *Bilinearity*: for all $u, v \in \mathbb{G}$ and $a, b \in \mathbb{Z}_p^*$, we have $e(u^a, v^b) = e(u, v)^{ab}$.
2. *Non-degeneracy*: $e(g, g) \neq 1$.
3. *Computability*: There is an efficient algorithm to compute $e(u, v)$ for $\forall u, v \in \mathbb{G}$.

Bilinear Diffie-Hellman (BDH) Assumption

Let $a, b, c \in \mathbb{Z}_p^*$ be chosen at random and g be a generator of $\mathbb{G}$. The Bilinear Diffie-Hellman problem is to compute $e(g, g)^{abc} \in \mathbb{G}_T$ given $g, g^a, g^b, g^c \in \mathbb{G}$ as input. The BDH assumption [36][37] states that no PPT algorithm can solve the BDH problem with non-negligible advantage.

3.3.4 Constructions

Based on Bilinear Diffie-Hellman assumption in the random oracle model, two constructions of symmetric-key equality predicate encryption scheme are given: (1) SEPE which is a basic construction of symmetric-key equality predicate encryption scheme and (2) SEPE$_n$ which has one-to-n tokens mapping so that n multiple tokens are possible to the corresponding ciphertext. The following hash functions are needed; $H_0 : \{0, 1\}^* \rightarrow \mathbb{Z}_p^*$, $H_1 : \{0, 1\}^* \rightarrow \mathbb{G}$, $H_2^i : \{0, 1\}^* \rightarrow \mathbb{Z}_p^*$, where $i \in \mathbb{N}$ is an index and $H_3 : \mathbb{G}_T \rightarrow \{0, 1\}^{\log p}$. H_2^i can be easily constructed from a keyed hash algorithm like MAC in which an index i is used as a key.

Construction of SEPE

SEPE scheme consists of the following algorithms.

Initialize(1^λ): The initialize algorithm randomly chooses a prime p with the bit length λ and generates a bilinear group $\mathbb{G}$ of order p with its generator g. Then, it outputs a tuple $\mathsf{Param}=\langle p, g, \mathbb{G}, \mathbb{G}_T \rangle$ as a global parameter. Note that Param is taken as an input implicitly next algorithms.

$KeyGen(aux)$: The key generating algorithm takes a binary string aux with arbitrary length as an auxiliary input and computes $SK = H_0(aux) \in \mathbb{Z}_p^*$. Then, it outputs SK as a secret key.

$Encrypt(SK, M)$: Let us denote $SK{=}\alpha \in \mathbb{Z}_p^*$. The encryption algorithm first chooses $r \in \mathbb{Z}_p^*$ at random and then computes a ciphertext $CT = \langle h^r, g^r, H_3(e(h, g^{\alpha})^r)\rangle$, where $h = H_1(M) \in \mathbb{G}$.

$GenToken(SK, M)$: Let us denote $SK{=}\alpha \in \mathbb{Z}_p^*$. The algorithm first chooses $k \in \mathbb{Z}_p^*$ at random and then computes a token of a plaintext M as $TK = \langle h^{\alpha+k}, g^k\rangle$, where $h = H_1(M) \in \mathbb{G}$.

$Test(TK, CT)$: We assume that $CT = \langle c_1, c_2, c_3\rangle$ for a plaintext M and $TK = \langle t_1, t_2\rangle$ for a plaintext M'. The test algorithm computes

$$\gamma = e(t_1, c_2)/e(c_1, t_2).$$

If $c_3 = H_3(\gamma)$, it outputs 'Yes' ; if not, it outputs 'No'.

The algorithm *KeyGen* generates a secret key from aux which is distributed over $\{0,1\}^*$. Let us denote the min-entropy of the distribution of aux by H_{∞}^{aux}.

Theorem 1. *SEPE described above is semantically secure against an adaptively chosen plaintext attack in the random oracle model assuming that BDH is intractable and H_{∞}^{aux} is a polynomial in λ.*

Proof. Suppose $\mathcal{A}$ is an adversary algorithm that tries to break SEPE. $\mathcal{A}$ may run in two ways; (1) guessing a binary string aux to extract the secret key and (2) performing the security game described in Section 3.3.2.

Let us denote by $p(\cdot)$ a polynomial function. Since the min-entropy of the distribution of aux is $H_{\infty}^{aux} = p(\lambda)$, $\mathcal{A}$'s probability of guessing the correct aux is $2^{-p(\lambda)}$. Thus, extracting the correct secrete key is infeasible.

The only way to break SEPE is by the security game. Suppose that $\mathcal{A}$ has advantage ϵ in breaking SEPE by running the security game. We show that an algorithm $\mathcal{B}$ that solves the BDH problem can be constructed using algorithm $\mathcal{A}$. Suppose $\mathcal{A}$ makes at most q_C ciphertext queries, q_T tokens queries and at most q_{H_3} hash function queries to H_3. Then, the advantage of algorithm $\mathcal{B}$ is at least $\epsilon' = \epsilon/eq_{H_3}(q_T + q_C)$, where e is the base of the natural logarithm. If the BDH assumption holds in $\mathbb{G}$, then ϵ' is negligible in λ and consequently ϵ must be negligible in λ. Let g be a generator of $\mathbb{G}$. Algorithm $\mathcal{B}$ is given $g, u_1 = g^{\alpha}, u_2 = g^{\beta}, u_3 = g^{\gamma} \in \mathbb{G}$. $\mathcal{B}$ simulates the challenger and interacts with algorithm $\mathcal{A}$ to output $v = e(g, g)^{\alpha\beta\gamma} \in \mathbb{G}_T$ as follows:

Setup: Algorithm $\mathcal{B}$ gives $\mathcal{A}$ the global parameter: $\langle p,\, g,\, \mathbb{G}, \mathbb{G}_T\rangle$.

H_1-queries: Algorithm $\mathcal{A}$ can make queries to the random oracle H_1 at any time. $\mathcal{B}$ maintains the H_1-list which is a list of tuples $\langle M_i, h_i, a_i, k_i, r_i, c_i \rangle$ to respond to the query. The list is initially empty. Here i represents the sequence of the queries $(1 \leq i \leq q_C + q_T)$. For each query $M_i \in \{0,1\}^*$, algorithm $\mathcal{B}$ responds as follows:

1. If the query M_i is already in the H_1-list in a tuple $\langle M_i, h_i, a_i, k_i, r_i, c_i \rangle$ then $\mathcal{B}$ responds with $H_1(M_i)$=$h_i \in \mathbb{G}$.

2. Otherwise, $\mathcal{B}$ picks a random $c_i \in \{0,1\}$ so that $\Pr\{c_i = 0\}$=$1/(q_C + q_T + 1)$, and also picks $a_i, k_i, r_i \in \mathbb{Z}_p^*$ at random.

   If $c_i = 0$, then $\mathcal{B}$ computes $h_i = u_2 g^{a_i}$.

   If $c_i = 1$, then $\mathcal{B}$ computes $h_i = g^{a_i}$.

3. Algorithm $\mathcal{B}$ appends the tuple $\langle M_i, h_i, a_i, k_i, r_i, c_i \rangle$ to the H_1-list and sends $H_1(M_i)$=h_i to $\mathcal{A}$.

H_3-queries: To respond to these queries, $\mathcal{B}$ maintains a list of tuples $\langle t_i, V_i \rangle$ called the H_3-list. The list is initially empty. If the query t_i already appears on the H_3-list then $\mathcal{B}$ responds with $H_3(t_i)$=V_i. Otherwise, $\mathcal{B}$ responds with $H_3(t_i) = V$ by generating $V \in \mathbb{G}_T$ at random and adds the new tuple $\langle t_i, V \rangle$ to the H_3-list.

Ciphertext queries: When $\mathcal{A}$ issues a query for the ciphertext of M_i, $\mathcal{B}$ responds as follows:

1. $\mathcal{B}$ responds to H_1-queries by running the above algorithm to obtain h_i=$H_1(M_i)$. If c_i=0 then $\mathcal{B}$ outputs $\perp$ and halts. If c_i=1 then $h_i = g^{a_i} \in \mathbb{G}$. Construct $h_i^{r_i}, g_i^{r_i}$ and E=$\mathrm{e}(h_i, u_1)^{r_i}$.

2. $\mathcal{B}$ also gets $H_3(E)$ for the query E by running the above algorithm and gives the correct ciphertext CT= $\langle h_i^{r_i}, g_i^{r_i}, H_3(E) \rangle$ to $\mathcal{A}$.

Token queries: When $\mathcal{A}$ issues a query for the token corresponding to M_i, $\mathcal{B}$ responds as follows:

1. Similar to the ciphertext query, $\mathcal{B}$ gets h_i=$H_1(M_i)$ for M_i by running the above algorithm.

2. If c_i=0 then $\mathcal{B}$ outputs $\perp$ and halts. If c_i=1 then $h_i = g^{a_i} \in \mathbb{G}$. $\mathcal{B}$ gives the token $TK = \langle h_i^{k_i} u_1^{a_i}, g^{k_i} \rangle$ to algorithm $\mathcal{A}$. Observe that TK is correct because $h_i^{k_i} u_1^{a_i} = h_i^{k_i} h_i^{\alpha} = h_i^{k_i + \alpha}$.

Challenge: Algorithm $\mathcal{A}$ produces a pair of challenging plaintext M_0 and M_1. $\mathcal{B}$ generates the challenge as follows:

1. Algorithm $\mathcal{B}$ runs the above algorithm for responding to H_1-queries to obtain a $h_0, h_1 \in \mathbb{G}$ such that $H_1(M_0)=h_0$ and $H_1(M_1)=h_1$. If both $c_0 = 1$ for h_0 and $c_1 = 1$ for h_1 then $\mathcal{B}$ outputs $\perp$ and halts.

2. Otherwise, at least one of c_0, c_1 is equal to 0. Algorithm $\mathcal{B}$ picks a $b \in \{0, 1\}$ at random such that $c_b = 0$.

3. Algorithm $\mathcal{B}$ responds with the challenge ciphertext $CT=\langle I,\ u_3,\ J\rangle$ for a random $I \in \mathbb{G}$ and $J \in \mathbb{G}_T$. Observe that the challenge implicitly defines I, J as follows:

$$\begin{aligned}
I &= H_1(M_b)^{\gamma}, \\
J &= H_3(e(H_1(M_b), u_1^{\gamma})) = H_3(e(u_2 g^{a_b}, g^{\alpha\gamma})) \\
&= H_3(e(g, g)^{\alpha\gamma(\beta+a_b)}).
\end{aligned}$$

Hence, the challenge is a valid ciphertext for M_b.

More ciphertexts, token queries: $\mathcal{A}$ continues to ask for the ciphertext or the token of M_i of his choice except M_0, M_1. Algorithm $\mathcal{B}$ responds to these queries as before.

Output: Finally, $\mathcal{A}$ outputs its guess $b \in \{0, 1\}$. Then, $\mathcal{B}$ picks randomly a tuple $\langle t, V\rangle$ from the H_3-list and outputs $t/e(u_1, u_3)^{a_b}$ as its guess for $e(g, g)^{\alpha\beta\gamma}$.

This completes the description of algorithm $\mathcal{B}$. Now we analyze the probability that $\mathcal{B}$ does not output $\perp$ during the simulation. We define two events; E_1 denotes an event that $\mathcal{B}$ does not output $\perp$ during token or ciphertext queries, and E_2 denotes an event that $\mathcal{B}$ does not output $\perp$ during the challenge phase. The probability that a ciphertext or a token query cause $\mathcal{B}$ to output $\perp$ is $\Pr\{c_i = 0\} = 1/(q_C + q_T + 1)$. Since $\mathcal{A}$ makes at most $q_C + q_T$ of all queries the probability that event E_1 occurs is at least $(1 - 1/(q_C + q_T + 1))^{(q_C+q_T)} \geq 1/e$. That is, $\Pr\{E_1\} \geq 1/e$. Also, $\mathcal{B}$ will output $\perp$ in the challenge phase if $\mathcal{A}$ produces M_0, M_1 such that $c_0 = c_1 = 1$. Since $\Pr\{c_i = 0\} = 1/(q_C + q_T + 1)$ for $i = 0, 1$, and the two values are independent, $\Pr\{c_0 = c_1 = 1\} = (1 - 1/(q_C + q_T + 1))^2 \leq 1 - 1/(q_C + q_T)$. Therefore, $\Pr\{E_2\} = 1 - \Pr\{c_0 = c_1 = 1\} \geq 1/(q_C + q_T)$. These two events E_1 and E_2 are independent because $\mathcal{A}$ can never issue a ciphertext or token query for the challenge plaintext, M_0 and M_1. Therefore, the probability that $\mathcal{B}$ does not output $\perp$ during the simulation is $\Pr\{E_1 \wedge E_2\} = \Pr\{E_1\}\Pr\{E_2\} \geq 1/e(q_C + q_T)$.

Next, assuming that $\mathcal{B}$ does not output $\perp$, we argue that during the simulation $\mathcal{A}$ issues a query for $H_3(e(H_1(M_b), u_1^{\gamma}))$ with probability at least ϵ as shown in the proof of Theorem 3.1 of [9], where M_b is the challenge plaintext. That is, the value $H_3(e(H_1(M_b), u_1^{\gamma})) = e(g^{\beta+a_b}, g)^{\alpha\gamma}$ will appear on some tuple in the H_3-list with probability at least ϵ. Algorithm $\mathcal{B}$ will choose the correct tuple with probability at least $1/q_{H_3}$ and therefore, it will produce the correct answer with probability at least ϵ/q_{H_3} assuming that algorithm $\mathcal{B}$ does not output $\perp$.

Consequently, since $\mathcal{B}$ does not output $\perp$ with probability at least $1/e(q_C + q_T)$ the probability that algorithm $\mathcal{B}$ succeeds overall is at least $\epsilon' = \epsilon/eq_{H_3}(q_C + q_T)$.

□

Construction of SEPE$_n$

SEPE$_n$ scheme consists of the following algorithms.

Initialize$_n$ $(1^{\lambda'})$: The initialize algorithm parses an input $1^{\lambda'}$ in a polynomial time as a tuple $\langle 1^{\lambda}, d\rangle$ that contains two security parameters 1^{λ} and d. The algorithm randomly chooses a prime p with the bit length λ and generates $\mathbb{G}$ of order p with its generator g. Then, it outputs $\mathsf{Param}{=}\langle p, g, \mathbb{G}, \mathbb{G}_T, d\rangle$ as a global parameter. Note that the security parameter d inherently defines a set of permutations Φ in which a permutation domain is $D = \{1, 2, \ldots, d\} \subset \mathbb{N}$.

KeyGen$_n$ (aux): The key generating algorithm takes a binary string aux with arbitrary length as an auxiliary input and outputs a secret key $SK{=}\langle \alpha, \pi\rangle$ by computing $\alpha = H_0(aux) \in \mathbb{Z}_p^*$ and choosing randomly π from Φ.

Encrypt$_n$ (SK, M): Let us denote $SK{=}\langle \alpha, \pi\rangle$, where $\alpha \in \mathbb{Z}_p^*$ and $\pi : D \to D \in \Phi$. The encrypt algorithm first chooses $r \in \mathbb{Z}_p^*$ at random and then computes $h \in \mathbb{G}$ by using two vectors $\vec{u} = (1, 2, \ldots, d)$ and $\vec{v}_\pi = (H_2^{\pi(1)}(M), H_2^{\pi(2)}(M), \ldots, H_2^{\pi(d)}(M))$,

$$h = g^{\vec{u}\cdot\vec{v}_\pi}, \text{ where } \vec{u}\cdot\vec{v}_\pi = \sum_{i\in D} i \cdot H_2^{\pi(i)}(M).$$

The ciphertext is $CT = \langle h^r, g^r, H_3(e(h, g^{\alpha})^r)\rangle$.

GenToken$_n$ (SK, M): Let us denote $SK{=}\langle \alpha, \pi\rangle$. The algorithm first chooses $k \in \mathbb{Z}_p^*$ at random and then computes $h = g^{\vec{u}\cdot\vec{v}_\pi} = g^{\sum_{i\in D} i\cdot H_2^{\pi(i)}(M)}$. The token of the plaintext M is $TK = \langle h^{k+\alpha}, g^k\rangle$.

Test$_n$ (TK, CT): We assume that $CT = \langle c_1, c_2, c_3\rangle$ for a plaintext M and $TK = \langle t_1, t_2\rangle$ for a plaintext M'. The test algorithm computes

$$\gamma = e(t_1, c_2)/e(c_1, t_2).$$

If $c_3 = H_3(\gamma)$, it outputs 'Yes' ; if not, it outputs 'No'.

Table 3.1: Comparison of predicate encryption schemes

	PEKS [9]	Shen *et al.* [33]	Blundo *et al.* [34]	SEPE (SEPE$_n$)
Secret key environment	public-key	symmetric-key	symmetric-key	symmetric-key
Predicate type	equality	multiple	multiple	equality
Cross-user deduplication	No	Yes	Yes	Yes
One-to-n token mapping	No	No	No	Yes
Ciphertext size	$\|\mathbb{G}\| + \log g$	$\geq 4\|\mathbb{G}\|$	$\geq 2\|\mathbb{G}\| + \|\mathbb{G}_T\|$	$2\|\mathbb{G}\| + \log g$
Token size	$\|\mathbb{G}\|$	$\geq 4\|\mathbb{G}\|$	$\geq 2\|\mathbb{G}\|$	$2\|\mathbb{G}\|$
Encrypt algorithm cost	2e+p	$\geq$ 8m + 8e	$\geq$ 3e	3e+p
GenToken algorithm cost	e	$\geq$ 8m + 8e	$\geq$ 2e	2e
Test algorithm cost	p	$\geq$ 3m + 4p	$\geq$ 2m + 2p	m+2p

$|\mathbb{G}|$:size of an element in $\mathbb{G}$, $|\mathbb{G}_T|$:size of an element in $\mathbb{G}_T$, g: order of $\mathbb{G}$,

m:multiplication(division), e:exponentiation, p:pairing

Theorem 2. *SEPE$_n$ described above is semantically secure against an adaptively chosen plaintext attack in the random oracle model assuming that BDH is intractable and H_∞^{aux} is a polynomial in λ.*

Proof. The difference between SEPE and SEPE$_n$ is that in SEPE$_n$ scheme security parameter d, a random permutation π and hash functions H_2^i are additionally defined. Without loss of generality, we can assume that the security parameter d is a constant and a random permutation π is fixed. Then the difference between SEPE and SEPE$_n$ lies only in the computation of h. In the SEPE$_n$ scheme, h is calculated as $h = g^s$, where $s = \vec{u} \cdot \vec{v}_\pi = \sum_{i \in D} iH_2^{\pi(i)}(M)$, and $H_2^i : \{0,1\}^* \to \mathbb{Z}_p^*$ are d hash functions, while h is computed as h=$H_1(M)$ in the SEPE scheme, where H_1 is a hash function $H_1 : \{0,1\}^* \to \mathbb{G}$. In the random oracle model, all hash functions are treated as the random oracle and the distribution of hashed values is uniform. In the SEPE$_n$ scheme, evaluations of the random oracles H_2^i are independent with each other. Hence, the distribution of s is uniform in $\mathbb{N}$, which implies that the distribution of $h = g^s$ is also uniform in $\mathbb{G}$. From the view of an adversary, these two schemes are statistically indistinguishable because in the SEPE scheme the value of hash function H_1 is also distributed uniformly in $\mathbb{G}$. Therefore, assuming SEPE scheme is semantically secure against an adaptively chosen plaintext attack, SEPE$_n$ scheme is also semantically secure against the attack. And, by **Theorem 1**, SEPE$_n$ scheme is secure. $\square$

3.3.5 Discussion

We discuss the effectiveness of our constructions, SEPE and SEPE$_n$, by comparing with several predicate encryption schemes that are similar to ours. Table 3.1 summarizes the comparisons of predicate encryption schemes.

PEKS [9] is a public-key equality predicate encryption scheme that allows to identify equality relations among ciphertexts and tokens. PEKS is designed to be run in the public-key environment, and thus the equality of ciphertexts that are encrypted with one's public key can only be tested with tokens that are generated with the corresponding private key. This public-key setting is not suitable for cross-user data deduplication, in which the equality test should be performed across ciphertexts and tokens that are generated with different users' secret keys. For cloud storage services, it is more desirable to perform data deduplication across multiple users' outsourced data than deduplicating only over a user's data, since it gives more chance of eliminating the redundancy and saves more resources of the cloud server.

Cross-user deduplication can be implemented by a symmetric-key type of predicate encryption algorithm. Shen *et al.*[33] and Blundo *et al.*[34] proposed symmetric-key predicate encryption schemes which support more general predicates such as a comparison predicate ($x \geq a$), a subset predicate ($x \in S$), and arbitrary conjunctive predicates ($P_1 \wedge P_2 \wedge \ldots \wedge P_l$). Both schemes also support predicate privacy, which is a security property that a token reveals no information about the predicate that it contains. For data deduplication that only the equality predicate is sufficient, however, the feature that supports multiple predicates is not necessary, and is inefficient in terms of ciphertext size (token size) and computation time of encryption, token generation and predicate testing. Predicate privacy is also unnecessary for data deduplication, since only one type of predicate is required.

SEPE and SEPE$_n$ are designed to be more suitable for data deduplication. The symmetric-key based construction makes it applicable to cross-user deduplication that eliminates redundancy of data across multiple users. Moreover, SEPE (SEPE$_n$) uses less storage resources (*i.e.,* size of ciphertexts and tokens) and incurs less computation, particularly in *Test* algorithm that causes most of the computational burden in the cloud server, compared with other symmetric-key predicate encryption schemes. One-to-n token mapping of SEPE$_n$ is another feature that makes our constructions more useful for designing secure data deduplication.

3.4 Data Deduplication Scheme Construction

3.4.1 Definition and Notation

We state our definitions and notations. Table 3.2 gives the description of notations to be used in the proposed scheme. For each file, a data owner assigns a globally unique file id fid and encrypts with a set of randomly chosen symmetric encryption keys FEK=$\{$FEK$_1$,FEK$_2$,...,FEK$_l\}$. Also, FEK is encrypted with another secret key α. The cloud server maintains a search index $\mathcal{SI}$ to retrieve a stored file in the storage. $\mathcal{SI}$ is a list of entries in which each entry corresponds to a file entity logically, and comprises a set of search keys and an index value. Search keys include a file id fid and two predicate ciphertexts CT_{fid}, $CT_{n,fid}$ which are encrypted with SEPE and SEPE$_n$, respectively. Files in the storage are retrieved by the cloud server with one of search keys.

The usage of a search key depends on a type of file operation requested by users. An index value contains an address $addr_{File}$ at which the file is physically stored in the system. Each $\mathcal{SI}$

entry has its address $addr_{Index}$ and can be referenced through the address. Fig. 3.2 shows the format of a $\mathcal{SI}$ entry and the structure of file storage.

For data reliability, an erasure code is used in the proposed scheme. An erasure (x, y)-code is an encoding function $\mathcal{E} : X \to Y$ that transforms a message X comprised of x symbols to the code Y of $x + y$ redundant symbols such that any x of them are sufficient to reconstruct the original message X. The number of losses that the erasure code $\mathcal{E}$ can sustain is y, while the redundancy factor is $\mathsf{RD} = (x + y)/x$.

Table 3.2: Notations used in the proposed scheme description

Notation	**Description**
$\text{FEK}_1, \ldots, \text{FEK}_l$	symmetric file encryption keys
$\mathcal{SI}$	search index for file retrieval in the cloud storage
fid	a unique id assigned to each file
$addr_{obj}$	address at which an object obj resides in the system
CT_{fid}	SEPE ciphertext of a file of fid
$CT_{n,fid}$	SEPE$_n$ ciphertext of a file of fid
$CT^{[i]}_{fid}$	i-th element of CT_{fid} (i=1,2,3)
$CT^{[i]}_{n,fid}$	i-th element of $CT_{n,fid}$ (i=1,2,3)
TK_{fid}	token which corresponds to ciphertext CT_{fid}
$TK_{n,fid}$	token which corresponds to ciphertext $CT_{n,fid}$
$TK^{[i]}_{fid}$	i-th element of TK_{fid} (i=1,2)
$TK^{[i]}_{n,fid}$	i-th element of $TK_{n,fid}$ (i=1,2)
$\{M\}_{key}$	ciphertext of M encrypted by a symmetric encryption algorithm with an encryption key key
$<i_1, i_2, \ldots, i_n>$	tuple of n elements $i_1, i_2, \ldots, i_n$

3.4.2 Scheme Description

The proposed scheme is composed of several system level operations: *System Setup*, *New File Upload*, *File Access*, *File Update* and *File Deletion.*

System Setup: Initially, the cloud server and data owners have agreed two security parameters λ and d. The cloud server calls the algorithm $Initialize_n(\langle \lambda, d \rangle)$ and outputs the global parameter $\mathsf{Param}=\langle p, g, \mathbb{G}, \mathbb{G}_T, d \rangle$. The parameters p, g, $\mathbb{G}$ and $\mathbb{G}_T$ will be used commonly in both of SEPE and SEPE$_n$.

New File Upload: When uploading a file to the cloud server, a data owner proceeds as the following:

1. assign a unique file id fid to this data file.

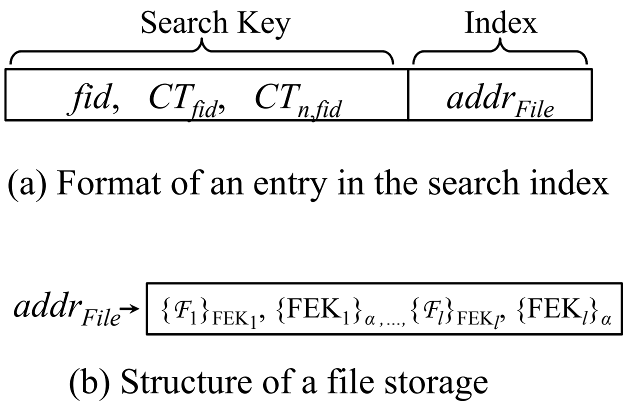

Search Key
Index
fid, CT_{fid}, $CT_{n,fid}$
$addr_{File}$
(a) Format of an entry in the search index
$addr_{File} \rightarrow$ $\{\mathcal{F}_1\}_{\mathrm{FEK}_1}$, $\{\mathrm{FEK}_1\}_{\alpha}$, ..., $\{\mathcal{F}_l\}_{\mathrm{FEK}_l}$, $\{\mathrm{FEK}_l\}_{\alpha}$
(b) Structure of a file storage

Figure 3.2: A search index and a file storage on a cloud server

2. generate a secret key $\langle \alpha, \pi \rangle$ by running $KeyGen_n(File)$, where $File$ is a binary representation of this file content. (α is also assigned to the secret key of SEPE.)
3. compute two tokens TK_{fid} and $TK_{n,fid}$ by running $GenToken(\alpha, File)$ and $GenToken_n(\langle \alpha, \pi \rangle, File)$, respectively.
4. compute two ciphertexts CT_{fid}=$Encrypt(\alpha, File)$ and $CT_{n,fid}$=$Encrypt_n(\langle \alpha, \pi \rangle, File)$.
5. query to the cloud server with a tuple $<fid, TK_{fid}, TK_{n,fid}, CT_{fid}, CT_{n,fid}>$.

To ensure that the tuple is correct, the data owner is requested to compute ciphertexts and tokens by sharing random values $r, k \in \mathbb{Z}_p^*$ between SEPE and SEPE$_n$ at steps 3 and 4 above. (Thus, the tuple should satisfy $CT_{fid}^{[2]} = CT_{n,fid}^{[2]}$ and $TK_{fid}^{[2]} = TK_{n,fid}^{[2]}$.)

(Tuple verification) Upon receiving the tuple, the cloud server first verifies the consistency of a received tuple by checking both $Test_n(TK_{n,fid}, CT_{n,fid})$ and $Test(TK_{fid}, CT_{fid})$ output 'Yes'. Then, the cloud server continues to test the equations (3.1), (3.2) and (3.3).

$$e\left(CT_{fid}^{[1]}, TK_{n,fid}^{[1]}\right) = e\left(CT_{n,fid}^{[1]}, TK_{fid}^{[1]}\right) \tag{3.1}$$

$$CT_{fid}^{[2]} = CT_{n,fid}^{[2]} \tag{3.2}$$

$$TK_{fid}^{[2]} = TK_{n,fid}^{[2]}. \tag{3.3}$$

If one of the verification processes fails, the cloud server reports an error and halts the operation.

(Deduplication) If the tuple is successfully verified, the cloud server retrieves entries in the search index $\mathcal{SI}$ comparing $TK_{n,fid}$ with $CT_{n,fid'}$ of each $\mathcal{SI}$ entry through the algorithm $Test_n(TK_{n,fid}, CT_{n,fid'})$. If a matching entry found, the cloud server gets an address $addr_{File}$ from the entry and creates a new entry $\langle fid, CT_{fid}, CT_{n,fid}, addr_{File} \rangle$. Then the cloud server appends it to $\mathcal{SI}$ and responds the data owner with a message *FileExist* (*i.e.,* client-side deduplication event occurs). If not, the cloud server responds with a message $FileNotExist$.

Upon receiving a message $FileNotExist$, the data owner continues to upload the actual file to the cloud server as the following:

1. transform *File* into $\mathcal{F}$ by running the erasure code $\mathcal{E}$, and split $\mathcal{F}$ according to block length b (*e.g.,* b=512 bits) into l blocks $\mathcal{F}_1, \mathcal{F}_2, ..., \mathcal{F}_l$ (padding is added if the length of $\mathcal{F}$ is not aligned to b).

2. choose randomly encryption keys FEK_1,FEK_2,...,FEK_l from the key space.

3. encrypt the file blocks with an encryption algorithm $C = \langle\{\mathcal{F}_1\}_{\mathrm{FEK}_1}$,$\{\mathrm{FEK}_1\}_\alpha$,...,$\{\mathcal{F}_l\}_{\mathrm{FEK}_l}$,$\{\mathrm{FEK}_l\}_\alpha\rangle$, where α is a hashed value of the file described above.

4. send a tuple $<fid$, $C>$ to the cloud server.

Upon receiving the message $<fid$, $C>$ from the data owner, the cloud server finds the same file comparing TK_{fid} with $CT_{fid'}$ of each $\mathcal{SI}$ entry through $Test(TK_{fid}, CT_{fid'})$.

If the same file already exists in the storage, the cloud server randomly picks l_r $(0 \leq l_r \leq l)$, and replaces l_r arbitrary blocks of the existing file (including their FEKs) by newly uploaded l_r blocks from C.

The cloud server then creates new entry $\langle$ fid, CT_{fid}, $CT_{n,fid}$, $addr_{File}\rangle$, where $addr_{File}$ is from the matching entry and appends it to $\mathcal{SI}$ (*i.e.,* server-side deduplication event occurs). If not found, the cloud server allocates new storage space with new address $addr_{File}$ and puts the encrypted file C into the file system. Then the server creates new $\mathcal{SI}$ entry $\langle fid$, CT_{fid}, $CT_{n,fid}$, $addr_{File}\rangle$ and appends it to $\mathcal{SI}$. Instead of two distinct searches in this operation, it is possible to perform a single search during uploading by retrieving $\mathcal{SI}$ with CT_{fid} and $CT_{n,fid}$ at once.

As described above, the hashed value α acts not only as a plaintext of SEPE and SEPE_n but also as an encryption key of FEK associated with the file itself. Using α as an encryption key helps keeping the system simple, since data owners in possession of same file should also exclusively share a hashed value of the file without any explicit key sharing. If a random encryption key is used instead of the file hash, then we will need costly key management mechanism such as broadcast encryption [23] to share the key among data owners who have a common file. Such a strategy is similar to convergent encryptions, but not the heart of the technique that enables deduplication contrary to convergent encryptions.

File Access: A user authorized to access a file stored in a cloud storage should be in possession of $\langle fid, \alpha\rangle$ of the file. The user sends a file access request message with a file id fid. The cloud server finds the stored file retrieving the search index $\mathcal{SI}$ with a search key fid. If found, the cloud server accesses the file in the file system with the address $addr_{File}$ from the matching entry, and returns $C = \langle\{\mathcal{F}_1\}_{\mathrm{FEK}_1}$,$\{\mathrm{FEK}_1\}_{\alpha}$,...,$\{\mathcal{F}_l\}_{\mathrm{FEK}_l}$,$\{\mathrm{FEK}_l\}_{\alpha}\rangle$ to the user. The user will recover the encrypted file with a decryption key α. The search index $\mathcal{SI}$ can be ordered according to a search key fid thus can be built into a tree-structure. The search operation cost with a search key fid then becomes $\mathrm{O}(\log n)$, where n denotes the size of $\mathcal{SI}$.

File Update: The procedure for updating a file is fully equal to *New File Upload* operation.

File Deletion: The data owner sends a file deletion request with fid. The cloud server searches a matching $\mathcal{SI}$ entry associated with fid. If the searched file is associated with more fids other than the requested one, the cloud server just removes the entry from $\mathcal{SI}$. Otherwise, the server also removes the actual file in the storage.

3.4.3 Efficiency Improvements

We present how to improve efficiency of the proposed scheme and to enhance its performance.

Auxiliary Search Index

The operations of *New File Upload* and *File Update* require a search operation on the search index $\mathcal{SI}$. Since $\mathcal{SI}$ entries cannot be sorted with respect to search keys CT and CT_n, the searching cost is linear to the size of $\mathcal{SI}$ (*i.e.,* the number of files). In this section, we describe a technique that improves the efficiency of the proposed scheme in terms of searching cost. An idea behind the technique is motivated from the fact that the size of uploading files is distributed between a few to giga bytes and the cloud server can get information of the size of a file from the encrypted one. Our idea is to use an auxiliary search index ($\mathcal{AI}$) in addition to $\mathcal{SI}$. In $\mathcal{AI}$, the size of a file is actually a search key, and an index value contains $addr_{Index}$, an address of the $\mathcal{SI}$ entry that indicates the corresponding file. Each $\mathcal{AI}$ entry can be sorted with respect to its search key. With help of a tree structure like B+-tree, an item insertion, deletion and searching operation in $\mathcal{AI}$ can be handled in an efficient and scalable manner.

When a data owner uploads (or updates) a file, the data owner sends a tuple $<size$, fid, TK_{fid}, $TK_{n,fid}$, CT_{fid}, $CT_{n,fid}>$ to the cloud server. Then, the server first looks up $\mathcal{AI}$ entries that correspond to *size*. If found, the cloud server gets matching $\mathcal{SI}$ entries which are addressed by $addr_{Index}$. The number of matching $\mathcal{SI}$ entries may be one or more according to a density of file size distribution. The cloud server then enumerates the matching $\mathcal{SI}$ entries and finds a corresponding entry such that $Test$ (or $Test_n$) algorithm results in 'Yes'. In this technique, using $\mathcal{AI}$ limits efficiently the search scope of $\mathcal{SI}$ thus eventually reduces searching cost to sub-linear complexity. In order to use this technique, the original scheme should be slightly modified. The only difference from the original scheme is that *size* field is added on the above tuple in this technique.

Tradeoff between Security and Network Efficiency

The proposed scheme is designed to be resistant against the online-guessing attack. While an unauthorized user has difficulty in guessing the content of a stored file, usual file upload operation may also have little chance that a client-side deduplication event occurs, which eventually causes the increase of network transmission overhead. In order to avoid losing network transmission efficiency, the proposed scheme allows the security parameter d of SEPE_n to be selected on the tradeoff between the security and the efficiency. The security parameter d denotes a domain size of permutation and actually determines the size of set of all permutations with a domain D such that $|D|=d$. Hence, the lower value of d gives higher probability that client-side deduplication occurs and vice versa.

This tradeoff is reasonable in practical applications. Data owners may upload not only high privacy requiring files that contain personal information but also common files like software setup files. The uploading files may also vary between high and low entropy. In the proposed scheme, the value of d is globally chosen in system setup phase. However, it is not difficult to modify the proposed scheme to allow d be selected for each file in uploading phase. Each file's security parameter d can be decided based on its properties like the entropy or the required level for privacy. File size can be another decision factor of d because it is reasonably assumed that file size and its entropy are highly correlated with each other [3]. For example, a file that requires no privacy or its size is greater than some threshold may be given its security parameter $d = 1$, which implies that client-side deduplication will always occur with probability 1 if the same file has existed in the cloud server.

A quantitative analysis of the relation between the security parameter d and both security and the network transmission overhead will be presented in Section 3.5.2.

3.4.4 Improving Reliability of the Cloud Storage

By applying a general data replication mechanism to the proposed scheme, we can further improve reliability of the scheme against malicious data owners incurring data loss. A data replication mechanism, where replication factor (RP) is r, creates r replicas of a data object and stores them across geographically distributed storage systems.

New File Upload operation of the proposed scheme is extended to utilize such a data replication as follows. When accepting $< fid, C >$ from a data owner, the cloud server searches on $\mathcal{SI}$ by using $Test$ algorithm. If a matching entry is found (*i.e.,* server-side deduplication event occurs), the cloud server gets $addr_{File}$ from the $\mathcal{SI}$ entry and determines the locations of existing replicas for the file. If the number of the existing replicas is less than r, a new one is allocated to store $C = \langle\{\mathcal{F}_1\}_{\text{FEK}_1},\{\text{FEK}_1\}_{\alpha},...,\{\mathcal{F}_l\}_{\text{FEK}_l},\{\text{FEK}_l\}_{\alpha}\rangle$. Otherwise, the cloud server selects one of them and replaces l_r blocks of the chosen replica, where l_r is randomly picked from $\{0, 1, ..., l\}$, by new blocks of C. In other cases, the cloud server follows the original protocol described in the previous section.

With help of an erasure (x, y)-code, corrupted data can be recovered as long as at least $\lceil lx/(x+y) \rceil$ blocks among r replicas stored within the storage sustain the corruption, where l is the number of blocks of the file.

Note that data replication is orthogonal to data deduplication. Goals of removing data redundancy and improving reliability can be achieved simultaneously using both data replication and data deduplication together.

3.5 Evaluation

In this section, we first present the security analysis of the proposed scheme. Then, we present the storage, communication and computation performance analysis.

3.5.1 Security Analysis

Data Confidentiality

In the proposed scheme, files in the cloud storage are encrypted with a symmetric encryption algorithm like AES [51]. Encrypting the file generates a randomized ciphertext, because FEK is randomly chosen from the key space. Therefore any adversary including the cloud server who has only ciphertexts of the file cannot distinguish it from the random data. That is, the adversary does not know any information of the content of a file.

The cloud server will perform data deduplication through running $Test$ and $Test_n$ algorithm with inputs of tokens and ciphertexts of SEPE and SEPE$_n$. Unless tokens are given, the cloud server has negligible advantage in getting any information of the plaintext from its ciphertexts by theorems, **Theorems 1** and **2**. Although the cloud server may have knowledge of equivalence relations among the ciphertexts in the search index, **Theorem 3** says that any PPT algorithm including the test algorithms, $Test$ and $Test_n$, does not reveal any partial information to the cloud server other than the equivalence relation.

Note that the cloud server obviously can infer to the information of file size from the corresponding ciphertext because the file size and its ciphertext size are inherently correlated with each other.

Theorem 3. *For any two plaintexts M and M' such that $M \neq M'$, any PPT algorithm given SEPE (or SEPE$_n$) ciphertexts and tokens of M and M' as inputs cannot output any partial information of the plaintexts M and M' under the random oracle assumption.*

Proof. For the sake of simplicity, let us assume that hash functions used in SEPE and SEPE$_n$ (*i.e.,* H_1, H_2) are denoted as $F : \{0,1\}^* \rightarrow \mathbb{G}$. In computation of the tokens and the ciphertexts, the plaintext M and M' are given as inputs in evaluating the hash function F. By collision resistance property of a cryptographic hash function, it is highly unlikely that $F(M)=F(M')$ for any two different plaintext M and M'. Besides, using the fact that a cryptographic hash function can be modeled as a random oracle [52], both of $F(M)$ and $F(M')$ will be indistinguishable from random in $\mathbb{G}$. Therefore, in the case of $Test$ and $Test_n$ algorithm, an intermediate computation result γ will be indistinguishable from a random data in $\mathbb{G}_T$, and the test algorithm will eventually result in meaningless output 'No'. Another PPT algorithms given ciphertexts and tokens of M and M' as inputs will also eventually output a data that looks like a random from the view of the algorithm. □

Resistance against Online-guessing Attack

Now we analyze security against an adversary who mounts online-guessing attack to figure out the information of interesting file in the cloud storage. The adversary is neither in possession of the file nor its hashed value. However, the adversary may have relatively small search domain for the file because the file has inherent low-entropy property or he has some knowledge of it. For each candidate in the search domain the adversary generates a token TK_n and then queries the token to the cloud server. If a matching ciphertext CT_n exists such that $Test_n(CT_n,TK_n)$ results 'Yes' for the query, the cloud server will respond with a message $FileExist$ indicating deduplication (at client side) event occurs. The adversary will continue querying to the cloud server until receiving $FileExist$.

In the attack scenario, an adversary may select a correct candidate in high likelihood of matching the target file. However, the adversary should query a correct token TK_n for the selected candidate in order to render a deduplication event (at client-side). **Theorem 4** says that a probability that an adversary computes a correct token TK_n such that $Test_n(CT_n, TK_n)$ results in 'Yes' is negligible in the security parameter d. Hence, the proposed scheme is resistant against online-guessing attacks.

Theorem 4. *For any PPT algorithm $\mathcal{A}$, the probability that $\mathcal{A}$ outputs the corresponding token TK_n for given SEPE$_n$ ciphertext CT_n is negligible in the security parameter d.*

Proof. Let us assume a PPT algorithm $\mathcal{A}$ which computes a token TK_n for a corresponding SEPE$_n$ ciphertext CT_n. $\mathcal{A}$ is given d, g, α and CT_n as inputs and tries to find out the correct token TK_n. We also assume that $CT_n = \langle h^r, g^r, H_3(e(h, g^a)^r)\rangle$, where r is chosen at random from $\mathbb{Z}_p^*$. If $\mathcal{A}$ can get h from h^r, it is obvious that $\mathcal{A}$ easily computes the corresponding token TK_n=$\langle h^{k+\alpha}, g^k\rangle$ with given h, where k is chosen randomly from $\mathbb{Z}_p^*$. However, the problem that computes h from h^r in a cyclic group $\mathbb{G}$ is not easier than the discrete logarithm problem (DLP) on elliptic curves which is generally known to be computationally infeasible for any PPT algorithm if p is sufficiently large [37]. Hence, the best way for $\mathcal{A}$ to find out the token is to choose one of possible tokens randomly and test it by running $Test_n$ algorithm. $\mathcal{A}$ can choose a token at random by tossing coins to choose a random permutation $\pi : D \to D$, where $D = \{1, 2, \ldots, d\}$. Now we analyze the probability that $\mathcal{A}$ outputs the correct token

TK_n given the input CT_n. Let us denote a set of all permutations $\pi : D \to D$ as Φ. We assume that CT_n has been computed with a permutation π. That is, $h = g^{\vec{u}\cdot\vec{v}_\pi}$ in CT_n , where $\vec{u}\cdot\vec{v}_\pi = \sum_{i\in D} iH_2^{\pi(i)}(M)$. When algorithm $\mathcal{A}$ chooses randomly a permutation π' from Φ, h is computed as $h = g^{\vec{u}\cdot\vec{v}_{\pi'}}$ in TK_n , where $\vec{u}\cdot\vec{v}_{\pi'} = \sum_{i\in D} iH_2^{\pi'(i)}(M)$. Note that $\vec{u}\cdot\vec{v}_\pi$ and $\vec{u}\cdot\vec{v}_{\pi'}$ also can be represented as $\vec{u}\cdot\vec{v}_\pi = pk + r$ and $\vec{u}\cdot\vec{v}_{\pi'} = pk' + r$, where p is order of $\mathbb{G}$, $k, k' \in \mathbb{N}$ and $0 \le r, r' < p$. Let us denote an event $p(k-1) \le \vec{u}\cdot\vec{v}_{\pi'} < pk$ as E_k. Then, the probability is

$$\begin{aligned}
&\Pr\{\mathcal{A} \text{ outputs the correct } TK_n\}\\
&= \Pr\{\pi' \xleftarrow{R} \Phi \text{ , } Test_n(CT_n, TK_n) = \text{`Yes'} \}\\
&= \Pr\{\pi' \xleftarrow{R} \Phi \text{ , } g^{\vec{u}\cdot\vec{v}_\pi} = g^{\vec{u}\cdot\vec{v}_{\pi'}}\}\\
&= \Pr\{\pi' \xleftarrow{R} \Phi \text{ , } \vec{u}\cdot\vec{v}_\pi = \vec{u}\cdot\vec{v}_{\pi'} \bmod p\}\\
&\le \sum_{k\in\mathbb{N}} \Pr\{r = r' | E_k\} \Pr\{\pi' \xleftarrow{R} \Phi \text{ , } E_k\}\\
&\le \sum_{k\in\mathbb{N}} \Pr\{\pi' \xleftarrow{R} \Phi \text{ , } \pi = \pi'\} \Pr\{\pi' \xleftarrow{R} \Phi \text{ , } E_k\}\\
&= \Pr\{\pi' \xleftarrow{R} \Phi \text{ , } \pi = \pi'\} \sum_{k\in\mathbb{N}} \Pr\{\pi' \xleftarrow{R} \Phi \text{ , } E_k\}\\
&\le \Pr\{\pi' \xleftarrow{R} \Phi \text{ , } \pi = \pi'\}\\
&= \frac{1}{d!} \le \frac{1}{2^{d-1}}.
\end{aligned}$$

Therefore, the probability that $\mathcal{A}$ outputs the corresponding TK_n is negligible in the security parameter d.

□

Collusion-resistance

Unauthorized users and even other data owners who do not have an access right to their interesting file can collude together to access the file. They should know the correct access token $\langle fid, \alpha\rangle$ in order to get the file via *File Access* operation. Hence they will try to compute $\langle fid, \alpha\rangle$ from information such as secret keys or tokens which they have. Assuming that fid is chosen from sufficiently large space and kept secret to the authorized users and data owners, the colluding adversaries cannot compute fid since fid is independent of any information which they have.

They may also try to compute the corresponding tokens TK, TK_n to take advantage of a side channel on client-side deduplication. However, since a permutation π of CT_n corresponding the file was chosen from Φ at random and is independent of any information, it is impossible for the colluding adversaries to get the file by **Theorem 4**.

Resistance against Malicious Data Owner

We analyze security of the proposed scheme against two kinds of attacks, which are launched by a malicious data owner who does not follow the protocol and behaves dishonestly.

Server-side deduplication disturbing attack: Let us consider a malicious data owner $\mathcal{M}$ who tries to disturb server-side deduplication so as to intentionally reduce the available storage resources of the cloud server. During *New File Upload* operation, $\mathcal{M}$ would mount such an attack by composing the wrong tuple $< fid, TK'_{fid}, TK_{n,fid}, CT'_{fid}, CT_{n,fid} >$, where TK'_{fid}, CT'_{fid} are calculated using the randomly generated secret key α' ($\neq \alpha$).

For the proposed scheme, this attack does not succeed, since the correctness of the received tuple is verified by the cloud server before acceptance. Specifically, $Test$ (or $Test_n$) algorithm checks whether a secret key is consistently used for computing both CT_{fid} and TK_{fid} (or $CT_{n,fid}$ and $TK_{n,fid}$). That is, the cloud server verifies the tuple by checking whether all of the following equations holds.

$$e\left(CT^{[1]}_{fid}, TK^{[1]}_{n,fid}\right) = e\left(CT^{[1]}_{n,fid}, TK^{[1]}_{fid}\right)$$
$$CT^{[2]}_{fid} = CT^{[2]}_{n,fid}$$
$$TK^{[2]}_{fid} = TK^{[2]}_{n,fid}.$$

By definition of bilinear map (in Section 3.3.3) and definition of our construction, it is infeasible for $\mathcal{M}$ to compose the wrong tuple such that TK'_{fid}, CT'_{fid} are calculated using randomly generated α', while also satisfying the above equations, as well as allowing $Test(TK'_{fid}, CT'_{fid})$ to output 'Yes'.

Duplicate faking attack: We consider a malicious data owner $\mathcal{M}$ who tries to corrupt other users' data. During *New File Upload* operation, $\mathcal{M}$ may send a tuple $< fid, TK_{fid}, TK_{n,fid}, CT_{fid}, CT_{n,fid} >$ for $File$ correctly, but then upload a fake $< fid, C' >$, where C' is constructed from $File'(\neq File)$ or a wrong secret key $\alpha'(\neq \alpha)$. We analyze security against such an attack in terms of two properties; (i) corruption detectability and (ii) data recoverability.

The proposed scheme ensures that the corruption of $File$ is always detected by data owners or their authorized users who treat the same file. Let us denote a decryption of ciphertext E with a key k by $\mathsf{Dec}(k, E)$, n-th element of a tuple T by $T^{[n]}$, and a decoding function of an erasure code $\mathcal{E}$ by $\mathcal{D}$. Data corruption can be detected as follows.

1. get $C = \langle\{\mathcal{F}_1\}_{\text{FEK}_1}, \{\text{FEK}_1\}_\alpha, ..., \{\mathcal{F}_l\}_{\text{FEK}_l}, \{\text{FEK}_l\}_\alpha\rangle$ via *File Access* operation.

2. for each $T_i = \langle\{\mathcal{F}_i\}_{\text{FEK}_i}, \{\text{FEK}_i\}_\alpha\rangle$ in C, where $1 \leq i \leq l$, compute $v_i = \mathsf{Dec}(\mathsf{Dec}(\alpha, T_i^{[2]}), T_i^{[1]})$.

3. compute $w = \mathcal{D}(v_1||v_2||...||v_l)$, and $z = KeyGen(w)$.

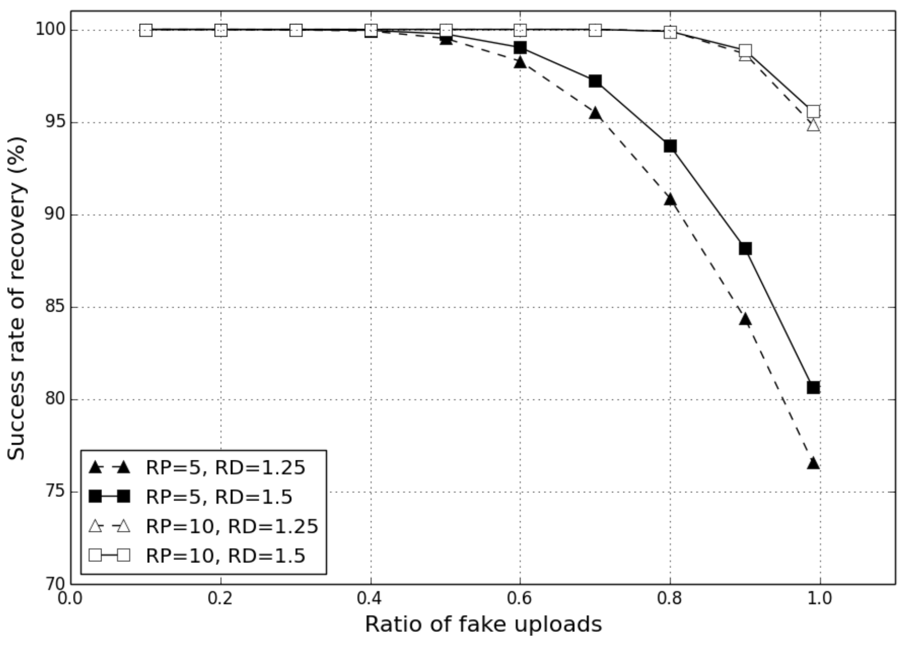


Figure 3.3: File recoverability against fake uploads

4. test $z = \alpha$.

If z is not equal to a secret key α, then the file is corrupted.

With help of data replication, the proposed scheme also ensures that with high probability, a file can be recovered from corruption against fake uploads by $\mathcal{M}$. To justify our argument, we conducted a simulation, in which *New File Upload* operations for the same file (the block size is $b = 512$ and the number of blocks is $l = 16,384$) were performed in 1,000 times including fake uploads among them. The ratio of the fake uploads varies from 0 to 0.99. At each *New File Upload* operation, it was checked that sufficient blocks on the storage remain uncorrupted for recovery.

Fig. 3.3 shows the simulation result. RP refers to a replication factor of data replication mechanism (*i.e.,* the number of replicas), and RD to a redundancy factor of the erasure code. With RP=10 and RD=1.5, the probability of successfully recovering corrupted data is more than 95%.

3.5.2 Performance Analysis

Storage Overhead

When a data owner uploads a file to the cloud server, data deduplication will always occur either at client side or at server side, if the same file exists in the cloud storage. That is, the proposed scheme always removes redundant copies of a file across multiple users, thus keeps the cloud storage optimized in terms of disk space utilization.

Additional storage overhead may be introduced due to the search index. The size of $\mathcal{SI}$ or $\mathcal{AI}$ entry, however, is negligible compared to the size of corresponding file.

Employing data replication, which is discussed in Section 3.4.4, may increase the required storage capacity. In the proposed scheme, however, data replication is run over deduplicated data, and thus is independent of eliminating the redundancy. The amount of increased storage resources does not exceed by the replication factor of a data replication mechanism.

Network Transmission Overhead

The hybrid approach that prevents the online-guessing attack may introduce some network transmission overhead due to unnecessary file uploads. We analyze how much the network bandwidth is actually consumed for the proposed scheme.

For this, we compare the proposed scheme with a basic scheme, which is exactly same to the proposed scheme except that client-side deduplication always occurs (*i.e.,* the security parameter $d = 1$). Note that the basic scheme incurs no network transmission overhead at all. Thus, the difference Δ between the amount of network bandwidth consumed for the proposed scheme and the amount for the basic scheme represents the network transmission overhead of the proposed scheme.

Several events in *New File Upload* operation are defined as follows:

1. E : The file $\mathcal{F}$ exists in the cloud storage.

2. T_P : *New File Upload* operation transmits the actual copy of $\mathcal{F}$ in the proposed scheme.

3. T_B : *New File Upload* operation transmits the actual copy of $\mathcal{F}$ in the basic scheme.

From the security analysis in Section 3.5.1, we have $\Pr\{T_P \mid E\} \geq 1 - \frac{1}{d!}$. Without loss of generality, we assume $\Pr\{T_P \mid E\} = 1 - \frac{1}{d!}$ in this section. Note that $\Pr\{T_B \mid E\}=0$. The probability that the event T_P occurs is

$$\begin{aligned}\Pr\{T_P\} &= \Pr\{T_P \mid E\}\Pr\{E\} + \Pr\{T_P \mid \neg E\}\Pr\{\neg E\} \\ &= \Pr\{T_P \mid E\}\Pr\{E\} + \Pr\{\neg E\} \\ &= \Pr\{T_P \mid E\}\Pr\{E\} + 1 - \Pr\{E\} \\ &= \left(1 - \frac{1}{d!} - 1\right)\Pr\{E\} + 1 = -\frac{\Pr\{E\}}{d!} + 1.\end{aligned}$$

By the similar way, the probability that the event T_B occurs is $\Pr\{T_B\}$=1-$\Pr\{E\}$.

The expected amount of network traffic in uploading the file $\mathcal{F}$ for the proposed scheme and the basic scheme are $s \cdot \Pr\{T_P\}$ and $s \cdot \Pr\{T_B\}$, respectively, where s is the size of $\mathcal{F}$. Thus, the network transmission overhead of the proposed scheme can be represented by the difference Δ

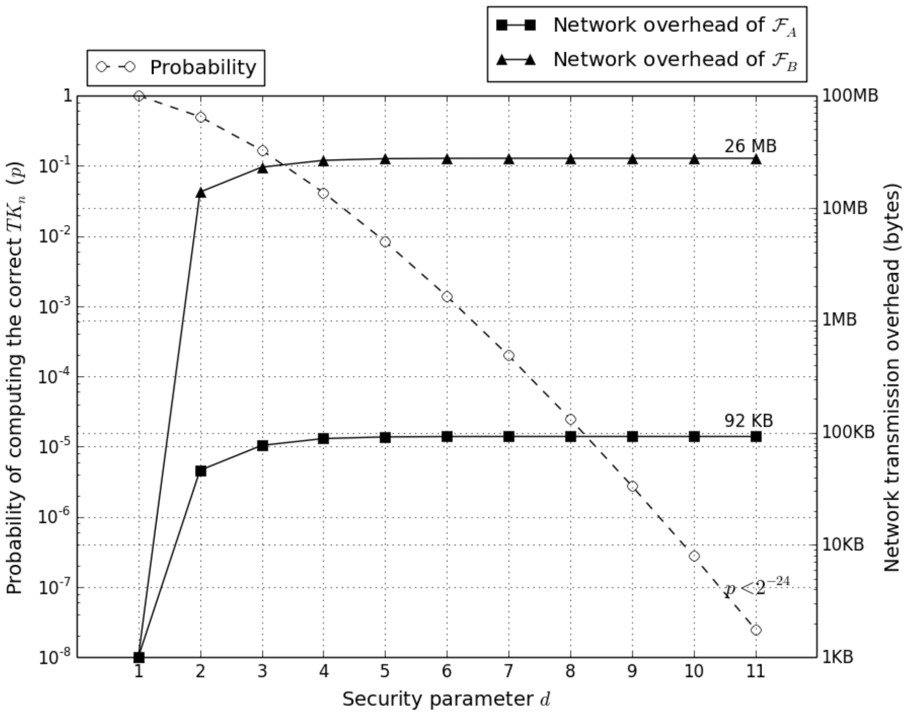


Figure 3.4: Tradeoff between security and the network transmission overhead over the security parameter d (The file size of $\mathcal{F}_A$ and $\mathcal{F}_B$ is 90 MBs, $\Pr\{E_{\mathcal{F}_A}\} = 0.001$ and $\Pr\{E_{\mathcal{F}_B}\} = 0.3$)

$$\Delta = \mid s\Pr\{T_P\} - s\Pr\{T_B\} \mid = s\Pr\{E\}\left(1 - \frac{1}{d!}\right). \tag{3.4}$$

For each file in the cloud storage, the probability $\Pr\{E\}$ will be distributed between 0 to 1 according to its properties, especially to the type of file content. That is, it is easily inferred that common files (*e.g.,* software install files) usually have high $\Pr\{E\}$, and private files, that contain sensitive and personal information, requiring high privacy have low $\Pr\{E\}$. From the above equation, we can observe that as $\Pr\{E\}$ goes to 0, so does Δ, while Δ closes to $s(1-1/d!)$ as $\Pr\{E\}$ goes to 1. This result implies that we can reduce overall network transmission overhead substantially by just only decreasing the security parameter d of common files, while keeping private file's parameter d high, as we described in section 3.4.3.

Let us consider an example that a data owner uploads two files $\mathcal{F}_A$ and $\mathcal{F}_B$, which represent a private file and a common file, respectively. Suppose that the size of both $\mathcal{F}_A$ and $\mathcal{F}_B$ is 90 MBs and the probabilities that the file exists in the cloud storage are $\Pr\{E_{\mathcal{F}_A}\} = 0.001$ and $\Pr\{E_{\mathcal{F}_B}\} = 0.3$. Fig. 3.4 shows the quantitative relation between the security parameter d and both security (*i.e.,* probability $p = \frac{1}{d!}$ that an adversary computes the correct TK_n) and the network transmission overhead ($\Delta_{\mathcal{F}_A}$ for $\mathcal{F}_A$ and $\Delta_{\mathcal{F}_B}$ for $\mathcal{F}_B$). Setting the parameter d to 11 for both $\mathcal{F}_A$ and $\mathcal{F}_B$ raises the overall network transmission overhead ($\Delta_{\mathcal{F}_A} + \Delta_{\mathcal{F}_B}$) to more than 26 MBs. On the other hand, the overall overhead can be reduced to 92 KBs by just only decreasing the common file $\mathcal{F}_B$'s parameter d to 1, while keeping the probability p of the private file $\mathcal{F}_A$ lower than 2^{-24}.

Table 3.3: Computation complexity of the proposed scheme considering two strategies: $\mathcal{SI}$ and $\mathcal{SI}$ with $\mathcal{AI}$ ($\mathcal{SI}$+$\mathcal{AI}$)

Operation	The Proposed Scheme	
	$\mathcal{SI}$	$\mathcal{SI}$+$\mathcal{AI}$
New File Upload	$O(N_s)$	$O(\log N_a + N_{s,a})$
File Update	$O(N_s)$	$O(\log N_a + N_{s,a})$
File Access	$O(\log N_s)$	$O(\log N_s)$

Computation Overhead

Computational burden at the cloud server is mainly introduced by searching over the storage. We analyze the computation complexity for the following operations that perform the searching : *New File Upload*, *File Update*, *File Access.*

In *New File Upload* and *File Update* operation, the cloud server searches the matching file in $\mathcal{SI}$ through two search keys CT and CT_n. For each $\mathcal{SI}$ entry the cloud server compares it with given tokens by computing $Test$ and $Test_n$ algorithms which require two pairings and one multiplication over $\mathbb{G}_T$. Let us denote N_s as the total number of $\mathcal{SI}$ entries, N_a as the total number of $\mathcal{AI}$ entries and $N_{s,a}$ as the number of $\mathcal{SI}$ entries which correspond to the matching $\mathcal{AI}$ entry. The computation complexity of *New File Upload* and *File Update* are shown in Table 3.3. With help of $\mathcal{AI}$, the proposed scheme ($\mathcal{SI}$+$\mathcal{AI}$ in Table 3.3) can have sub-linear complexity in this operations.

In *File Access* operation, $\mathcal{SI}$ is retrieved through the search key fid. Since $\mathcal{SI}$ is sorted according to fid, the searching cost is more efficient than that of above operations as shown in Table 3.3. We also note that no cryptographic operations are needed in comparison over fid.

3.6 Summary

This chapter aims at achieving both cost efficiency and data security in cloud computing. One challenge in this context is to build a data deduplication system which offers cost savings in terms of disk space and network bandwidth utilization, while also providing data security and privacy against the untrusted cloud server and the unauthorized users. In this chapter, we proposed an efficient and secure data deduplication scheme to resolve the challenging issue. In order to achieve both of efficiency and security, we constructed two equality predicate encryption schemes in the symmetric-key setting, on which the proposed data deduplication scheme is built. Our rigorous security proofs show that our proposed scheme is provably secure under cryptographic security models.

Chapter 4. Differentially Private Client-side Data Deduplication Protocol

4.1 Motivation and Contributions

Most cloud storage service providers utilize client-side (or source-based) data deduplication, which is a technique that eliminates duplicate copies of data (or files) across multiple users before uploading them, to save the cost of network bandwidth and disk storage [2]. It has been known that this technique is not secure, since by monitoring and analyzing network traffic, adversaries can use (client-side) deduplication as a side channel to obtain sensitive information of other users' data such as the existence of data or even its content [3][14]. Recently, some solutions have been proposed to prevent such information leakage, but several problems still remain unresolved.

Harnik *et al.*'s scheme [3] and its security-enhanced version [45] obfuscate the occurrence of deduplication by randomizing the event with certain probability. This randomized approach, however, causes huge network bandwidth consumption owing to the unnecessary file upload, thus lowering the effectiveness of the deduplication. In addition, the schemes are based on the strong assumption that all individual files are independent of each other. This leads to our novel attack, denoted by *related-files attack*, which we will propose in this chapter. Security of these approaches [3][45] are weakened by *related-files attack* in which an adversary can take advantage of knowledge of correlations among files. Olivier *et al.* proposed another solution for secure deduplication [46]. Their idea is to run the deduplication protocol at a home router provided by an ISP and mix network traffic for cloud storage with other service traffic. This solution, however, lacks flexibility because the fact that it requires an ISP home router limits its uses to a specific service model.

Allowing efficient client-side deduplication while reducing the risk of information leakage is still an open problem. In this chapter, we address this problem and propose a secure client-side deduplication protocol. Our solution utilizes a storage gateway, which is a network appliance that resides at the customer's premises and provides APIs to access the remote cloud storage server. Since they simplify interactions with cloud storage services, storage gateways are gaining popularity and being deployed as a key component in various cloud service models, such as public, private, or hybrid cloud computing [53]. The key idea of the proposed solution is that a storage gateway handles user requests for file uploads and performs data deduplication on them on behalf of the users (or clients). In this way, the network traffic for each user request is merged together at the storage gateway's WAN (Wide Area Network) interface. Without generating unnecessary network traffic, this approach weakens the link between the deduplication event and the amount of actually transferred. This idea is similar to Olivier *et al.*'s solution [46], but using storage gateways gives flexibility for adapting to various cloud service models.

For robust security, we apply a differentially private mechanism to our proposed protocol. Differential privacy is a security notion that has been presented in database security literature to design privacy enhanced statistical databases [54][55][56]. By exploiting differential privacy, proposed protocol strongly guarantees that the presence or absence of individual files in the cloud storage is hard to infer from any side channel attacks, including *related-files attack*.

The contributions of this chapter are three-fold. First, we discuss the security weakness of the previous schemes [3][45] that are based on the randomized approach. Second, we propose a storage gateway-based solution that guarantees robust security, while providing network efficiency and flexibility. Third, we analyze its security under the definition of differential privacy and evaluate its effectiveness by our experiments.

The rest of this chapter is organized as follows: In Section 4.2, we review the previously proposed schemes and describe their security weakness. In Section 4.3, details of our storage gateway-based deduplication solution are described. In Section 4.4, we present a security analysis and the experiment results of the proposed protocol. Finally, in Section 4.5, we give the summary of this chapter.

4.2 Related-files Attack

4.2.1 *Related-files attack* on Harnik *et al.*'s scheme

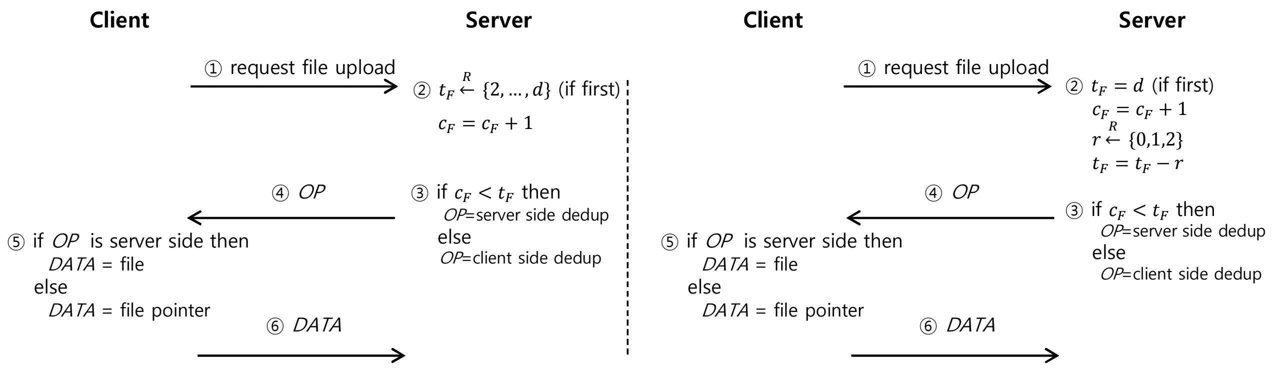
We briefly review Harnik *et al.*'s scheme [3] and present its security weakness.

The protocol description

For every file F, a cloud storage server assigns a threshold t_F chosen randomly in a range $[2, d]$, where d is a security parameter that might be known public. The cloud server keeps a counter c_F which represents the number of previous uploads of a copy of F. When a client uploads a new copy of F, the server performs (client-side) deduplication if at least t_F copies of F have been previously uploaded (*i.e.,* $c_F \geq t_F$) or if the copy is uploaded by an identical user who has previously uploaded F; no deduplication event occurs otherwise. This protocol is illustrated in Fig. 4.1(a).

Compared to the straightforward method, this randomized approach pays some penalty in terms of the network bandwidth utilization, since during the first $t_F - 1$ times of uploading F, a full data of F should be uploaded for hiding the occurrence of deduplication from users including adversaries. Upon receiving F, however, the cloud server always performs server-side deduplication on the data. Thus, in terms of the disk utilization, this approach can achieve the same level of efficiency as the normal deduplication.

The security of Harnik *et al.*'s scheme can be stated by that too much information about



(a) Harnik *et al.*'s protocol (b) Lee *et al.*'s protocol

Figure 4.1: Randomized client-side deduplication protocol

the inclusion of any file F in the cloud storage is not revealed with some probability. In order to analyze its security, two views of the adversaries are compared; one that the file F was already uploaded by another user, and the other that no copy of F has previously been uploaded.

For the sake of analysis, let us consider the following three types of events where the adversary is about to identify whether a copy of F was uploaded.

(i) ($2 < t_F < d$) The adversary finds out that a deduplication event occurs after uploading $2 < t < d$ copies of F. This could be due to two cases:

  (i-1) F already exists, and the adversary uploads $t = t_F - 1$ copies of F

  (i-2) F does not exist and the adversary uploads $t = t_F$ copies of F

(ii) ($t_F = 2$) The adversary finds out that a deduplication event occurs after uploading a single copy of F. It is the case that $t_F = 2$ and a single copy of F has been previously uploaded by another user.

(iii) ($t_F = d$) The adversary finds out that a deduplication event occurs after uploading d copies of F. It is the case that no copy of F was previously uploaded and $t_F = d$.

For F with $2 < t_F < d$, it can be shown that the events of two cases (i-1) and (i-2) occur with equal probabilities. Hence, a deduplication event gives an adversary no information of the existence of F in the storage. On the other hand, for F with $t_F = 2$ or $t_F = d$, the adversary can easily learn the existence of F by uploading just one or d copies of the file. Note that the two events (ii) and (iii) are mutually exclusive and each event happens with probability $\frac{1}{d-1}$. As a result, with probability $1 - \frac{1}{d-1}$, the scheme described above leaks no information which enables an adversary to distinguish between the case that a single copy of F was previously uploaded, and the case that the file was not uploaded.

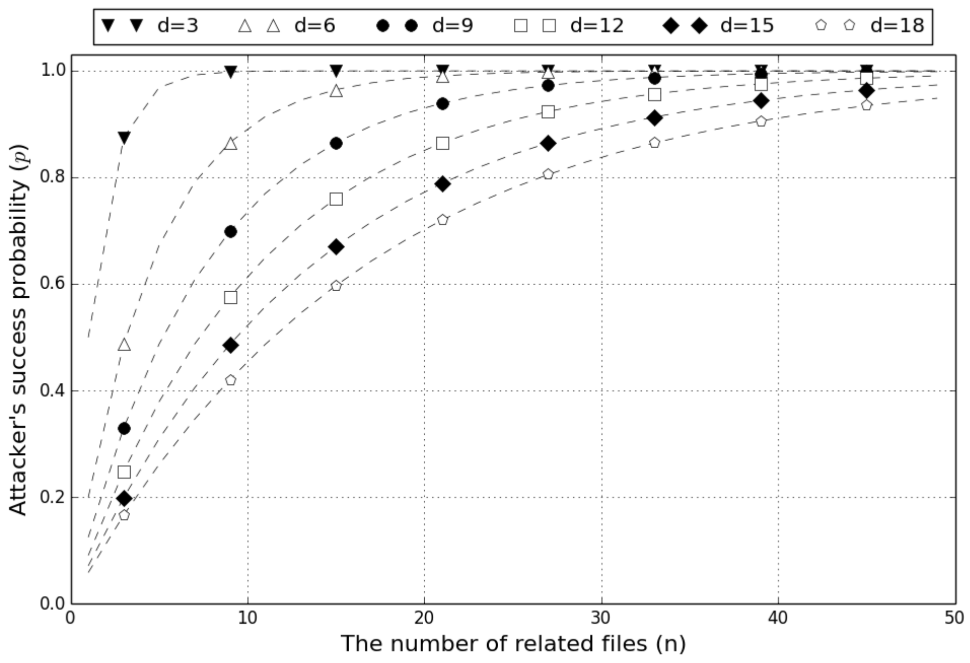


Figure 4.2: Success probability of *related-files attack* against Harnik *el al.*'s scheme

Related-files attack

The security of Harnik *et al.*'s scheme stands on an implicit assumption that all files are stored independently. This assumption is too strong, because in real environments some files are likely to be correlated to each other and thus stored at the same server together. For instance, some executable files may coexist to construct a software package. We also consider some document files with the same content in different formats, such as *doc*, *pdf*, and *xml*, coexist to support various document viewers.

Related-files attack exploits this correlations among files. By uploading not only F but also other files related to F, the adversary can infer the existence of F with high probability. For Harnik *et al.*'s scheme, the probability that the threshold of at least one is 2 or d among n related files is $p = 1 - \left(1 - \frac{1}{d-1}\right)^n$ and the probability increases as n increases. Fig. 4.2 shows the relationship between the number of related files n and the probability p that the adversary successfully gets information of F via *related-files attack* varying the security parameter d. For example, with 10 files related to F (*i.e.,* $n = 10$), the adversary can determine the existence of F with probability $p = 0.45$ (assume that $d = 18$), which is much higher than $p = 0.06$ with no related files (*i.e.,* $n = 1$).

4.2.2 *Related-files attack* on Lee *et al.*'s scheme

Lee *et al.* [45] observed that the probability of Harnik *et al.*'s scheme that information of the existence of a file is leaked to the adversary is not negligible in security parameter d, and

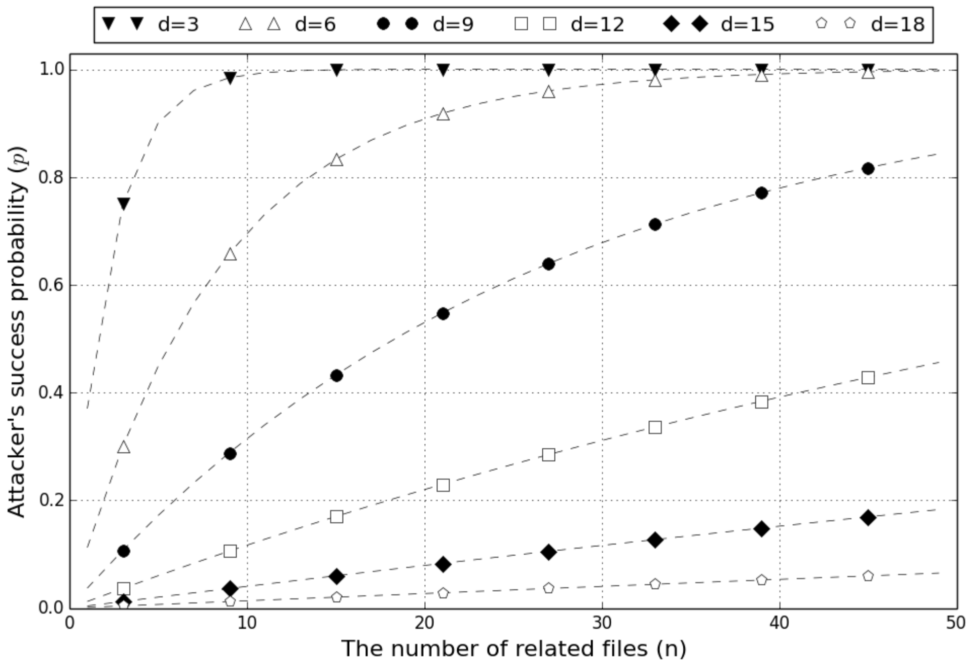


Figure 4.3: Success probability of *related-files attack* against Lee *el al.*'s scheme

proposed a security-enhanced version of Harnik *et al.*'s solution.

The protocol description

In Lee *et al.*'s scheme, a threshold t_F is initially set to d for every file F, where d is a security parameter. For each upload of a copy of F, the server randomly chooses $r \in \{0, 1, 2\}$ and calculates $t_F = t_F - r$ instantly. Then, as similar to Harnik *et al.*'s scheme, deduplication occurs at client side if $c_F \geq t_F$ or the copy of F is uploaded by the identical user; otherwise, deduplication is performed at sever-side. Fig. 4.1(b) illustrates this protocol.

Let us denote the counters of the number of events, $r = 0$, $r = 1$ and $r = 2$, to w, y and z, respectively. Then, the counter can be represented as $c_F = w + y + z$. Client-side deduplication occurs if

$$\begin{aligned} c_F &\geq d - \{(0 \cdot w) + (1 \cdot y) + (2 \cdot z)\} \\ &\geq d - \{y + (2 \cdot z)\}. \end{aligned}$$

The above equation can be simplified as follows:

$$w + (2 \cdot y) + (3 \cdot z) \geq d.$$

For analyzing the security, let us consider the following two cases.

- P: The adversary finds out that a deduplication event occurs after uploading $\lceil \frac{d}{3} \rceil$ copies of F. The probability of this event depends on the characteristic of d as follows:

  i) If d is a multiple of 3 (*i.e.,* $d = 3k$, where k is a positive integer). Then, $\Pr\{\mathsf{P}\} = \left(\frac{1}{3}\right)^k$.

  ii) If d is not a multiple of 3 (*i.e.,* $d = 3k+1$ or $d = 3k'+2$). Then,

$$\Pr\{\mathsf{P}\} = (2k+3)\left(\frac{1}{3}\right)^{k+1} + (k'+2)\left(\frac{1}{3}\right)^{k'+1}.$$

- Q: The adversary finds out that a deduplication event occurs after uploading d copies of F. This is the case that $w = d$ and $y = z = 0$. Thus, the probability is $\Pr\{\mathsf{Q}\} = \left(\frac{1}{3}\right)^d$.

For better security, we select d as a multiple of 3 ($i.e.$,$d = 3k, k > 1$). Then, the probability that the adversary learns information of the existence of F is

$$\Pr\{\mathsf{P} \text{ or } \mathsf{Q}\} = \left(\frac{1}{3}\right)^k + \left(\frac{1}{3}\right)^d = \left(\frac{1}{3}\right)^{d/3} + \left(\frac{1}{3}\right)^d,$$

which is negligible in the security parameter d.

Related-files attack

The security of Lee *et al.*'s scheme [45], however, stands on the same assumption as Harnik *et al.*'s method that files are not related to each other and stored independently. The probability that for at least one out of n related files, the event P or Q occurs is $p = 1 - \left\{1 - \left(\frac{1}{3}\right)^{d/3} - \left(\frac{1}{3}\right)^d\right\}^n$, which increases as the number of related files increases. Fig. 4.3 shows the relationship between the number of related files n and the probability p varying the security parameter (d). As shown through the graph in Fig. 4.3, Lee *et al.*'s solution is still vulnerable to *related-files attack*.

4.3 The Proposed Protocol

4.3.1 Background

The notion of differential privacy was introduced by Dwork *et al.*[54][55][56]. It is a standard privacy notion for release functions over sets of data items (or databases). Differential privacy is defined for randomized functions (*i.e.,* distributions over the set of possible outputs) and can be formally stated as follows.

Definition 3. *A randomized function $Q : D \to R$ satisfies ϵ-differential privacy if for all $r \in R$ and for all data sets D and D' that differ in a single item*

$$\Pr\{Q(D) = r\} \leq e^{\epsilon} \cdot \Pr\{Q(D') = r\}.$$

The definition states that the probability of any result of the randomized function Q is almost independent of whether any individual item among the data set is present in the input. For each item, it is almost as if the item was not used in the computation of the function, a very strong baseline for privacy. The privacy guarantee assumes that each data item is independent of the rest and applies to all aspects of the item.

Differential privacy is actually preserved by adding noise to the output of the function. Intuitively, the noise introduces uncertainty about the real value of the output, which naturally makes the real values of the inputs also uncertain to an adversary. The basic method for implementing differential privacy mechanism is to use Laplacian noise, which is a symmetric exponential distribution [54]. The mechanism is additive, *i.e.,* given a function it approximates its output by computing the function exactly and then adding noise sampled from a specific distribution.

Comparing with alternative privacy definitions, differential privacy gives strong security guarantee. Unlike the notion of differential privacy, many other privacy notions do not provide a direct guarantee or are even vulnerable to auxiliary information that an adversary might know. For example, the notion of k-anonymity [57], which provides security on releasing data such that the identity of individual data items remains private, is weaker than the notion of differential privacy. That is, k-anonymity is provided if the information for each item cannot be distinguished from at least k-1 other items. However, this definition gives no guarantee if the adversary has knowledge of some auxiliary information about the data sets [58].

4.3.2 System and Attack Model

We consider a general cloud storage service model that involves the following three entities (see Fig. 4.4).

- Cloud storage server (CSS): This is owned by a cloud service provider and provides many storage resources as a service to its users through WAN (or the Internet).
- Storage gateway (GW): This is a local disk attached network server that resides at the customer's on-premises site and provides an interface to access data at CSS. Every user request for a file upload or download is processed by GW, which in turn performs data deduplication and transfers the actual user data to CSS.
- Users (U): A user outsources its data to CSS through GW. Data deduplication becomes transparent to U since the process is performed on GW.

An adversary $\mathcal{A}$, acting as a user by creating its own account or compromising the accounts of others, interacts with GW by uploading or downloading some data. During the interactions, $\mathcal{A}$ could view all network traffic and its content between $\mathcal{A}$ (including compromised users) and GW. We assume that a connection from GW's WAN interface to CSS is encrypted by SSL (Secure

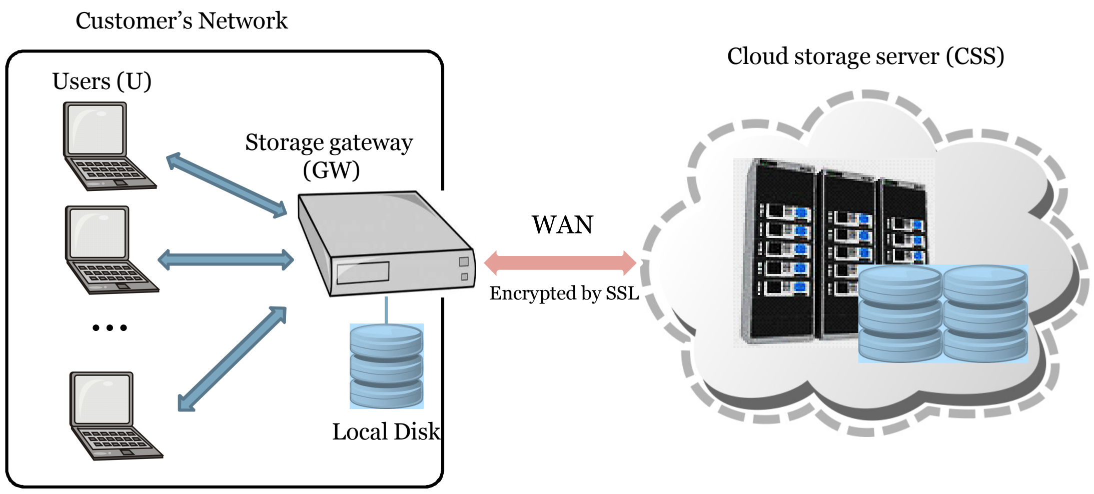


Figure 4.4: The system model of the proposed protocol

Socket Layer) or TLS (Transport Layer Security) protocol. As recent results [59][60] show that SSL/TLS keeps the content of communications securely from being sniffed by adversaries unless some flaws in its implementation exist, $\mathcal{A}$ can monitor only the amount of traffic at the link between GW and CSS, but not its content. The goals of $\mathcal{A}$ are (1) to determine the existence of a file of interest in CSS, and eventually (2) to learn the content of the file. $\mathcal{A}$ may have auxiliary information that several files coexist with the file of interest. Monitoring the network traffic, $\mathcal{A}$ will mount a sophisticated traffic analysis attack with powerful computing resources to achieve its goals.

4.3.3 Design Goal

According to the attack model described above, $\mathcal{A}$ views a deduplication as an oracle and will use it when mounting the attack. That is, $\mathcal{A}$ continues file uploading and observing the amount of network traffic that eventually occurs during the attack. Thus, we can model the deduplication as a function $Q_S(F)$, which on storage S takes a file F as input and outputs the amount of network traffic.

Definition 4. *We say that storage S is a set of files in CSS. A randomized function $Q_S(F)$ gives ϵ-differential privacy if for all storages S_1 and S_2 that differ in a single file F (i.e., $S_2 = S_1 \bigcup \{F\}$) and all $0 \le \tau \le |F|$,*

$$\Pr\{Q_{S_2}(F) = \tau\} \le e^{\epsilon} \cdot \Pr\{Q_{S_1}(F) = \tau\} \ .$$

The proposed protocol is expected to achieve the following security and performance goals.

- **Single-file privacy (ϵ-differential privacy)**: The proposed protocol should give ϵ-differential privacy so that $\mathcal{A}$ cannot distinguish between two storages S_1 and S_2, where $S_2 = S_1 \bigcup \{F\}$. That is, the proposed protocol should prevent $\mathcal{A}$ from using deduplication as a side channel.

- **Related-files privacy**: The proposed protocol should prevent $\mathcal{A}$ from obtaining any information even when knowing some related files $\{F_1, F_2, ..., F_n\}$ for any $n > 1$. That is, $\mathcal{A}$ should not gain a higher advantage in distinguishing S_1 from $S_2 = S_1 \bigcup \{F_1, F_2, ..., F_n\}$ than distinguishing it from $S_2 = S_1 \bigcup \{F\}$.

- **Efficiency**: Network traffic overhead that is caused by the proposed protocol should be minimized. We also require that the proposed protocol should not significantly degrade the system performance on GW.

4.3.4 Protocol Description

The proposed deduplication protocol is implemented on GW. The main idea is that: (1) For hiding the deduplication from $\mathcal{A}$, GW runs a differentially private algorithm when deciding the volume of data to be sent to CSS; (2) for minimizing the network overhead, GW maintains a queue, denoted as T, in which data accepted from U is temporarily stored before transmission, and if needed, GW uses data from T as noise traffic instead of generating dummy traffic. The proposed protocol is comprised of two operations: **File Upload** and **File Download**.

File Upload operation

U uploads a file F to GW. During this uploading, deduplication is not run and all bytes of F are always sent to GW. Once F accepted, GW runs Algorithm 1 to perform deduplication and transfer the data to CSS. Let us denote by $\{b_1, b_2, ..., b_{|F|}\}$ a sequence of bytes of F and by $\{c_1, c_2, ..., c_{|T|}\}$ a sequence of bytes of data stored in T. The details of Algorithm 1 are as follows.

1. Put $F = \{b_1, b_2, ..., b_{|F|}\}$ into the local disk of GW.

2. Communicate with CSS to check if F has already been stored in CSS. In detail, CHECKFILEEXISTS() algorithm calculates $h(F)$, a hash value of F, and sends $h(F)$ to CSS. Then, it receives an answer *FileExists* $\in \{\textbf{true}, \textbf{false}\}$ from CSS.

3. Compute τ ($0 \leq \tau \leq |F|$), the amount of data to be actually transferred to CSS. In order to determine τ, GETTRANSFERSIZE() (Algorithm 2) uses $Poi(\lambda)$, which generates a *Poisson*-distributed random number from a finite interval $[0, 1]$ with a mean λ.

4. If F does not exist in CSS (*i.e.,* a deduplication event does not occur), all bytes of F should be transferred to CSS. A subset of bytes $\{b_1, b_2, ..., b_\tau\}$ of F is moved to *Buffer*

Algorithm 1

```
procedure UPLOAD($F$)
    $Buffer \leftarrow \emptyset$
    $FileExists \leftarrow$ CHECKFILEEXISTS($F$)
    $\tau \leftarrow$ GETTRANSFERSIZE($|F|$, $FileExists$)
    if $FileExists$ is false then
        $Buffer \leftarrow \{b_1, b_2, ..., b_\tau\}$
        ENQUEUE($T, \{b_{\tau+1}, b_{\tau+2}, ..., b_{|F|}\}$)        ▷ Put $|F| - \tau$ bytes of $F$ into $T$
    else
        $\{c_1, c_2, ..., c_t\} \leftarrow$ DEQUEUE($T, \tau$)        ▷ Get $\tau$ bytes of data from $T$
        $Buffer \leftarrow \{c_1, c_2, ..., c_t\}$
        if $t < \tau$ then
            $DummyBytes \xleftarrow{R} \{0,1\}^{(|\tau|-|t|)\times 8}$
            $Buffer \leftarrow Buffer \,||\, DummyBytes$
        end if
    end if
    TRANSFER($Buffer$)        ▷ Send data in $Buffer$ to CSS
end procedure
```

from the local storage for transmission. The remaining bytes $\{b_{\tau+1}, b_{\tau+2}, ..., b_{|F|}\}$ are then enqueued into T by running ENQUEUE() algorithm.

5. If F already exists in CSS (*i.e.,* a deduplication event occurs), it is not necessary to transfer F to CSS. Hence, τ bytes of data $\{c_1, c_2, ..., c_t\}$ are dequeued from T by running DEQUEUE() and then put onto *Buffer* for transmission. If there are not enough data in T (*i.e.,* $t < \tau$), dummy bytes are randomly generated and padded to *Buffer* so that the total size becomes equal to τ.

6. Finally, the data in *Buffer* are sent to CSS by running TRANSFER().

File Download operation

U sends GW a request for downloading F. Upon receiving, GW retrieves F in the storage of CSS. If found, GW returns the requested file F to U.

Algorithm 2

```
1: procedure GetTransferSize($|F|$, $FileExists$)
2:     if $FileExists$ is true then
3:         $\alpha \xleftarrow{R} Poi\left(\frac{1+\epsilon}{2}\right)$        ▷ $\epsilon$ is a privacy parameter ($0 \le \epsilon \le 1$)
4:     else
5:         $\alpha \xleftarrow{R} Poi\left(\frac{1-\epsilon}{2}\right)$
6:     end if
7:     $\tau \leftarrow \lceil \alpha |F| \rceil$        ▷ $|F|$ is the size of $F$ in bytes and $0 \le \alpha \le 1$
8:     return $\tau$
9: end procedure
```

4.4 Evaluation

4.4.1 Security Analysis

We analyze the security of the proposed protocol in terms of two requested security properties. For analysis, the proposed protocol is modeled as a randomized function $Q_S(F)$ in this section. Suppose that an adversary algorithm $\mathcal{A}$ tries to learn information on the file of interest in CSS.

Single-file privacy

$\mathcal{A}$ gets no information on the occurrence of deduplication by monitoring the link between $\mathcal{A}$ and GW since all bytes of the file are always sent to GW at the link. At WAN link of GW, the content of the network is not visible to $\mathcal{A}$ due to encryption, thus $\mathcal{A}$ has no option but to conduct traffic analysis by querying $Q_S(F)$. The output of $Q_S(F)$, denoted by τ in the previous section, is actually determined by Algorithm 2. By the definition of *Poisson* distribution, $Pr\{Poi(\lambda) = x\} = \lambda^x e^{-x}/x!$. It can be shown that the proposed protocol satisfies the security that is described in **Definition 4** and hence gives single-file privacy.

$$\frac{\Pr\{Q_{S_2}(F) = \tau\}}{\Pr\{Q_{S_1}(F) = \tau\}} = \frac{\Pr\left\{Poi\left(\frac{1-\epsilon}{2}\right) = \alpha\right\}}{\Pr\left\{Poi\left(\frac{1+\epsilon}{2}\right) = \alpha\right\}} = \frac{\left(\frac{1-\epsilon}{2}\right)^{\alpha} e^{\frac{-1+\epsilon}{2}}}{\left(\frac{1+\epsilon}{2}\right)^{\alpha} e^{\frac{-1-\epsilon}{2}}}$$
$$= \left(\frac{1-\epsilon}{1+\epsilon}\right)^{\alpha} e^{\epsilon} \le e^{\epsilon}.$$

Single-file privacy means that for any two storages that differ on a single file, the proposed protocol will release approximately the same volume of network transmission on both storages. This property guarantees that the presence or absence of an individual file will not affect the output of the proposed protocol significantly, and thus, prevents the adversary $\mathcal{A}$ from using deduplication as a side channel.

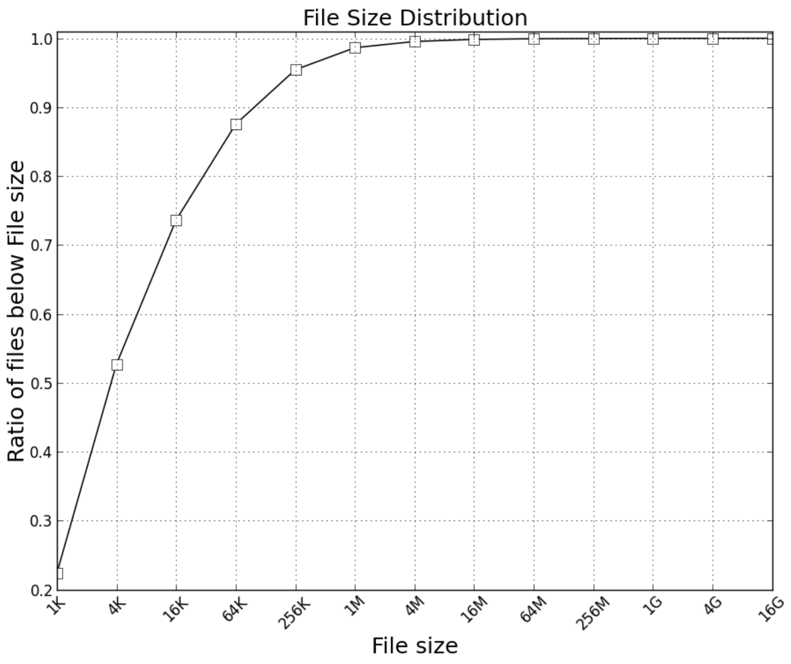


Figure 4.5: Cumulative file size distribution in the data set

Related-files privacy

Let us denote three storages: S_1, $S_2 = S_1 \bigcup \{F\}$, and $S_3 = S_1 \bigcup \{F_1, F_2, ..., F_n\}$, where $F \in \{F_1, F_2, ..., F_n\}$. It is obvious that the computation of $Q_S(F)$ is independent of the rest of the files, and $\Pr\{Q_{S_2}(F) = \tau\} = \Pr\{Q_{S_3}(F) = \tau\}$. The ratio of probabilities is therefore

$$\frac{\Pr\{Q_{S_2}(F) = \tau\}}{\Pr\{Q_{S_1}(F) = \tau\}} = \frac{\Pr\{Q_{S_3}(F) = \tau\}}{\Pr\{Q_{S_1}(F) = \tau\}}.$$

This result implies that the statistical difference of $Q_S(F)$ between S_1 and S_3 is always equal to the difference between S_1 and S_2. Hence, the advantage that $\mathcal{A}$ succeeds in distinguishing between S_1 and S_3 (*i.e.,* with knowledge of related files $\{F_1, F_2, ..., F_n\}$) is not greater than the probability of distinguishing between storages S_1 and S_2, which differ in a single file.

4.4.2 Experiments

We conducted experiments for evaluating the effectiveness and efficiency of the proposed protocol. Our test bed consisted of two servers, which acted as GW and CSS, with an Intel Core processor i5 running at 2.8 GHz and 4 GB RAM memory with a 2 TB hard disk. One server acting as the GW is located at local area network and connected with a client PC. The other server is located remotely over the Internet and acting as the CSS. The data set used in our experiments consists of files including Windows system files, office documents, and media files, totaling up to 3 TBs. Fig. 4.5 shows cumulative file size distribution of the data set.

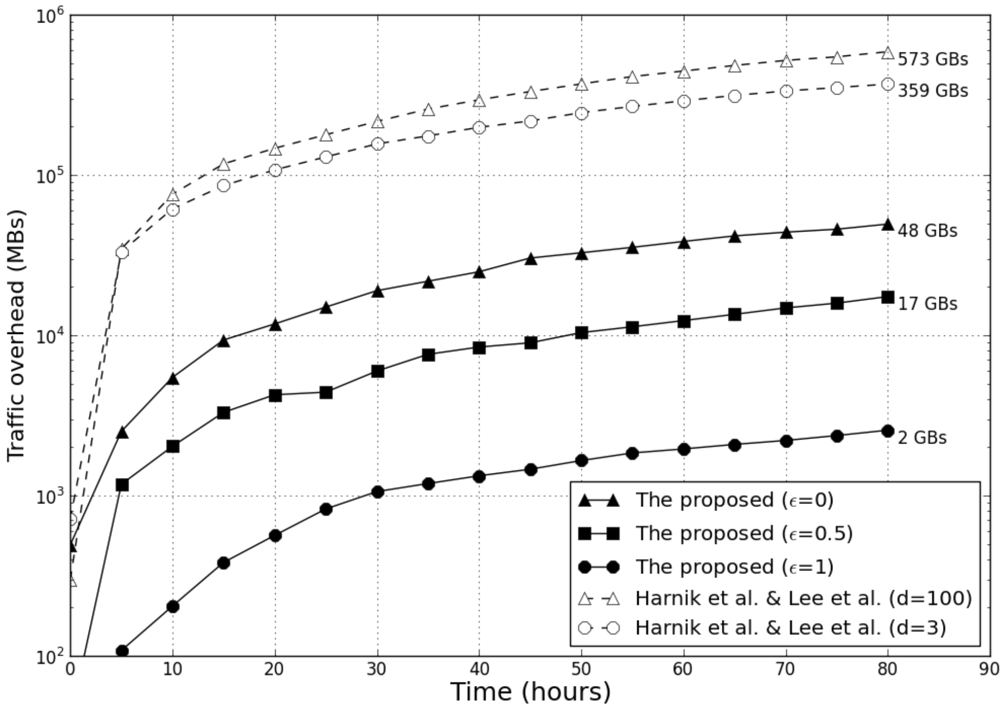


Figure 4.6: Network transmission overhead over time

We split the data set into two groups of equal size and put one group into CSS and used the other group for uploading at client side during the experiment. Uploading behavior by users was simulated such that the upload event followed exponential distribution with its frequency varying from 10 to 600 uploads per 10 min.

Network efficiency

We first evaluated the network efficiency by measuring traffic overhead and comparing the measurements with the previous randomized schemes of Harnik *et al.* and Lee *el al.*. For the experiment, we simulated the previous schemes with all network traffic passing through GW. In the proposed and the previous schemes, traffic overhead is measured at the outgoing (WAN) interface of GW to CSS. Fig. 4.6 shows the amount of accumulated traffic overhead over time with varying security parameters. "Traffic overhead" in Fig. 4.6 refers to the amount of extra bandwidth consumption used to obfuscate the occurrence of deduplication. In the proposed protocol, dummy traffic is generated and transferred to CSS only if there are not enough data in T. Hence, the extra bandwidth consumption is quite small as compared to that of the previous schemes, in which copies of duplicate data are always transferred until it reaches the threshold. With greatest security guarantee ($\epsilon = 0$), the proposed protocol incurs 48 GBs in total, which is about 13% of the traffic overhead incurred by the previous schemes with the lowest security guarantee ($d = 3$).

System performance

We also evaluated the system performance on GW by measuring two metrics: required local disk capacity and average processing time for each request. Both metrics are sufficient to measure the required system resources on GW for running the proposed protocol. For comparison, we considered a batch algorithm that is similar to the proposed protocol but performs in a naive way. The batch algorithm immediately accepts user files and performs deduplication but transfers them to CSS periodically every given batch interval. For ease of comparison, we ran the proposed protocol such that at every batch interval all remaining data in T were uploaded to CSS. Fig. 4.7 and 4.8 shows two measurements under different batch intervals 60 and 120 min, respectively with varying upload frequencies. In Fig. 4.7, "Max storage" refers to the greatest amount of data stored in the local disk on GW during the experiment, which indicates the disk capacity required to run these schemes. In Fig. 4.8, "Avg. processing time" refers to average time taken for each upload request to complete its upload to CSS. The results of this experiment indicate that the proposed protocol uses a decreasing amount of system resources on GW as the number of users (*i.e.,* upload frequency) increases as compared to the batch algorithm.

4.5 Summary

In order to achieve cost savings of network bandwidth and disk storage, cloud storage service providers apply client-side (or source-based) data deduplication techniques. However, deduplication can be used as a side channel by adversaries who try to obtain sensitive information of other users' data. Several solutions that have been proposed to prevent such information leakage are based on a strong assumption that all individual files (or data) are independent of each other.

Observing this assumption, we proposed new attack, which is called *related-files attack*, to the previous approaches. In order to mitigate such an attack, we proposed a storage gateway based secure client-side deduplication protocol. A storage gateway is a network appliance that provides access to the remote cloud server and simplifies interactions with cloud storage services, and is used in various cloud service delivery models such as public, private and hybrid cloud computing. The proposed solution, by utilizing the storage gateway as an important component in the system design, achieves greater network efficiency and architectural flexibility while also reducing the risk of information leakage.

For more robust security guarantee, we applied a differential private mechanism to the proposed protocol. Differential privacy is a security notion that has been used for designing privacy enhanced databases. By exploiting a differential private mechanism, the proposed protocol strongly guarantees that the presence or absence of individual files in the cloud storage is hard to infer from any side channel attacks including *related-files attack* that has been proposed in the chapter.

For validation of the effectiveness and efficiency of the proposed protocol, we conducted several experiments using real data sets. The network efficiency and the system performance of the proposed protocol were evaluated by measuring traffic overhead, required local disk capacity and average processing time for each request. The experiments showed that the proposed protocol outperforms the previous approaches.

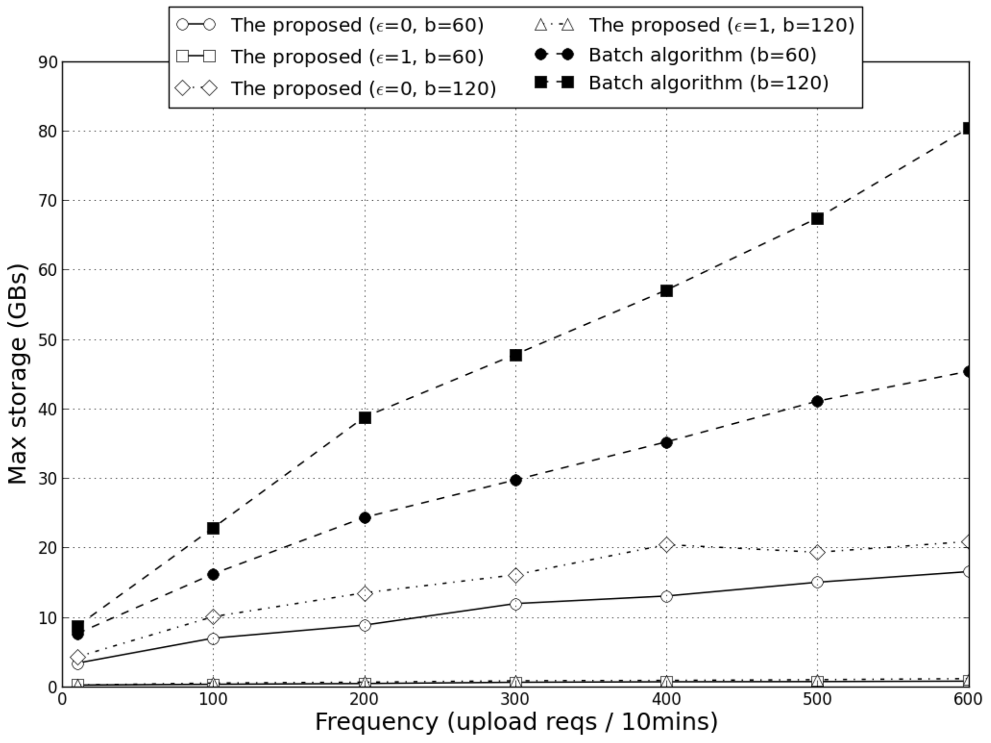


Figure 4.7: Storage overhead (b refers to batch interval in minutes)

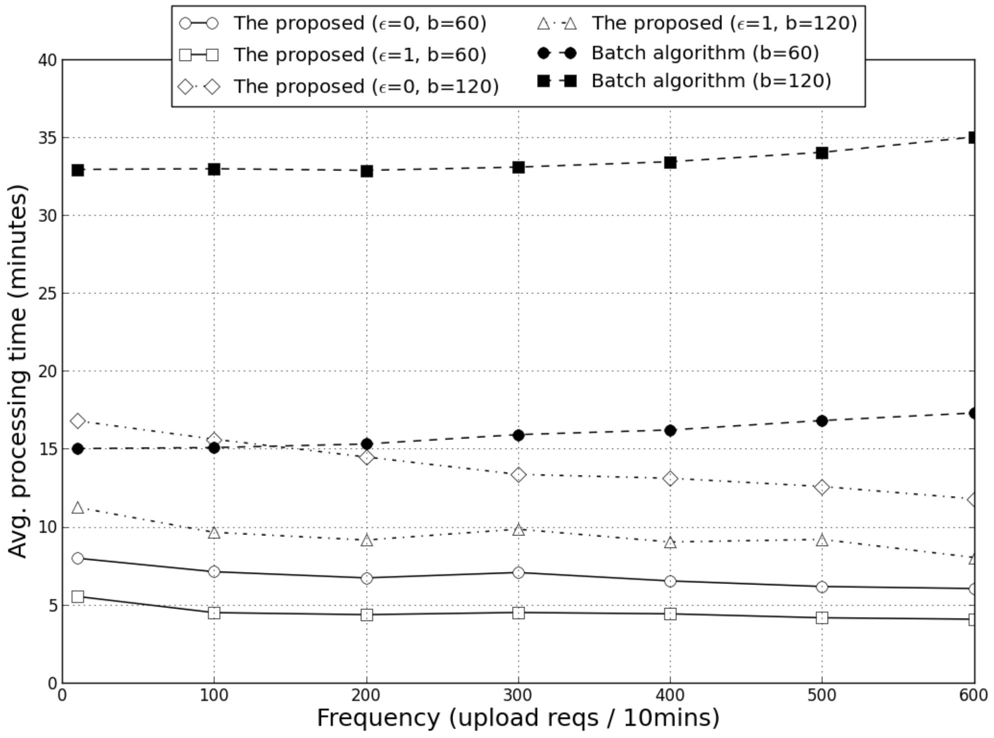


Figure 4.8: Average processing time (b refers to batch interval in minutes)

Chapter 5. Security-enhanced Proof of Storage with Deduplication

5.1 Motivation and Contributions

In terms of cloud storage security, there have been three important security properties:

- Provable Data Possession (PDP): This property allows a cloud storage client to verify the integrity of its outsourced data in more efficient way than the straightforward solution that downloads the whole data for verification. The notion of PDP was introduced by [25] and has been addressed in various ways [30][31][32].
- Proof of Retrievability (POR): This property is a compact proof by a cloud storage server to a client that a outsourced data is intact, that the client can fully recover it. That is, it ensures that the cloud storage client can actually recover the outsourced data in the cloud server. Compared to PDP, this property uses Error Correction/Erasure Codes to tolerate the damage to a part of the outsourced data. The notion of POR, which was introduced by [24], has been addressed in various ways [26][27][28][29].
- Proof of Ownership (POW): This property lets a cloud storage client efficiently prove to a cloud server that the client actually holds a data (or file), rather than just short information about it, such as a hash value. The notion of POW was introduced by [20] and has been extended in various ways [48][49][50][61].

The notions of PDP and POR have been introduced for detecting the corruption of outsourced data in the cloud storage, while the notion of POW is for enabling the storage server to use data deduplication techniques more securely. These two aspects seem to be to the opposite of each other. However, it is known that public verifiability that offered by PDP/POR can be used to design a POW scheme [62].

Based on the insight of exploiting public verifiability, a single scheme that couples the notions of PDP/POR and POW can be built. Proof of Storage with Deduplication (POSD), which is proposed by Zheng *et al.* [62], is the first scheme that achieves these three security properties together. The POSD scheme makes not only a client to be assured the integrity of its outsourced data, but also a storage server to take advantage of deduplication techniques in a secure manner.

In the POSD scheme, the verification of auditing and deduplication protocol entirely depend on public keys, which are created and provided by cloud storage clients. Hence, the validity of the scheme is implicitly based on an assumption, which is called random key assumption in the rest of this chapter, that all clients are honest in terms of generating their keys. In the cross-multiple users and the cross-domain environment of the cloud computing, however, such an assumption is unrealistic. Eliminating the random key assumption may cause cloud storage systems that utilize the POSD scheme to face a new security threat that has not been considered before. Unfortunately, the scheme has a serious security breach under new attack model in which malicious clients are allowed to dishonestly manipulate their keys.

In this chapter, the security weakness of the POSD scheme is presented. More specifically, it is shown that the scheme fails to satisfy two security requirements, server unforgeability and (κ, θ)-uncheatability, under new attack model that is very reasonable and effective. A countermeasure against this attack is provided by modifying the scheme such that the client-created keys are blended with the random values contributed by the storage server. The proposed solution actually weakens the client's capability to control their keys. The modification is minimized so that our scheme preserves the efficiency while providing more robust security.

This chapter is organized as follows: In Section 5.2, the POSD scheme is briefly reviewed. In Section 5.3, new attack model and some attack scenarios are presented. In Section 5.4, the proposed countermeasure against the attack is described. Finally, in Section 5.5, this chapter is summarized.

5.2 Proof of Storage with Deduplication

In this section, a brief review of the POSD [62] scheme including the cryptographic background, system model and security requirements are presented.

5.2.1 Preliminaries

Computational Diffie-Hellman (CDH) assumption

Let $g, g^w, h \in \mathbb{G}$ be chosen uniformly at random. The CDH problem is to compute h^w. The CDH assumption states that no PPT algorithm can solve the CDH problem with non-negligible advantage.

Discrete Log (DLOG) assumption

The DLOG problem is to find w such that $g^w = h$, given any two random elements $g, h \in \mathbb{G}$. The DLOG assumption states that no PPT algorithm can solve the DLOG problem with non-negligible advantage. Note that the DLOG assumption is weaker than the CDH assumption.

5.2.2 System Model

In the POSD scheme, we consider a system that consists of the following three participants.

- Cloud storage server: It provides abundant storage resources to the cloud storage clients. The server also has relevant assurance procedures, by which the clients can verify the integrity of their outsourced data. For saving the storage space, the cloud storage server can utilize data deduplication techniques.

- Cloud storage clients: They outsource their own data to the cloud storage server via secure channel. The clients want their data to be stored securely, while allowing the server to perform data deduplication operation.

- Auditor: It is a third party which is allowed to check the integrity of the client's outsourced data. Any client in possession of an outsourced data may act as an auditor of that specific data.

5.2.3 Definition and Security requirements

Built on the definitions of PDP/POR[25][24] and POW[20], the POSD scheme is defined as follows:

Definition 5. *A* POSD *scheme consists of the following tuple of probabilistic polynomial algorithms* (**KEYGEN**, **UPLOAD**, **AUDITINT**, **DEDUP**).

KEYGEN: It takes a security parameter λ as an input, and outputs two pair of public/private keys (PK_{int}, SK_{int}) and (PK_{dup}, SK_{dup}), where PK_{int} is known to public and SK_{int} is the corresponding private key of a client, PK_{dup} is made public and SK_{dup} is the server's corresponding private key.

UPLOAD: This is a data (or file) uploading protocol running by a cloud storage server and a client. For uploading a new data file F to the cloud storage server, the client takes as inputs a unique file identifier fid as well as the secret key SK_{int}, and generates Tag_{int}, which can be used to audit the integrity of F later. Upon receiving a tuple (fid, F, Tag_{int}) from the client, the server calculates Tag_{dup} using SK_{dup} and stores them in the storage.

AUDITINT: This is the auditing protocol for checking the data integrity, which is run by a cloud storage server and an auditor. The protocol can be written formally as follows:

$$b \leftarrow (\mathrm{AUDITOR}(\mathsf{fid}, \mathrm{PK}_{int}) \Longleftrightarrow \mathrm{S}(\mathsf{fid}, \mathsf{F}, \mathrm{Tag}_{int}))$$

, where $b \in \{0, 1\}$ and S denotes the cloud storage server. The auditor takes the file identifier fid and the corresponding public key PK_{int} as inputs. Also, the server takes the file F corresponding to fid and the file's Tag_{int} as inputs. At the beginning of the protocol, the auditor sends a challenge to the server and the server computes and sends back a response. If the response is valid with respect to the challenge, the auditor outputs 1, and 0 otherwise.

DEDUP: This is the data deduplication protocol running by a cloud storage server and a client. The protocol can be written formally as follows:

$$b \leftarrow (\mathrm{S}(\mathsf{fid}, \mathrm{Tag}_{dup}, \mathrm{SK}_{dup}, \mathrm{PK}_{dup}) \Longleftrightarrow \mathrm{C}(\mathsf{fid}, \mathsf{F}))$$

, where C denotes a cloud storage client, who is about to outsource a data file F. The server sends a challenge to the client, which returns a response value that is generated using F and its relevant information. The server verifies the response value using Tag_{dup}, PK_{dup} and SK_{dup}, and then outputs 1 if it is successful, and 0 otherwise.

The POSD scheme is required to satisfy the two security properties: server unforgeability, (κ, θ)-uncheatability.

Server unforgeability : No cheating server can fool an honest client (or an auditor), who is to verify the outsourced file F, by presenting $\mathsf{F}' \neq \mathsf{F}$ with non-negligible probability. This security property can be defined formally as follows:

Definition 6. *Let us consider the following game between an adversary $\mathcal{A}$ and a challenger. In the game, $\mathcal{A}$ plays the role of the cloud server, and may control many compromised clients. The challenger acts as an honest client during the game. The* POSD *scheme is server unforgeable if the winning probability for any polynomial algorithm $\mathcal{A}$ is negligible in λ.*

Setup phase: **KEYGEN** algorithm is executed to generate (PK_{int},SK_{int}) and (PK_{dup},SK_{dup}). PK_{int} and PK_{dup} are known to public, while SK_{int} and SK_{dup} are given to the challenger and $\mathcal{A}$, respectively. Note that the challenger keeps the private key SK_{int} secret from others including $\mathcal{A}$.

Challenge phase: At this phase, $\mathcal{A}$ does the following.

- $\mathcal{A}$ adaptively chooses a file $\mathsf{F} \in \{0,1\}^*$ and runs the **UPLOAD** protocol with the challenger. The process may be repeated for polynomial times. At the end of the execution, $\mathcal{A}$ obtains (fid, F, Tag_{int}) and the challenger keeps a record of $\mathcal{Q} = \{\mathsf{fid}\}$.
- $\mathcal{A}$ may execute **AUDITINT** protocol with the challenger with respect to any $\mathsf{fid} \in \mathcal{Q}$ chosen by the challenger, and execute **DEDUP** protocol with respect to some file.

Forgery phase: Finally, $\mathcal{A}$ outputs an $\mathsf{fid} \in \mathcal{Q}$ corresponding to F. The adversary $\mathcal{A}$ wins the above game if for any $\mathsf{F}' \neq \mathsf{F}$,

$$1 \leftarrow (\mathrm{AUDITOR}(\mathsf{fid}, \mathrm{PK}_{int}) \Longleftrightarrow \mathcal{A}(\mathsf{fid}, \mathsf{F}', \cdot)).$$

(κ,θ)-uncheatability : Given a file F with min-entropy κ, no cheating client, who gets up to θ-bit entropy of F, can fool the server with non-negligible probability. This security property can be defined formally as follows:

Definition 7. *Let us consider the following game between an adversary $\mathcal{A}$ and a challenger, where $\mathcal{A}$ plays the role of (compromised) clients, and the challenger acts as the server and an honest client. The* POSD *scheme is (κ,θ)-uncheatable if the winning probability for any polynomial algorithm $\mathcal{A}$ is negligible in λ.*

Setup phase: **KEYGEN** algorithm is executed to generate (PK_{int},SK_{int}) and (PK_{dup},SK_{dup}). PK_{int} and PK_{dup} are known to public. For a compromised client, the corresponding SK_{int} is given to $\mathcal{A}$, while both SK_{dup} and SK_{int} corresponding to the honest client (the challenger) are kept secret from $\mathcal{A}$. Then, the challenger chooses a file F of κ-bit min entropy and its unique fid. The challenger honestly executes the **UPLOAD** protocol by acting the roles of both the client and the server. During the process, $\mathcal{A}$ can observe and obtain the public information.

Challenge phase: In order to inter the information of F, $\mathcal{A}$ runs the **UPLOAD, AUDITINT** and **DEDUP** protocols with the challenger, while also utilizing computing resources of compromised clients. At the end of this phase, $\mathcal{A}$ may learn up to θ-bit entropy of F.

Forgery phase: $\mathcal{A}$ outputs some F'. The adversary $\mathcal{A}$ wins the game if

$$1 \leftarrow (\mathrm{S}(\mathsf{fid}, \mathrm{Tag}_{dup}, \mathrm{SK}_{dup}, \mathrm{PK}_{dup}) \Longleftrightarrow \mathcal{A}(\mathsf{fid}, \mathsf{F}')).$$

5.2.4 The Scheme Construction

Let p, q be two sufficiently large primes and $\mathbb{G}$, $\mathbb{G}_T$ be cyclic groups of order q. Let $g \in \mathbb{G}$ be a generator of $\mathbb{G}$ and $e : \mathbb{G} \to \mathbb{G}_T$ be an admissible bilinear map. Let F be a data file consisting of n blocks and each block F_i ($1 \leq i \leq n$) consist of m symbols in $\mathbb{Z}_q$. Let us denote each symbol of F_i as F_{ij} for $1 \leq j \leq m$. Let fid be a unique file id, and let $H_1 : \{0,1\}^* \to \mathbb{G}$ and $H_2 : \{0,1\}^* \to \mathbb{Z}_q$ be hash functions. The POSD scheme consists of the following probabilistic and polynomial algorithms and protocols:

KEYGEN: This algorithm generates two pairs of public key and private key. A client runs this protocol as follows:

1. Choose v_1 and v_2 randomly from $\mathbb{Z}_p^*$ such that the orders of subgroups generated by v_1 and v_2 are q. Choose s_{j1} and s_{j2} randomly from $\mathbb{Z}_q^*$ and set $z_j = v_1^{-s_{j1}} v_2^{-s_{j2}}$ for $1 \leq j \leq m$.
2. Choose u uniformly at random from $\mathbb{G}$ and w from $\mathbb{Z}_q^*$. Then, set $z_q = g^w$, where g is a generator of $\mathbb{G}$.

3. Set the public key PK_{int}=$\{q, p, g, u, v_1, v_2, z_1, z_2, \ldots, z_m, z_g\}$ and the private key SK_{int}= $\{(s_{11}, s_{12}), \ldots, (s_{m1}, s_{m2}), w\}$.

4. Set PK_{dup}=PK_{int} and SK_{dup}=null.

UPLOAD: A client who is to outsource the file F and the server run this protocol as follows:

1. For each block F_i $(1 \leq i \leq n)$, the client chooses r_{i1}, r_{i2} at random from $\mathbb{Z}_q^*$ and computes

$$\begin{aligned} x_i &= v_1^{r_{i1}} v_2^{r_{i2}} \bmod p, \\ y_{i1} &= r_{i1} + \sum_{j=1}^{m} \mathsf{F}_{ij} s_{j1} \bmod q, \\ y_{i2} &= r_{i2} + \sum_{j=1}^{m} \mathsf{F}_{ij} s_{j2} \bmod q, \\ t_i &= \left(H_1(\mathsf{fid} \parallel i) u^{H_2(x_i)}\right)^w \quad (\in \mathbb{G}). \end{aligned}$$

2. The client sends (fid, F, Tag_{int}) to the server, where Tag_{int}=$\{(x_i, y_{i1}, y_{i2}, t_i)_{1 \leq i \leq n}\}$.

3. The server receives (fid, F, Tag_{int}) and sets Tag_{dup}=Tag_{int}.

AUDITINT: An auditor, which can be the client itself, and the server run this protocol as follows:

1. The auditor chooses a set of c elements I=$\{\alpha_1, \alpha_2, \ldots, \alpha_c\}$, where $\alpha_i \in \mathbb{N}$, and chooses a set of coefficients $\beta = \{\beta_1, \beta_2, \ldots, \beta_c\}$, where $\beta_i \in \mathbb{Z}_q^*$. The auditor sends a challenge $\mathsf{chal} = (\mathrm{I}, \beta)$ to the server.

2. Upon receiving chal, the server computes

$$\begin{aligned} \mu_j &= \sum_{i \in \mathrm{I}} \beta_i \mathsf{F}_{ij} \bmod q \ (1 \leq j \leq m), \\ \mathrm{Y}_1 &= \sum_{i \in \mathrm{I}} \beta_i y_{i1} \bmod q, \\ \mathrm{Y}_2 &= \sum_{i \in \mathrm{I}} \beta_i y_{i2} \bmod q, \\ \mathrm{T} &= \prod_{i \in \mathrm{I}} t_i^{\beta_i} \quad (\in \mathbb{G}) \end{aligned}$$

and sends resp=$(\{\mu_j\}_{1 \leq j \leq m}, \{x_i\}_{i \in \mathrm{I}}, \mathrm{Y}_1, \mathrm{Y}_2, \mathrm{T})$ to the auditor.

3. Upon receiving resp, the auditor computes :

$$\mathrm{X} = \prod_{i\in\mathrm{I}} x_i^{\beta_i} \bmod p,$$
$$\mathrm{W} = \prod_{i\in\mathrm{I}} H_1(\mathsf{fid} \parallel i)^{\beta_i}$$

and verifies

$$\mathrm{X} \overset{?}{=} v_1^{\mathrm{Y}_1} v_2^{\mathrm{Y}_2} \prod_{j=1}^{m} z_j^{\mu_j} \bmod p \qquad (5.1)$$
$$e(\mathrm{T}, g) \overset{?}{=} e(\mathrm{W}u^{\sum_{i\in\mathrm{I}} \beta_i H_2(x_i)}, z_g) \quad (\in \mathbb{G}_T)\,. \qquad (5.2)$$

If both hold, return PASS; otherwise, return FAIL.

DEDUP: The server and the client run this protocol as follows:

1. The server generates a challenge $\mathsf{chal} = (\mathrm{I}, \beta)$, where I=$\{\alpha_1, \alpha_2, \ldots, \alpha_c\}$ and $\beta = \{\beta_1, \beta_2, \ldots, \beta_c\}$ as the **AUDITINT** protocol, and sends it to the client.

2. Upon receiving chal, the client computes

$$\mu_j = \sum_{i\in\mathrm{I}} \beta_i \mathsf{F}_{ij} \bmod q\ ,\ 1 \le j \le m$$

and sends $\mathsf{resp} = (\{\mu_j\}_{1\le i\le m})$ to the server.

3. Upon receiving resp, the server verifies

$$\mathrm{X} \overset{?}{=} v_1^{\mathrm{Y}_1} v_2^{\mathrm{Y}_2} \prod_{j=1}^{m} z_j^{\mu_j} \bmod p \qquad (5.3)$$
$$e(\mathrm{T}, g) \overset{?}{=} e(\mathrm{W}u^{\sum_{i\in\mathrm{I}} \beta_i H_2(x_i)}, z_g) \quad (\in \mathbb{G}_T)\,, \qquad (5.4)$$

where $\mathrm{Y}_1, \mathrm{Y}_2, \mathrm{W}, \mathrm{X}$ and T are computed from Tag$_{dup}$=$\{(x_i, y_{i1}, y_{i2}, t_i)_{i\le i\le n}\}$. If both hold, return PASS; otherwise, return FAIL.

5.3 Weakness of the POSD scheme

In this section, new attack model against the POSD scheme is described and some practical attack scenarios are presented.

5.3.1 Weak key Attack

With the elimination of the random key assumption, a malicious client may create the manipulated keys of his/her own dishonestly rather than executing the **KEYGEN** algorithm. Some manipulated keys, which we call weak keys, can incur a security breach of the POSD scheme. The weak keys can be constructed as follows:

1. ξ is chosen at random from $\mathbb{Z}_p^*$ such that the order of ξ is q.
2. $\psi_1, \psi_2, \ldots, \psi_m$ are chosen at random from $\mathbb{Z}_q^*$.
3. ξ and ψ_j $(1 \leq j \leq m)$ are assigned as follows:

$$\begin{aligned} &v_1 = \xi, \; v_2 = \xi^{-1}, \\ &s_{j1} = s_{j2} = \psi_j \; (1 \leq j \leq m). \end{aligned}$$

4. The rest of key components g, u, w, $z_1, z_2, \ldots, z_m$ and z_g are generated correctly according to the **KEYGEN** algorithm.
5. The weak keys are formed as:

$$\begin{aligned} &\widehat{\mathrm{PK}}_{int} = \widehat{\mathrm{PK}}_{dup} = \{q, p, g, u, v_1, v_2, z_1, z_2, \ldots, z_m, z_g\}, \\ &\widehat{\mathrm{SK}}_{int} = \{(s_{11}, s_{12}), \ldots, (s_{m1}, s_{m2}), w\}, \\ &\widehat{\mathrm{SK}}_{dup} = \mathsf{null}. \end{aligned}$$

Under new attack model that allows to exploit the weak keys, a malicious client can break two security properties, server unforgeability and (κ, θ)-uncheatability, of the POSD scheme. The details are described below.

Breaking server unforgeability

We show that under the weak key attack, an auditor will be fooled into assuring the integrity of an outsourced file F even if a storage server does not have the correct file but an arbitrary $\mathsf{F}'(\neq \mathsf{F})$. Let us denote an auxiliary audit information computed from the weak keys as $\widehat{\mathrm{Tag}}_{int}$. Suppose that an adversary $\mathcal{A}$, which acts as a client, generates a weak key $\widehat{\mathrm{PK}}_{int}$ and $\widehat{\mathrm{SK}}_{int}$. $\mathcal{A}$ begins uploading a tuple $(\mathsf{fid}, \mathsf{F}', \widehat{\mathrm{Tag}}_{int})$, where $\mathsf{F}' \neq \mathsf{F}$ and $\widehat{\mathrm{Tag}}_{int}, \mathsf{fid}$ is for F, to the storage server. An auditor who is to verify the integrity of the file F will start the **AUDITINT** protocol by sending the storage server a challenge (I, β), where $\mathrm{I}=\{\alpha_1, \alpha_2, \ldots, \alpha_c\}$ and $\beta = \{\beta_1, \beta_2, \ldots, \beta_c\}$. Upon receiving the challenge, the storage server computes the following with F' and $\widehat{\mathrm{Tag}}_{int}$:

$$\begin{aligned}
\mu'_j &= \sum_{i\in \mathrm{I}} \beta_i \mathsf{F'}_{ij} \bmod q \quad (1 \le j \le m),\\
\mathrm{Y}_1 &= \sum_{i\in \mathrm{I}} \beta_i y_{i1} \bmod q,\\
\mathrm{Y}_2 &= \sum_{i\in \mathrm{I}} \beta_i y_{i2} \bmod q,\\
\mathrm{T} &= \prod_{i\in \mathrm{I}} t_i^{\beta_i} \quad (\in \mathbb{G}),
\end{aligned}$$

and sends $\mathsf{resp} = \Big(\{\mu'_j\}_{1\le j\le m}, \{x_i\}_{i\in\mathrm{I}}, \mathrm{Y}_1, \mathrm{Y}_2, \mathrm{T}\Big)$ to the auditor. In the verification phase, Eq. (5.1) holds as follows:

$$\begin{aligned}
&v_1^{\mathrm{Y}_1} v_2^{\mathrm{Y}_2} \prod_{j=1}^{m} z_j^{\mu'_j}\\
&= v_1^{\sum_{i\in\mathrm{I}}\beta_i y_{1i}} v_2^{\sum_{i\in\mathrm{I}}\beta_i y_{12}} \prod_{j=1}^{m} \left(v_1^{-s_{j1}} v_2^{-s_{j2}}\right)^{\sum_{i\in\mathrm{I}}\beta_i \mathsf{F'}_{ij}}\\
&= v_1^{\sum_{i\in I}\beta_i y_{1i}} v_2^{\sum_{i\in\mathrm{I}}\beta_i y_{12}} \prod_{j=1}^{m} \xi^{(\psi_j-\psi_j)\sum_{i\in\mathrm{I}}\beta_i \mathsf{F'}_{ij}}\\
&= v_1^{\sum_{i\in\mathrm{I}}\beta_i\left(r_{i1}+\sum_{j=1}^{m}\mathsf{F}_{ij}S_{j1}\right)} v_2^{\sum_{i\in\mathrm{I}}\beta_i\left(r_{i2}+\sum_{j=1}^{m}\mathsf{F}_{ij}S_{j2}\right)}\\
&= \left(v_1^{\sum_{i\in\mathrm{I}}\beta_i r_{i1}} v_2^{\sum_{i\in\mathrm{I}}\beta_i r_{i2}}\right) \xi^{\sum_{i\in\mathrm{I}}\sum_{j=1}^{m}\beta_i\mathsf{F}_{ij}\psi_j - \sum_{i\in\mathrm{I}}\sum_{j=1}^{m}\beta_i\mathsf{F}_{ij}\psi_j}\\
&= (v_1^{r_{i1}} v_2^{r_{i2}})^{\sum_{i\in\mathrm{I}}\beta_i}\\
&= \prod_{i\in\mathrm{I}} x_i^{\beta_i}\\
&= \mathrm{X}.
\end{aligned}$$

Eq. (5.2) also holds since the resp is valid and any x_i, y_{i1}, y_{i2} and t_i $(1 \le i \le n)$ of the Tag_{int} are not forged in the attack. Thus, the verification process returns PASS, which indicates that the POSD scheme fails to satisfy server unforgeability.

Breaking (κ, θ)-uncheatability

Suppose that an adversary $\mathcal{A}$ has uploaded a file F of κ-bit min-entropy in the form of a tuple $(\mathsf{fid}, \mathsf{F}, \widehat{\mathrm{Tag}_{int}})$ to the storage server, and another client $\mathcal{B}$ executes the **DEDUP** protocol to take an ownership of F. Upon receiving of a challenge (I, β) from the server, $\mathcal{B}$ picks a dummy $\mathsf{F'}$ and computes

$$\mu'_j = \sum_{i\in\mathrm{I}} \beta_i \mathsf{F'}_{ij} \bmod q \quad (1 \le j \le m).$$

Then, $\mathcal{B}$ sends $\mathsf{resp} = \left(\{\mu'_j\}_{1\leq j\leq m}\right)$ to the server. In the verification phase, Eqs. (5.3) and (5.4) hold as the calculations are the same as the aforementioned case of breaking server unforgeability. Note that even without knowing any information of F (*i.e.,* $\theta = 0$), $\mathcal{B}$ can pass the **DEDUP** protocol. Hence, the POSD scheme fails to satisfy (κ, θ)-uncheatability.

5.3.2 Attack Scenario

Exploiting the weak keys, several practical attacks against storage systems using the POSD scheme are feasible. Some plausible attack scenarios are listed below.

Malware Distribution

A malware writer can use a storage server as a malware distribution platform by exploiting the weak keys. For clarity, let us denote the malware writer as $\mathcal{A}$, which may play a role of a client in the POSD scheme. In order to deploy the malware effectively, $\mathcal{A}$ may use a setup file F of a popular software like Acrobat Reader or Google Chrome for hosting the malware. $\mathcal{A}$ modifies F into F' by attaching the malware to F and changing the program's execution flow. Executing the **UPLOAD** protocol, $\mathcal{A}$ uploads a tuple (fid, F', $\widehat{\mathrm{Tag}}_{int}$), where fid and $\widehat{\mathrm{Tag}}_{int}$ is for F, to the storage server.

Any (victim) client, denoted as $\mathcal{C}$, who wants to outsource F to the storage server will execute the **DEDUP** protocol with the server. The **DEDUP** protocol will pass though the server actually has only F'. As a result of the protocol, C takes an ownership of file F. When downloading, however, $\mathcal{C}$ will get F' instead of F. In order to verify the integrity of the outsourced file, $\mathcal{C}$ or an auditor may perform the **AUDITINT** protocol with the server. However, the protocol will also pass though the server has modified version of F.

Unintended CDN

A storage server utilizing the POSD scheme also can be used as a CDN (Content Distribution Network) among malicious clients. Suppose that there are two malicious clients, $\mathcal{A}$ and $\mathcal{B}$: $\mathcal{A}$ owns a file F which is potentially huge and probably copyright violating like pirated movie files, and wishes to send F to $\mathcal{B}$ who is not in possession of the file. Generating the weak key, $\mathcal{A}$ uploads F to the storage server through the **UPLOAD** protocol. Then, $\mathcal{B}$ starts executing the **DEDUP** protocol for F. Since $\mathcal{B}$ has no information of F, $\mathcal{B}$ may present a dummy F', given a challenge, to the storage server. The **DEDUP** protocol will pass though $\mathsf{F}' \neq \mathsf{F}$, and $\mathcal{B}$ can take an ownership of F and eventually will download it. As noted in [20], the behavior of $\mathcal{A}$ and $\mathcal{B}$ conflicts with the business model of the cloud storage server, which is meant to support many uploads but few downloads (restores).

5.4 Mitigation and Security Analysis

We present an improved POSD scheme by modifying the original scheme to mitigate the weak key attack described in the previous section. Then, we give a security analysis of the modified scheme.

5.4.1 Security-enhanced Scheme Construction

The security problem caused by the weak key attack comes from the fact that a client is fully capable of controlling its key generation. Hence, our mitigation solution against the attack is to weaken the client's capability by blending parts of the keys with random values contributed by the storage server. This solution requires only a slight modification of the **UPLOAD** protocol, and **KEYGEN, AUDITINT, DEDUP** protocols remain same as those of the original scheme. The modified POSD scheme is described as follows:

KEYGEN: This algorithm generates two pairs of public key and private key. A client runs this protocol as follows:

1. Choose v_1 and v_2 randomly from $\mathbb{Z}_p^*$ such that the orders of subgroups generated by v_1 and v_2 are q. Choose s_{j1} and s_{j2} randomly from $\mathbb{Z}_q^*$ and set $z_j = v_1^{-s_{j1}} v_2^{-s_{j2}}$ for $1 \leq j \leq m$.
2. Choose u uniformly at random from $\mathbb{G}$ and w from $\mathbb{Z}_q^*$. Then set $z_q = g^w$, where g is a generator of $\mathbb{G}$.
3. Set the public key PK_{int}=$\{q, p, g, u, v_1, v_2, z_1, z_2, \ldots, z_m, z_g\}$ and the private key SK_{int}= $\{(s_{11}, s_{12})\ , \ldots, (s_{m1}, s_{m2}), w\}$.
4. Set PK_{dup}=PK_{int} and SK_{dup}=null.

UPLOAD: The modified file uploading protocol is described as follows:

1. For each block F_i $(1 \leq i \leq n)$, the client chooses r_{i1}, r_{i2} at random from $\mathbb{Z}_q^*$ and computes

$$\begin{aligned} x_i &= v_1^{r_{i1}} v_2^{r_{i2}} \bmod p, \\ y_{i1} &= r_{i1} + \sum_{j=1}^{m} \mathsf{F}_{ij} s_{j1} \bmod q, \\ y_{i2} &= r_{i2} + \sum_{j=1}^{m} \mathsf{F}_{ij} s_{j2} \bmod q, \\ t_i &= \left(H_1(\mathsf{fid} \parallel i) u^{H_2(x_i)}\right)^w \quad (\in \mathbb{G})\,. \end{aligned}$$

2. The client sends (fid, F, Tag_{int}) to the server, where Tag_{int}=$\{(x_i, y_{i1}, y_{i2}, t_i)_{1 \leq i \leq n}\}$.

3. Upon receiving (fid, F, Tag$_{int}$), the server selects σ_{j1} and σ_{j2} uniformly at random from $\mathbb{Z}_q^*$ for $1 \leq j \leq m$, and computes with the corresponding public key PK$_{int}$ as follows:

$$z_j' = z_j v_1^{-\sigma_{j1}} v_2^{-\sigma_{j2}} \left(= v_1^{-s_{j1}-\sigma_{j1}} v_2^{-s_{j1}-\sigma_{j1}} \right) \bmod p.$$

For each block F_i, where $1 \leq i \leq n$, the server computes

$$y_{i1}' = y_{i1} + \sum_{j=1}^{m} \mathsf{F}_{ij}\sigma_{j1} \left(= r_{i1} + \sum_{j=1}^{m} \mathsf{F}_{ij}(s_{j1} + \sigma_{j1}) \right),$$
$$y_{i2}' = y_{i2} + \sum_{j=1}^{m} \mathsf{F}_{ij}\sigma_{j2} \left(= r_{i2} + \sum_{j=1}^{m} \mathsf{F}_{ij}(s_{j2} + \sigma_{j2}) \right),$$

where y_{i1}', y_{i2}' are computed under mod q.

4. The server updates the corresponding public keys, PK$_{int}$ and PK$_{dup}$, by replacing z_j with z_j' for $1 \leq j \leq m$ and updates Tag$_{int}$ by replacing y_{i1}, y_{i2} with y_{i1}', y_{i2}' for $1 \leq i \leq n$. Then the server sets Tag$_{dup}$=Tag$_{int}$.

AUDITINT: An auditor, which can be the client itself, and the server run this protocol as follows:

1. The auditor chooses a set of c elements I=$\{\alpha_1, \alpha_2, \ldots, \alpha_c\}$, where $\alpha_i \in \mathbb{N}$, and chooses a set of coefficients $\beta = \{\beta_1, \beta_2, \ldots, \beta_c\}$, where $\beta_i \in \mathbb{Z}_q^*$. The auditor sends a challenge chal $= (\mathrm{I}, \beta)$ to the server.

2. Upon receiving chal, the server computes

$$\mu_j = \sum_{i \in \mathrm{I}} \beta_i \mathsf{F}_{ij} \bmod q \ (1 \leq j \leq m),$$
$$\mathrm{Y}_1 = \sum_{i \in \mathrm{I}} \beta_i y_{i1} \bmod q,$$
$$\mathrm{Y}_2 = \sum_{i \in \mathrm{I}} \beta_i y_{i2} \bmod q,$$
$$\mathrm{T} = \prod_{i \in \mathrm{I}} t_i^{\beta_i} \quad (\in \mathbb{G})$$

and sends resp=$(\{\mu_j\}_{1 \leq j \leq m}, \{x_i\}_{i \in \mathrm{I}}, \mathrm{Y}_1, \mathrm{Y}_2, \mathrm{T})$ to the auditor.

3. Upon receiving resp, the auditor computes :

$$\mathrm{X} = \prod_{i \in \mathrm{I}} x_i^{\beta_i} \bmod p,$$
$$\mathrm{W} = \prod_{i \in \mathrm{I}} H_1(\mathsf{fid} \parallel i)^{\beta_i}$$

   and verifies

$$\mathrm{X} \overset{?}{=} v_1^{\mathrm{Y}_1} v_2^{\mathrm{Y}_2} \prod_{j=1}^{m} z_j^{\mu_j} \bmod p$$
$$e(\mathrm{T}, g) \overset{?}{=} e(\mathrm{W} u^{\sum_{i \in \mathrm{I}} \beta_i H_2(x_i)}, z_g) \quad (\in \mathbb{G}_T)\,.$$

   If both hold, return PASS; otherwise, return FAIL.

DEDUP: The server and the client run this protocol as follows:

1. The server generates a challenge $\mathsf{chal} = (\mathrm{I}, \beta)$, where $\mathrm{I}=\{\alpha_1, \alpha_2, \ldots, \alpha_c\}$ and $\beta = \{\beta_1, \beta_2, \ldots, \beta_c\}$ as the **AUDITINT** protocol, and sends it to the client.

2. Upon receiving chal, the client computes

$$\mu_j = \sum_{i \in \mathrm{I}} \beta_i \mathsf{F}_{ij} \bmod q \ , \ 1 \leq j \leq m$$

   and sends $\mathsf{resp} = (\{\mu_j\}_{1 \leq i \leq m})$ to the server.

3. Upon receiving resp, the server verifies

$$\mathrm{X} \overset{?}{=} v_1^{\mathrm{Y}_1} v_2^{\mathrm{Y}_2} \prod_{j=1}^{m} z_j^{\mu_j} \bmod p$$
$$e(\mathrm{T}, g) \overset{?}{=} e(\mathrm{W} u^{\sum_{i \in \mathrm{I}} \beta_i H_2(x_i)}, z_g) \quad (\in \mathbb{G}_T)\,,$$

   where $\mathrm{Y}_1, \mathrm{Y}_2, \mathrm{W}, \mathrm{X}$ and T are computed from $\mathrm{Tag}_{dup}=\{(x_i, y_{i1}, y_{i2}, t_i)_{i \leq i \leq n}\}$. If both hold, return PASS; otherwise, return FAIL.

5.4.2 Security Analysis

Now we show that the modified scheme satisfies two security requirements, server unforgeability and (κ, θ)-uncheatability, even under the weak key attack.

Theorem 5. *The modified* POSD *scheme is server unforgeable and (κ, θ)-uncheatable in the random oracle model without the random key assumption.*

Proof. The proof is presented through a hybrid argument. In each game, an adversary $\mathcal{A}$ plays a role of the malicious client or the malicious server, and the challenger verifies the adversary $\mathcal{A}$ in the role of the sever or a client (an auditor).

Game0: In Game0, $\mathcal{A}$ and the challenger run the original POSD scheme assuming that all clients including $\mathcal{A}$ honestly execute the **KEYGEN** algorithm (the random key assumption). $\mathcal{A}$ tries to fool the challenger by breaking the scheme within the polynomially bounded resources. Given the random key assumption, $\mathcal{A}$ can break the scheme just with negligible probability, as proved in [62]. Hence, the required security properties are preserved in Game0.

Game1: The only difference between Game0 and Game1 is that in Game1, the original POSD scheme is replaced with the modified scheme, where additional values $\sigma_{j1}, \sigma_{j2} \in \mathbb{Z}_q^*$ $(1 \leq j \leq m)$ are chosen at random and supplemented in the computation of the **UPLOAD** protocol. Since the distributions of $\{\sigma_{j1}, \sigma_{j2}\}_{1 \leq j \leq m}$ are statistically independent from that of the other variables of the scheme, $\mathcal{A}$ has no additional advantage breaking the modified scheme even with the knowledge of $\{\sigma_{j1}, \sigma_{j2}\}_{1 \leq j \leq m}$. Hence, the required security properties are preserved in Game1.

Game2: In Game2, we remove the random key assumption from Game1. $\mathcal{A}$ may try to break the modified POSD scheme through generating weak keys. Suppose that $\mathcal{A}$ generates weak keys such that $v_1 = \xi, v_2 = \xi^{-1}$ and $s_{j1} = s_{j2} = \psi_j$ for $1 \leq j \leq m$ and executes the **UPLOAD** protocol with the server. At the end of the protocol, the corresponding $\widehat{\mathrm{PK}}_{int}$ and $\widehat{\mathrm{Tag}}_{int}$ ($\widehat{\mathrm{PK}}_{dup}$ and $\widehat{\mathrm{Tag}}_{dup}$) will be updated so that s_{j1}, s_{j2} $(1 \leq j \leq m)$ are blended with randoms generated by the server. Hence, $\mathcal{A}$ can control just only v_1 and v_2 in the key generation.

We show that if $\mathcal{A}$ (in the role of the server) is still able to fool the auditor by making the keys such that $v_1 = \xi$ and $v_2 = \xi^{-1}$, then there is an efficient algorithm to compute the DLOG (Discrete Logarithm Problem) with respect to base ξ. Suppose that a DLOG-solving algorithm $\mathcal{B}$ is given $a \in \mathbb{Z}_p^*$ as an instance of the DLOG problem with respect to base ξ. $\mathcal{B}$ can solve the problem by interacting with $\mathcal{A}$. The algorithm $\mathcal{B}$ is constructed as follows:

- First, $\mathcal{B}$ generates the keys as follows: $\mathcal{B}$ chooses s_{j1}, s_{j2} at random from $\mathbb{Z}_q^*$ for $2 \leq j \leq m$ and set

$$v_1 = \xi, v_2 = \xi^{-1},$$
$$z_1 = a,\ z_j = v_1^{-s_{j1}} v_2^{-s_{j2}} \quad (2 \leq j \leq m).$$

Then, $\mathcal{B}$ selects the remaining part of the keys according to **KEYGEN** algorithm.

- $\mathcal{B}$ answers H_1 and H_2 queries through modeling $H_1(\cdot)$ and $H_2(\cdot)$ as random oracles. In addition, $\mathcal{B}$ may interact with $\mathcal{A}$ executing the **UPLOAD, AUDITINT** and **DEDUP** protocols on behalf of the server.
- When $\mathcal{B}$ is asked to compute Tag$_{int}$ for F, $\mathcal{B}$ executes the following: For each F_i $(1 \leq i \leq n)$, $\mathcal{B}$ chooses y_{i1}, y_{i2} at random from $\mathbb{Z}_q^*$ and computes

$$x_i = v_1^{y_{i1}} v_2^{y_{i2}} \prod_{j=1}^{m} z_j^{\mathsf{F}_{ij}},$$
$$t_i = \left(H_1(\mathsf{fid} \parallel i) u^{H_2(x_i)}\right)^w.$$

  Then, $\mathcal{B}$ returns Tag$_{int}$=$\{(x_i, y_{i1}, y_{i2}, t_i)_{1 \leq i \leq n}\}$ to $\mathcal{A}$.

- Eventually, $\mathcal{A}$ outputs a forgery of the original file F as resp=$(\{\mu'_j\}_{1 \leq j \leq m}, \mathrm{Y}'_1, \mathrm{Y}'_2, \mathrm{T}', \{x'_i\}_{i \in \mathrm{I}})$ which is computed with F' $(\neq \mathsf{F})$. Note that $\mathrm{T}' = \mathrm{T}$ and $x'_i = x_i$ for all $i \in \mathrm{I}$ since otherwise, $\mathcal{A}$ would be used for solving the CDH problem [62].
- Let us assume $z_1 (= a) = \xi^x$. The following verification holds since resp= $(\{\mu'_j\}_{1 \leq j \leq m}, \mathrm{Y}'_1, \mathrm{Y}'_2, \mathrm{T}', \{x'_i\}_{i \in \mathrm{I}})$ is the valid response to the challenge:

$$\begin{aligned}
& v_1^{\mathrm{Y}'_1} v_2^{\mathrm{Y}'_2} \prod_{j=1}^{m} z_j^{\mu'_j} \\
&= \left(v_1^{\mathrm{Y}_1} v_2^{\mathrm{Y}_2} \prod_{j=1}^{m} z_j^{\mu_j}\right) \left(v_1^{\mathrm{Y}'_1 - \mathrm{Y}_1} v_2^{\mathrm{Y}'_2 - \mathrm{Y}_2} \prod_{j=1}^{m} z_j^{\mu'_j - \mu_j}\right) \\
&= \mathrm{X} \left(\xi^{(\mathrm{Y}'_1 - \mathrm{Y}_1) - (\mathrm{Y}'_2 - \mathrm{Y}_2)} \prod_{j=1}^{m} z_j^{\mu'_j - \mu_j}\right) \\
&= \mathrm{X} \left(\xi^{(\mathrm{Y}'_1 - \mathrm{Y}_1) - (\mathrm{Y}'_2 - \mathrm{Y}_2)} \xi^{\sum_{j=2}^{m} (-s_{j1} + s_{j2})(\mu'_j - \mu_j)} z_1^{\mu'_1 - \mu_1}\right) \\
&= \mathrm{X} \left(\xi^{(\mathrm{Y}'_1 - \mathrm{Y}_1) - (\mathrm{Y}'_2 - \mathrm{Y}_2) + \sum_{j=2}^{m} (-s_{j1} + s_{j2})(\mu'_j - \mu_j) + x(\mu'_1 - \mu_1)}\right) \\
&= \mathrm{X}.
\end{aligned}$$

  Thus, $\mathcal{B}$ outputs the DLOG of a with respect to base ξ as

$$x = \frac{(\mathrm{Y}_1' - \mathrm{Y}_1) - (\mathrm{Y}_2' - \mathrm{Y}_2) + \sum_{j=2}^{m} (-s_{j1} + s_{j2})(\mu_j' - \mu_j)}{\mu_1 - \mu_1'}.$$

For the adversary $\mathcal{A}$ acting as the client in **DEDUP** protocol, it also can be shown that the adversary can be converted into the DLOG-solving algorithm in similar manner. Thus, the required security properties are still preserved in Game2. $\square$

5.5 Summary

In order to make cloud storage services to be secure while losing no efficiency, it is required for the system to achieve all desired security properties; PDP (Provable Data Possession), POR (Proof of Retrievability) and POW (Proof of Ownership). Proof of storage with deduplication (POSD) [62] is the first scheme that fits that requirement. In this chapter, a security weakness of the POSD scheme was addressed. Under new attack model that allows malicious clients to exploit dishonestly manipulated keys, the POSD scheme fails to guarantee required security. In order to mitigate the attack, a security-enhanced scheme against any attacks exploiting the vulnerability was proposed. The main approach is to blend client's keys with the server contributed random values. By doing so, the proposed scheme keeps all the desired security properties even under the new attack model. The proposed scheme was proved to guarantee the security based on the CDH (Computational Diffie-Hellman) assumption and the DLOG (Discrete Log) assumption in the random oracle model.

Chapter 6. Concluding Remark

6.1 Conclusion

For the fast growth of cloud storage services, it is certainly necessary to address the problem of data security and privacy on the outsourced data without losing efficiency in resource utilization. The challenge is to build a data deduplication system which offers cost savings in terms of utilization of disk space and network bandwidth, while providing data security and privacy against the untrusted cloud server and the unauthorized users. In the dissertation, we studied the security implications of data deduplication techniques in the cloud computing environment in various aspects, and proposed novel solutions for secure and efficient data deduplication in cloud storage systems.

First, we proposed a secure file deduplication scheme that offers data confidentiality on the outsourced data as well as the storage and network efficiency. For the basis of building the deduplication scheme, we constructed symmetric-key equality predicate encryption algorithms, by which the cloud server is allowed to perform deduplication over encrypted files without getting any information about their content. The proposed scheme adopts an hybrid approach so that deduplication occurs either of at server side or at client side in randomized manner. This randomized strategy greatly increases attack overhead of online-guessing adversaries, hence reduces the risk of information leakage on the stored data. We presented analysis of the proposed scheme as well as the constructed encryption algorithms in terms of security and performance.

Second, we addressed a problem of information leakage through the side channel in client-side data deduplication. We also discussed the security weakness of previous solutions under newly proposed attack model as well as their inefficiency. For enhancing security and privacy on the outsourced data, we proposed a differentially private client-side data deduplication protocol, which strongly guarantees that the side channel adversaries have difficulty in knowing the existence of individual data in the cloud storage. By utilizing a storage gateway in the cloud storage infrastructure, the proposed scheme prevents generating unnecessary network traffic and eliminates the side channel. Through the rigorous analysis of security and performance, we showed that the proposed solution is very effective.

Third, we addressed a security problem of a proof of storage with deduplication in a cloud storage system. We showed that the previous scheme fails to satisfy the desired security requirements under a new attack model in which malicious users run the protocol with dishonestly manipulated keys. For mitigating such an attack and improving security, we presented a security enhanced scheme. In the proposed scheme, the user keys are blended with the random values contributed by the cloud storage server, hence the adversary's capability to manipulate their keys is weakened. We also showed that the proposed scheme preserves the efficiency while providing more robust security.

The secure deduplication schemes that have been proposed in the dissertation bring high level of efficiency in terms of utilizing the resources of storage space and network bandwidth. These schemes also ensure strong data security and privacy against an untrusted cloud server and unauthorized users. We expect that the proposed schemes will contribute to the advance of secure deduplication techniques for cloud storage services.

6.2 Future Work

The literature of secure data deduplication in the cloud computing environment is relatively new, and there are still many challenging problems to be resolved to accelerate the growth of cloud storage services. As future work, we will extend the proposed solutions with more security features such as secure deletion from the untrusted and persistent storage, flexible and fine-grained access control over encrypted storage with deduplication, and scalable convergent key management. We also intend to enhance the proposed solutions so that they are suitable for real applications. We will provide full implementations of the proposed schemes and test them in real cloud environments.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, *et al.*, "Above the clouds: A berkeley view of cloud computing," Technical Report UCB/EECS-2009-28, Dept. EECS, UC Berkerely, 2009.

[2] D. Russell, "Data deduplication will be even bigger in 2010," *Gartner*, 2010.

[3] D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Side channels in cloud services: Deduplication in cloud storage," *IEEE Security and Privacy Magazine*, vol. 8, pp. 40–47, 2010.

[4] S. Yu, C. Wang, K. Ren, and W. Lou, "Achieving secure, scalable, and fine-grained data access control in cloud computing," in *Proc. IEEE Conference on Computer Communications (INFOCOM'10)*, pp. 534–542, 2010.

[5] S. Yu, C. Wang, K. Ren, and W. Lou, "Attribute based data sharing with attribute revocation," in *Proc. ACM Symposium on Information, Computer and Communications Security (ASIACCS'10)*, pp. 261–270, 2010.

[6] J. Hur and D. Noh, "Attribute-based access control with efficient revocation in data outsourcing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, pp. 1214–1221, 2011.

[7] Z. Zhou and D. Huang, "Efficient and secure data storage operations for mobile cloud computing." Cryptology ePrint Archive, Report 2011/185, 2011. http://eprint.iacr.org/.

[8] M. Green, S. Hohenberger, and B. Waters, "Outsourcing the decryption of abe ciphertexts," in *Proc. USENIX Security Symposium (SEC'11)*, 2011.

[9] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," *Advances in Cryptology - EUROCRYPT'04,* LNCS 3027, pp. 506–522, 2004.

[10] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: improved definitions and efficient constructions," in *Proc. ACM Conference on Computer and Communications Security (CCS'06)*, pp. 79–88, 2006.

[11] M. Bellare, A. Boldyreva, and A. O'Neill, "Deterministic and efficiently searchable encryption," *Advances in Cryptology - CRYPTO'07,* LNCS 4622, pp. 535–552, 2007.

[12] S. Sedghi, P. van Liesdonk, J. M. Doumen, P. H. Hartel, and W. Jonker, "Adaptively secure computationally efficient searchable symmetric encryption," Technical Report TR-CTIT-09-13, Centre for Telematics and Information Technology University of Twente, 2009.

[13] C. Dong, G. Russello, and N. Dulay, “Shared and searchable encrypted data for untrusted servers,” *Journal of Computer Security*, vol. 19, pp. 367–397, 2011.

[14] M. Mulazzani, S. Schrittwieser, M. Leithner, M. Huber, and E. Weippl, “Dark clouds on the horizon: using cloud storage as attack vector and online slack space,” in *Proc. USENIX Security Symposium (SEC’11)*, 2011.

[15] M. W. Storer, K. Greenan, D. D. Long, and E. L. Miller, “Secure data deduplication,” in *Proc. ACM International Workshop on Storage security and survivability (StorageSS’08)*, pp. 1–10, 2008.

[16] R. D. John, A. Atul, J. B. William, S. Dan, and T. Marvin, “Reclaiming space from duplicate files in a serverless distributed file system,” in *Proc. International Conference on Distributed Computing Systems (ICDCS’02)*, 2002.

[17] L. Marques and C. J. Costa, “Secure deduplication on mobile devices,” in *Proc. Workshop on Open Source and Design of Communication (OSDOS’11)*, pp. 19–26, 2011.

[18] P. Anderson and L. Zhang, “Fast and secure laptop backups with encrypted deduplication,” in *Proc. International Conference on Large installation system administration (LISA’10)*, pp. 1–8, 2010.

[19] S. Goldwasser and S. Micali, “Probabilistic encryption,” *Journal of computer and system sciences*, vol. 28, no. 2, pp. 270–299, 1984.

[20] S. Halevi, D. Harnik, B. Pinkas, and A. Shulman-Peleg, “Proofs of ownership in remote storage systems,” in *Proc. ACM Conference on Computer and Communications Security (CCS’11)*, pp. 491–500, 2011.

[21] V. Goyal, O. Pandey, A. Sahai, and B. Waters, “Attribute-based encryption for fine-grained access control of encrypted data,” in *Proc. ACM Conference on Computer and Communications Security (CCS’06)*, pp. 89–98, 2006.

[22] J. Bethencourt, A. Sahai, and B. Waters, “Ciphertext-policy attribute-based encryption,” in *Proc. IEEE Symposium on Security and Privacy (SP’07)*, pp. 321–334, 2007.

[23] D. Naor, M. Naor, and J. B. Lotspiech, “Revocation and tracing schemes for stateless receivers,” *Advances in Cryptology - CRYPTO’01,* LNCS 2139, pp. 41–62, 2001.

[24] A. Juels and B. S. Kaliski, Jr., “Pors: Proofs of retrievability for large files,” in *Proc. ACM Conference on Computer and Communications Security (CCS’07)*, pp. 584–597, 2007.

[25] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, “Provable data possession at untrusted stores,” in *Proc. ACM Conference on Computer and Communications Security (CCS’07)*, pp. 598–610, 2007.

[26] H. Shacham and B. Waters, "Compact proofs of retrievability," in *Advances in Cryptology - ASIACRYPT'08,* LNCS 5350, pp. 90–107, 2008.

[27] Y. Dodis, S. Vadhan, and D. Wichs, "Proofs of retrievability via hardness amplification," in *Proc. Conference on Theory of Cryptography (TCC'09),* LNCS 5444, pp. 109–127, 2009.

[28] K. D. Bowers, A. Juels, and A. Oprea, "Proofs of retrievability: Theory and implementation," in *Proc. ACM Workshop on Cloud Computing Security (CCSW'09)*, pp. 43–54, 2009.

[29] E. Shi, E. Stefanov, and C. Papamanthou, "Practical dynamic proofs of retrievability," in *Proc. ACM Conference on Computer and Communications Security (CCS'13)*, pp. 325–336, 2013.

[30] G. Ateniese, R. D. Pietro, L. V. Mancini, and G. Tsudik, "Scalable and efficient provable data possession," in *Proc. International Conference on Security and Privacy in Communication Networks (SecureComm'08)*, pp. 9:1–9:10, 2008.

[31] C. Erway, A. Küpçü, C. Papamanthou, and R. Tamassia, "Dynamic provable data possession," in *Proc. ACM Conference on Computer and Communications Security (CCS'09)*, pp. 213–222, 2009.

[32] Y. Zhang and M. Blanton, "Efficient dynamic provable possession of remote data via balanced update trees," in *Proc. ACM Symposium on Information, Computer and Communications Security (ASIACCS'13)*, pp. 183–194, 2013.

[33] E. Shen, E. Shi, and B. Waters, "Predicate privacy in encryption systems," in *Proc. Conference on Theory of Cryptography (TCC'09),* LNCS 5444, pp. 457–473, 2009.

[34] C. Blundo, V. Iovino, and G. Persiano, "Private-key hidden vector encryption with key confidentiality," in *Proc. Cryptography and Network Security 2009 (CANS'09)*, pp. 259–277, 2009.

[35] D. Boneh, A. Sahai, and B. Waters, "Functional encryption: definitions and challenges," in *Proc. Conference on Theory of Cryptography (TCC'11),* LNCS 6597, pp. 253–273, 2011.

[36] D. Boneh and M. K. Franklin, "Identity-based encryption from the weil pairing," *Advances in Cryptology - CRYPTO'01,* LNCS 2139, pp. 213–229, 2001.

[37] A. Joux, "The weil and tate pairings as building blocks for public key cryptosystems," in *Proc. International Symposium on Algorithmic Number Theory (ANTS'05)*, pp. 20–32, 2002.

[38] D. Boneh and B. Waters, "A fully collusion resistant broadcast, trace, and revoke system," in *Proc. ACM Conference on Computer and Communications Security (CCS'06)*, pp. 211–220, 2006.

[39] L. Jin, C. Xiaofeng, L. Mingqiang, L. Jingwei, L. Patrick, and L. Wenjing, "Secure deduplication with efficient and reliable convergent key management," *IEEE Transactions on Parallel and Distributed Systems*, vol. PP, 2013.

[40] M. Bellare, S. Keelveedhi, and T. Ristenpart, "Dupless: Server-aided encryption for deduplicated storage," in *Proc. USENIX Security Symposium (SEC'13)*, pp. 179–194, 2013.

[41] M. Bellare, S. Keelveedhi, and T. Ristenpart, "Message-locked encryption and secure deduplication," *Advances in Cryptology - EUROCRYPT'13,* LNCS 7881, pp. 296–312, 2013.

[42] D. Yitao, "Distributed key generation for secure encrypted deduplication." Cryptology ePrint Archive, Report 2013/807, 2013. http://eprint.iacr.org/.

[43] M. Bellare, S. Keelveedhi, and T. Ristenpart, "Message-locked encryption and secure deduplication (full version)." Cryptology ePrint Archive, Report 2012/631, 2012. http://eprint.iacr.org/.

[44] T. Pulls, "(more) side channels in cloud storage," *Privacy and Identity Management for Life*, vol. 375, pp. 102–115, 2012.

[45] S. Lee and D. Choi, "Privacy-preserving cross-user source-based data deduplication in cloud storage," in *Proc. International Conference on ICT Convergence*, pp. 329–330, 2012.

[46] Olivier, C. Neumann, L. Montalvo, and S. Defrance, "Improving the resistance to side-channel attacks on cloud storage services," in *Proc. International Conference on New Technologies, Mobility and Security (NTMS'12)*, pp. 1–5, 2012.

[47] R. Merkle, "A certified digital signature," *Advances in Cryptology - CRYPTO'90,* LNCS 435, pp. 218–238, 1990.

[48] W. K. Ng, Y. Wen, and H. Zhu, "Private data deduplication protocols in cloud storage," in *Proc. ACM Symposium on Applied Computing (SAC'12)*, pp. 441–446, 2012.

[49] X. Jia, C. Ee-Chien, and Z. Jianying, "Leakage-resilient client-side deduplication of encrypted data in cloud storage." Cryptology ePrint Archive, Report 2011/538, 2011. http://eprint.iacr.org/.

[50] R. Di Pietro and A. Sorniotti, "Boosting efficiency and security in proof of ownership for deduplication," in *Proc. ACM Symposium on Information, Computer and Communications Security (ASIACCS'12)*, pp. 81–82, 2012.

[51] J. Daemen and V. Rijmen, *The design of Rijndael: AES-the advanced encryption standard.* Springer, 2002.

[52] M. Bellare and P. Rogaway, "Random oracles are practical: a paradigm for designing efficient protocols," in *Proc. ACM Conference on Computer and Communications Security (CCS'93)*, pp. 62–73, 1993.

[53] S. Zaffos and A. Couture, "Hybrid cloud gateway appliances expand cloud storage use cases," *Gartner*, 2011.

[54] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in *Proc. Conference on Theory of Cryptography (TCC'06),* LNCS 3876, pp. 265–284, 2006.

[55] C. Dwork, "Differential privacy: A survey of results," *Theory and Applications of Models of Computation,* LNCS 4978, pp. 1–19, 2008.

[56] C. Dwork and A. Smith, "Differential privacy for statistics: What we know and what we want to learn," *Journal of Privacy and Confidentiality*, vol. 1, no. 2, pp. 135–154, 2010.

[57] L. Sweeney, "k-anonymity: A model for protecting privacy," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 10, no. 5, pp. 557–570, 2002.

[58] A. Narayanan and V. Shmatikov, "Robust de-anonymization of large sparse datasets," in *Proc. IEEE Symposium on Security and Privacy (SP'08)*, pp. 111–125, 2008.

[59] S. Fahl, M. Harbach, T. Muders, L. Baumgärtner, B. Freisleben, and M. Smith, "Why eve and mallory love android: An analysis of android ssl (in)security," in *Proc. ACM Conference on Computer and Communications Security (CCS'12)*, pp. 50–61, 2012.

[60] H. Krawczyk, K. G. Paterson, and H. Wee, "On the security of the tls protocol: A systematic analysis," *Advances in Cryptology - CRYPTO'13,* LNCS 8042, pp. 429–448, 2013.

[61] F. Rashid, A. Miri, and I. Woungang, "Proof of retrieval and ownership protocols for enterprise-level data deduplication," in *Proc. Conference on the Center for Advanced Studies on Collaborative Research (CASCON'13)*, pp. 81–90, 2013.

[62] Q. Zheng and S. Xu, "Secure and efficient proof of storage with deduplication," in *Proc. ACM Conference on Data and Application Security and Privacy (CODASPY'12)*, pp. 1–12, 2012.

Summary

Secure and Efficient Data Deduplication Techniques for Cloud Storage Systems

클라우드 스토리지 서비스는 스토리지 저장 용량과 네트워크 대역폭 등 IT 자원을 효율적으로 사용하기 위해 대부분 데이터 중복 제거 기술을 사용하고 있다. 데이터 중복 제거 기술이란 여러 사용자가 동일한 파일을 스토리지에 아웃소싱 하게 되는 경우 물리적으로 하나의 파일을 제외하고 나머지는 논리적인 링크로 대체하는 기술을 말한다. 데이터 중복 제거 기술은 중복 제거가 일어나는 위치에 따라 서버 방식과 클라이언트 방식으로 구분할 수 있다. 서버 방식의 중복 제거는 클라우드 서버에서 중복된 파일을 찾아 제거하는 방식이며 클라이언트 방식은 클라이언트가 파일을 업로드 하기 전에 중복 여부를 탐지하여 중복을 제거한다. 클라이언트 방식의 중복 제거는 서버의 스토리지 용량을 절감할 수 있을 뿐 아니라 네트워크에서 소모되는 대역폭도 절감할 수 있는 장점이 있다. 이러한 데이터 중복 제거 기술은 디스크 저장 공간과 네트워크 대역폭을 최대 90% 가까이 절감할 수 있는 것으로 나타났다. 따라서 IT 자원의 효율적 이용이 금전적 비용의 절감과 직결되면서 데이터 중복 제거 기술은 클라우드 스토리지 서비스 사업자에게는 반드시 필요한 기술로 받아들여지고 있다.

그러나 현재의 데이터 중복 제거 기술은 사용자의 데이터에 대해 심각한 보안 위협을 초래한다. 클라우드 서버가 사용자의 파일들에 대해 중복 여부를 판단하기 위해서는 파일이 평문 형태로 저장이 되어야한다. 각 사용자의 암호키로 암호화되어 저장된다면 중복 여부를 판단할 수 없기 때문이다. 따라서 클라우드 서버 시스템에 침투한 해커 등 외부 공격자로부터 사용자의 데이터에 대한 기밀성과 프라이버시가 전혀 보장되지 못한다. 침해 사고에 대비해 클라우드 서버가 저장 데이터에 대하여 직접 암호화를 할 수 는 있으나 역시 신뢰할 수 없는 클라우드 서버에 대하여는 사용자 데이터를 보호할 수 없다.

데이터 중복 제거 기술이 야기하는 또 다른 형태의 보안 문제도 존재한다. 최근의 연구에 따르면 클라이언트 방식의 중복 제거를 사용하는 클라우드 스토리지 서비스에서 비인가 사용자가 부 채널을 이용하여 저장된 파일의 정보를 획득할 수 있는 보안 취약점이 발견되었다. 이 취약점을 이용하면 공격자는 온라인 추측 공격을 실행하여 저장된 파일의 내용을 복구할 수 있다. 공격 방법은 간단하다. 공격자는 파일의 내용을 추측하여 후보 파일을 만들고 이를 클라우드 서버에 업로드를 요청한다. 클라우드 서버는 파일 업로드를 하기 전에 파일의 중복 여부를 판단하여 동일 파일이 존재하면 업로드를 하지 않고 요청을 처리한다. 공격자는 네트워크 전송 크기를 모니터링 하여 후보 파일이 실제로 업로드 되는지를 판단할 수 있다. 공격자는 이러한 과정을 반복하면서 서버에 저장된 파일의 내용을 알아낼 수 있다. 현재의 데이터 중복 제거 기술은 위의 두 가지 보안 위협에 대하여 사용자 데이터에 대한 안전성을 전혀 보장하지 못한다. IT 자원의 효율적 활용을 가능하게 하는 동시에 사용자 데이터의 안전성을 보장하는 안전한 데이터 중복 제거 기술의 구현은 아직까지 해결하지 못한 문제로 남아있다.

본 연구는 이 문제를 다루어 효율적이면서도 안전한 데이터 중복 제거 기법을 제시한다. 첫째, 신뢰할 수 없는 클라우드 서버와 비인가 사용자들에게 저장된 데이터의 정보가 노출되는 것을 방지할 수 있는 효율적 데이터 중복 제거 기법을 제안한다. 이를 위해 대칭키 환경의 동치술어 암호 시스템을 구현하고 이에 기반한 데이터 중복 제거 프로토콜을 구현한다. 동치술어 암호란 두 암호문이 주어졌을 때 그 평문의 동치 관계를 판단할 수 있는 암호 시스템을 말한다. 이를 활용하면 클라우드 서버는 사용자가 암호화한 파일에 대해서도 중복 여부를 판단할 수 있게 된다. 또한 온라인 추측 공격으로부터 사용자의 데이터를 보호하기 위하여 하이브리드 방식의 중복 제거 방식도 제안한다. 하이브리드 방식이란 일정한 확률에 따라 클라이언트 또는 서버에서 중복 제거를 실행하는 것으로 온라인 추측 공격의 복잡도를 증가시키고 데이터의 정보가 노출될 위험을 최소화 시킬 수 있다.

둘째, 클라이언트 방식의 데이터 중복 제거 기술을 사용 시 부 채널에 의한 정보 유출 문제를 다루고 기존에 제안 된 기법들의 안전성 및 효율성 문제에 대해 논의한다. 부 채널을 방지하면서 보다 향상된 안전성 기준을 만족하기 위하여 차분적 프라이버시를 보장하는 새로운 클라이언트 데이터 중복 제거 기법을 제안한다. 차분적 프라이버시는 특정 데이터가 클라우드 서버에 존재하는지 또는 존재하지 않는 지에 대한 정보가 공격자에게 쉽게 노출되지 않는 것을 보장하는 새로운 안전성 개념이다. 본 연구에서 제안한 기법은 차분적 프라이버시 개념을 충족하면서 기존에 제안된 기법들에 비해 네트워크 대역폭 낭비가 대폭 줄어드는 장점을 가지고 있다.

셋째, 클라우드 스토리지에서 데이터 중복 제거와 소유권 증명 및 무결성 검증을 모두 제공하는 기법을 고안할 때 발생하는 보안 문제에 대해 논의 하고 안전성을 향상시키기 위한 방안을 제안한다. 구체적으로, 사용자가 자신의 비밀키를 악의적인 방법으로 생성할 수 있는 새로운 보안 모델을 제시하고, 이 보안 모델에서 기존에 제안 된 기법에 보안 취약점이 존재함을 보인다. 그리고 새로운 보안 모델에서 안전성을 보장하기 위하여 기존 기법을 개선한 방법을 제안한다. 제안한 방법은 기존의 안전성과 효율성을 그대로 유지하면서 새 보안 모델에서의 공격도 차단할 수 있는 장점을 가지고 있다.

본 연구에서 제시된 기법들은 모두 스토리지 공간과 네트워크 대역폭 활용 측면에서 효율성을 제공하는 동시에 클라우드 서버와 비인가 사용자 등 내외부 공격자들로부터 높은 수준의 데이터 기밀성과 프라이버시를 보장한다. 본 제안 방법들이 앞으로 클라우드 스토리지 시스템에서 더욱 효율적이고 안전한 데이터 중복 제거 기술이 개발되는 데 기여할 수 있을 것으로 기대한다.