# Solving the Secure Storage Dilemma: An Efficient Scheme for Secure Deduplication with Privacy-Preserving Public Auditing

Süleyman Kardaş* and Mehmet Sabır Kiraz+
*Dept. of Computer Science and Engineering, Batman University, Turkey
+ Mathematical and Computational Sciences Labs
TÜBİTAK BİLGEM, Turkey
E-mail: skardas@gmail.com, mehmet.kiraz@tubitak.gov.tr

**Abstract.** Existing cloud storage systems obtain the data in its plaintext form and perform conventional (server-side) deduplication mechanisms. However, disclosing the data to the cloud can potentially threaten the security and privacy of users, which is of utmost importance for a real-world cloud storage. This can be solved by secure deduplication mechanisms which enables the user to encrypt the data on the client-side (or via an encryption-as-a-service module) before uploading it to the cloud storage. Conventional client-side encryption solutions unfortunately make the deduplication more challenging. Privacy-preserving public auditing schemes, on the other hand, is also crucial because the clients outsource their data to the cloud providers and then permanently deletes the data from their local storages. In this paper, we consider the problem of secure deduplication over encrypted data stored in the cloud while supporting a privacy-preserving public auditing mechanism. We show that existing solutions cannot support both goals simultaneously due to the conflict of their security and efficiency requirements. In this respect, we present an efficient and secure deduplication scheme that supports client-side encryption and privacy-preserving public auditing. We finally show that our scheme provides better security and efficiency with respect to the very recently proposed existing schemes.

**Keywords:** Secure Client-Side Deduplication, Public auditing, Privacy, Cloud Storage.

## 1 Introduction

When $n$ clients upload the same copy of a file $F$, the cloud provider stores only one copy that is sent back to these clients upon download. This is called *deduplication* technology which is a useful method for reducing data redundancy on storage space, reducing computations on disk storage, saving on network bandwidth, and speeding up the actual backups. The goal of a deduplication mechanism is to find out basically all identical files on the storage space. More storage costs on the server-side would lead to the clients to pay higher storage fees making deduplication to be one of the most cost saving mechanism for Cloud Storage Providers (CSPs).

In terms of location of deduplication processing, the underlying techniques can be classified into two methods: (1) deduplication occurs before the upload on the client-side (see Figure 1), (2) deduplication occurs after the upload on the server-side (see Figure 2). Deduplication can also be classified into two methods in terms of data processing method: (1) deduplication at a block level where the data is divided into smaller fixed-size or variable-size blocks and only a single copy of each block is stored [1], (2) at a file level where only a single copy of a file is stored [2].

The server-side deduplication allows a server to check whether the two stored files are identical. In this case, the client first uploads a file, and then the server can internally check if there is a duplication. In this case, the clients do not know whether there is a duplicate. Server-side deduplication mechanisms are easy to handle and already in use [3–5]. However, server-side solutions only reduces the cost of storage for the server and do not reduce the bandwidth.

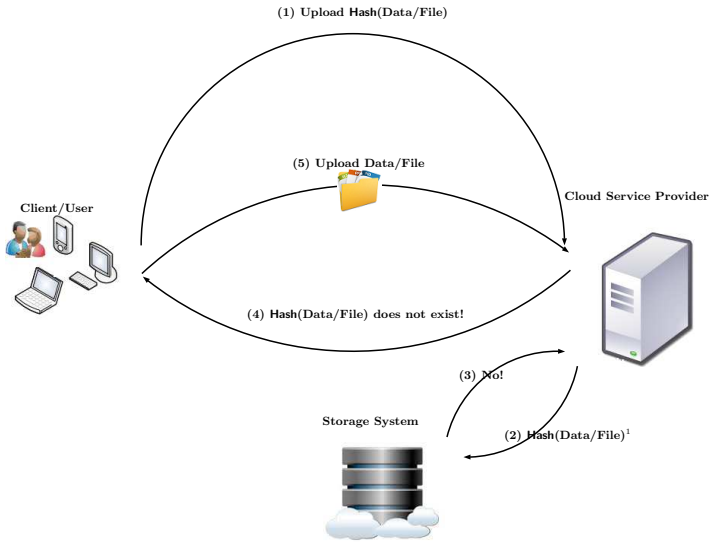


**Fig. 1.** Client-Side Deduplication. Note that if the answer is Yes! at Step (3) then a proof of ownership protocol is required to be executed between the user and the server.

Client-side deduplication is more interesting because both the clients and the CSPs do not want the files to be uploaded if they already exist in the server which also save high bandwidth costs [6, 7]. However, the client-side deduplication requires a client to prove the possession of the data without sending the actual data (e.g., the hash of the data instead of full plaintext where hash is a function taking a string as input and generates a unique compressed string). Namely, duplication control is performed on the client-side before users upload their files. More concretely, the client-side deduplication is jointly performed by the client and the server to check the existence of the uploading files in the server. In general, proof of ownership mechanisms are used to assure that the owner has the file without uploading it to the server. For example, a malicious user which does not own a file $F$ may obtain its hash value, and then may send to the cloud server to prove that the file was actually on the local storage. Therefore, there must be a secure cryptographic protocol for the proof of possession mechanism.

User data is already known to the CSPs in the existing deduplication solutions. What if the user does not want to reveal her data to the cloud providers while the CSPs still desire deduplication? The solution of this problem is known as *secure deduplication* and is not easy for the following reason: Assume that Alice and Bob have the same file $F$, and they encrypt $F$ using a conventional cryptographic scheme like AES under their own secret symmetric keys $K_A$ and $K_B$, respectively. Finally, they upload the ciphertexts $F_A = \mathsf{Enc}_{K_A}(F)$ and $F_B = \mathsf{Enc}_{K_B}(F)$ to the CSP, respectively. However, it is now computationally hard for the CSP to decide whether $F_A$ and $F_B$ are the same (because of the underlying encryption algorithm and $K_A \neq K_B$). Furthermore, even if the CSP could decide, it would be difficult for the CSP to store $F_A$ and $F_B$ in a short copy that allows both Alice and Bob to obtain the plaintext $F$ by decrypting it. Therefore, it should be ensured that the resulting ciphertexts are the same without giving any sensitive information about $F$ to the CSPs. In the case that semantically secure encryption is used, secure deduplication will also be impossible because neither Alice nor Bob can decide whether the two ciphertexts correspond to the same plaintext. Note also that offline dictionary attacks are applicable if the keys are derived from the $F$ (i.e., $K_A\ (= K_B) = \mathsf{H}(F)$ where $\mathsf{H}$ is a conventional hash function mapping inputs of arbitrary length to an output of a fixed length). Existing solutions in use unfortunately have either security and privacy issues (e.g., Dropbox, Amazon AWS, Google Drive, Bitcasa, Mozy, Memopal apply client-side deduplication [6, 8].).

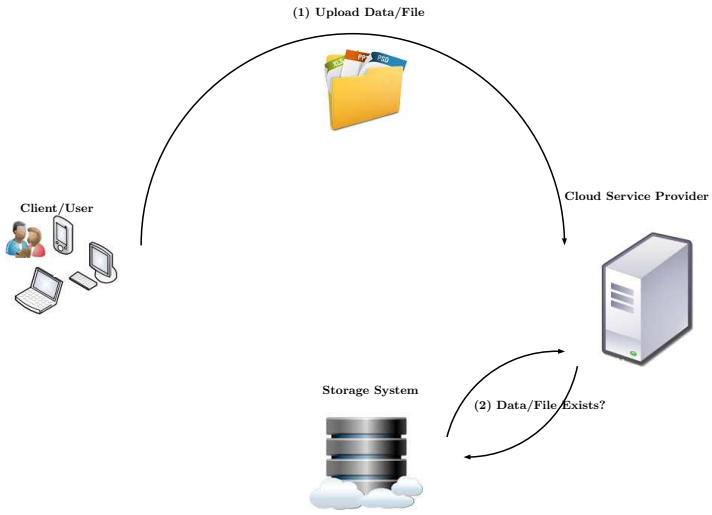


**Fig. 2.** Server-Side Deduplication

As a growing need, data integrity means that data should be correctly stored in the cloud server, and any modification without the consent of a user (e.g., if data is lost, becomes corrupt, modified or compromised) should easily be detected. More concretely, users store files to the cloud and want to guarantee that they are never changed or removed [9].

Integrity auditing is usually solved by using either message authentication codes (MACs) or digital signatures. Complete assurance of data integrity in an efficient manner is also not easy to handle. The main reason of scalability problem is that the computing resources is not run locally and the users cannot be sure that the CSP is correctly performing the computation. A direct solution may be first is to download complete files and check the integrity. However, this requires very high communication cost which resulting a non-scalable solution. Hence, to reduce the communication cost the work is generally delegated to a Third Party Auditor (TPA). Briefly, the TPA is assumed to be semi-honest and continuously performs the integrity auditing on behalf of the users by querying only a random subset of the small pieces of data [10–16]. The TPA does not need the client through the auditing process to assure that her/his data stored is indeed in the CSP.

### 1.1 Our Contributions

Privacy-preserving public auditing and secure deduplication are the two main lines of the research for secure storage of the outsourced data. However, these solutions may contradict each other because existing privacy-preserving auditing mechanisms require both the data and its signatures while deduplication requires not to upload the duplicated data again. The difficulty in solving this contradictory problem comes from the fact that once a user signs and uploads a file then how it can be possible for the next user to audit the same duplicated file without sending the actual file and her signatures. In this paper, we focus on this contradictory problem and propose an efficient solution for secure deduplication with a privacy-preserving public auditing mechanism.

To that extent, our proposal is the first scheme for secure storage providing both a secure deduplication mechanism and a privacy-preserving public auditing solution simultaneously. In particular, it maintains the storage and bandwidth savings of the client-side solution. Furthermore, it can also support both file and block level deduplication. Our main contributions are as follows:

– Our first contribution is to provide a efficient privacy-preserved public auditing mechanism anonymously while supporting secure deduplication, and to verify the signatures without compromising the privacy of the users. Hence, our mechanisms ensure that the CSP cannot obtain any information about the data and the TPA assures that the data is available and has not been changed. To ensure the privacy of users, we basically use anonymous public-key certificate architectures [17–21].
– Our second contribution is supporting client-side encryption while also assuring proof of ownership. To achieve this we use an Oblivious Pseudorandom function (OPRF) for encryption key generation.
– Our third contribution is to disallow CSP for denying the existence of the uploaded file. We utilize digital signature mechanism while uploading a new or deduplicated file to the CSP.

### 1.2 Roadmap

In Section 2, we summarize the related works by focusing on deduplication, privacy-preserving public auditing, and secure deduplication with privacy-preserving public auditing separately. We then give the necessary background and preliminaries in Section 3 that will be used throughout the manuscript. In Section 4, we propose our security model, and Section 5 presents our mechanism for secure deduplication with privacy-preserving public auditing. We analyze the security of our protocol in Section 6 and give its complexity in Section 7. Finally, Section 8 concludes the paper.

## 2 Related Work

### 2.1 Secure Deduplication

The concept of secure deduplication is interesting for both research and industrial community because of reducing high costs in cloud computing environment (e.g., Amazon S3, Dropbox, OwnCloud, TeamDrive, Box, OneDrive (formerly SkyDrive), Google Drive, DepSky, and SugarSync). The first solution is to apply *convergent encryption* mechanism which is designed by Douceur et al. [22]. The data here is encrypted using a symmetric encryption scheme with a key which is deterministically derived from the hash of the data content. The convergent encryption mechanism is actively used by commercial CSPs like Amazon S3, Dropbox, Google, and Bitcasa [23, 24]. Note that convergent encryption does not provide semantic security because of content-guessing attacks using the deterministic nature of the content hashing [2].

Several secure deduplication schemes have already been proposed [6, 25–27]. In [28], Harnik et al. showed some attacks in the case of client-side deduplication which can lead to data leakage. Later, the concept of proof of ownership has been introduced to prevent such attacks [29, 30]. However, none of these results guarantee the confidentiality against semi-honest or malicious CSPs. In [31], the authors discuss performance, back-up, and security and privacy issues of Mozy, Carbonite, Dropbox, and CrashPlan. [32] the authors specifically analyze the Dropbox client software with its communication protocol, and show the security and privacy issues within the Dropbox such as unauthorized access to the stored files.

Later, Bellare, Keelveedhi, and Ristenpart in [23] formalized the convergent encryption, which is defined as *message-locked encryption*. Message-locked encryption basically allows a user to encrypt the data under a symmetric key $sk$ where $sk = \mathsf{H}(M)$, $\mathsf{H}$ is a conventional hash function [23, 27, 33–35]. Therefore, a plaintext $M$ will lead to the same ciphertext $\mathsf{Enc}_{sk}(M)$.

[2] Semantic security in this context means that ciphertexts leak no information about the underlying plaintexts except the knowledge of their equalities.

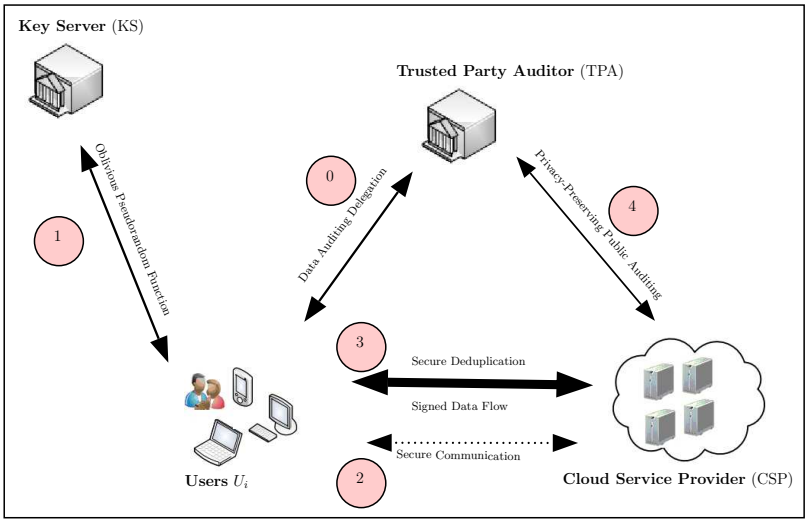


**Fig. 3.** A Privacy-Preserving Public Auditing Model with Secure Deduplication: (0) $i$-th user $U_i$ is delegating the auditing mechanism to a TPA, (1) a protocol for oblivious pseudorandom random function is run between the $i$-th user $U_i$ and the KS, (2) the $i$-th user $U_i$ securely communicates with the CSP, (3) the $i$-th user $U_i$ uploads a file to the CSP which supports both the secure deduplication and the privacy-preserving public auditing properties, (4) a privacy-preserving public auditing is run between the TPA and the CSP.

Furthermore, only $sk$ is stored and her encrypted file $M$ in the CSP in order to retrieve later and decrypt it. However, this mechanism of Bellare et al. does not provide semantic security and only satisfies security for unpredictable messages (since the secret key can be easily obtained for predictable files, and therefore, the knowledge of initialization vectors of the underlying symmetric encryption schemes become useless). Bellare et al. later presented another scheme called DupLESS which is a server-aided encryption mechanism for deduplicated storage and ensures semantic security [23].

In [36], the authors propose the first secure cross-user deduplication scheme that provides the client-side encryption. The scheme is based on using a PAKE (Password Authenticated Key Exchange) protocol and does not require any additional independent servers. Furthermore, in [37], the authors present a specific secure architecture enabling the encrypted cloud media center.

### 2.2 Privacy-preserving public auditing

There are basically two research direction of data integrity in the cloud setting: Provable Data Possession (PDP) and Proof of Retrievability (POR). PDP was first introduced by Ateniese et al. [38,39] for assuring that the CSPs indeed possess the files without retrieving or downloading the whole data. It is basically a challenge-response protocol which assures a client that her data stored on the cloud has not been modified without being detected. The protocol is realized by asking a random proper subset of blocks from the CSPs to prove that they indeed possess these blocks. The verifier TPA, which is assumed to be semi-honest, only checks a small portion of data. In [40], Wang et al. proposed proxy PDP in public clouds. It is also rather important to utilize stateless and semi-honest third party verifiers (i.e., auditors) [41–45]. In general, existing

mechanisms comprised of four main procedures: (1) signature generation by the client, (2) challenges by the TPA, (3) proof by the CSP, and (4) verification by the TPA.

Compared to PDP, POR (which is a challenge-response protocol) not only assures the cloud servers possess the target files, but also guarantees their full recovery [46–48].

### 2.3 The secure storage dilemma: Secure deduplication with privacy-preserving public auditing

Secure management and verification of integrity for the outsourced data contents are of importance in practice. Although they have been studied extensively in industrial and academic field a trivial combination of privacy-preserving public auditing and secure deduplication mechanisms does not simply solve both the secure deduplication and integrity auditing in an efficient manner (see Figure 3). This is because achieving storage efficiency contradicts with the deduplication of authentication tags (i.e., signatures). If the deduplicated data and its authentication tags were never sent to the cloud then how could the user audit data integrity on the cloud side?

We would like to first highlight that there are many solutions that deal with public auditing with deduplication [49, 50]. However, these solutions do not provide secure deduplication simultaneously because files are already visible to storage providers. In this respect, to the best of our knowledge, only the following papers consider both secure deduplication and public auditing notions simultaneously. Similarly, in [51], the authors claim that public auditing with deduplication. However, this scheme does not provide secure deduplication because the data is available in its plaintext form on the cloud-side.

To the best of our knowledge, the only existing schemes supporting both secure deduplication and public auditing are belong to [52] and [53]. In [52], the authors propose a variant of client-side deduplication mechanism where an honest *mediator* obtains the data in plaintext form and performs a block-level deduplication on users' data before they are sent to the cloud. Privacy-preserving public auditing is performed as usual via the trusted party auditor using Wang et al.'s scheme [42]. The main drawback of this scheme is to use of a mediator in the client's company environment. Therefore, not many deduplication would occur on a small size of enterprises instead of performing deduplication on the large scale CSPs, and hence, this solution looses the benefits of deduplication. .

Li et al. [53] proposed SecCloud+ which aims to solve both data integrity and deduplication by using the convergent encryption technique. The main drawback of this scheme is that the user also uploads his/her file to a TPA which is assumed to be trusted. Namely, this assumption is not realistic where the TPA may easily violate the privacy of the user. To the best our knowledge, disclosing the data to the TPA is also never modeled under the existing privacy-preserving public auditing schemes. Furthermore, this scheme has also several security and efficiency drawbacks. For example, their protocol is also too inefficient because of the use of the Type-1 pairing in their setting [54]. More concretely, for a secure implementation they need supersingular elliptic curves over large prime fields which significantly increase the complexity of their protocol (see [54] for details). On the other hand, uploading the file to an honest TPA a strong assumption because TPAs are always assumed to be semi-honest in all privacy-preserving existing auditing mechanisms to prevent privacy violations. Also, and more importantly, because the data is also sent to the TPA the bandwidth advantage of the client-side solutions is completely eliminated for the user. Their protocol is also too inefficient because of the same reasoning as above (i.e., they use Type-1 pairing [54]).

## 3 Preliminaries and Notation

### 3.1 Bilinear Maps for Auditing Schemes.

Bilinear maps are one of the best candidate for enabling auditing property. We follow the lines of [55, Chapter IX], [54] for the properties of pairings.

Let $(\mathbb{G}_1, +)$ and $(\mathbb{G}_2, +)$ be two additive cyclic groups of order $q$ with $\mathbb{G}_1 =< Q >$ and $\mathbb{G}_2 =< P >$, $(\mathbb{G}_3, \cdot)$ be a multiplicative cyclic group of order $q$, where $q$ is a prime number, 0 denotes the identity element of $\mathbb{G}_1, \mathbb{G}_2$, and 1 denotes the identity element of $\mathbb{G}_3$. Assume that Discrete Logarithm Problem ($\mathsf{DLP}$) is hard in both $\mathbb{G}_1$ and $\mathbb{G}_2$ (i.e., given a random $R \in \mathbb{G}_1$ (or $\in \mathbb{G}_2$), it is computationally infeasible to find an integer $x \in \mathbb{Z}$ such that $R = x \cdot Q$). A *bilinear map* is a map $e: \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_3$ satisfying the following properties:

- **Bilinearity:** For all $P_1, Q_1 \in \mathbb{G}_1, P_1', Q_1' \in \mathbb{G}_2$, $e$ is a group homomorphism in each component, i.e.
  1. $e(P_1 + Q_1, P_1') = e(P_1, P_1') \cdot e(Q_1, P_1')$,
  2. $e(P_1, P_1' + Q_1') = e(P_1, P_1') \cdot e(P_1, Q_1')$.
- **Non-degeneracy:** $e$ is non-degenerate in each component, i.e.
  1. For all $P \in \mathbb{G}_1$, $P \neq 0$, there is an element $Q \in \mathbb{G}_2$ such that $e(P, Q) \neq 1$,
  2. For all $Q \in \mathbb{G}_2$, $Q \neq 0$, there is an element $P \in \mathbb{G}_1$ such that $e(P, Q) \neq 1$.
- **Computability:** There exists an algorithm which computes the bilinear map $e$ efficiently.

Furthermore, the *Computational Diffie-Hellman* ($\mathsf{CDH}$) problem in a multiplicative group $\mathbb{G}$ with a generator $g$ is defined as follows: given only $g$, $g^x$, $g^y \in \mathbb{G}$ where $x, y \in \mathbb{Z}$, compute $g^{xy}$ without the knowledge of $x$ or $y$ [56]. We also assume that the $\mathsf{CDH}$ problem is computationally intractable (which will be used for the underlying privacy-preserving public auditing mechanism).

For the sake of easy notation, we use multiplicative notation for elements of $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_3$ in our protocol descriptions.

### 3.2 Symmetric Encryption & Hash Functions.

We denote a symmetric key encryption with

$$\mathcal{E}_{\mathsf{sym}} = \mathsf{SymEnc}_k(M)$$

and decryption with

$$M = \mathsf{SymDec}_k(\mathcal{E}_{\mathsf{sym}}),$$

where $k$ is a secret key and $M$ is a plaintext to be encrypted. We denote $\mathsf{H}_1(.)$ for the conventional collision resistant hashing process where

$$\mathsf{H}_1 : \{0,1\}^* \rightarrow \{0,1\}^{\ell},$$

where $\ell$ is a fixed length of output (e.g., if $\mathsf{H}_1$ is SHA-512 based, then $\ell = 512$). Denote $\mathsf{H}_2(.)$ for a secure hashing of an arbitrary binary string to a point on the elliptic curve group $\mathbb{G}_1$, i.e.,

$$\mathsf{H}_2 : \{0,1\}^* \rightarrow \mathbb{G}_1.$$

### 3.3 Asymmetric Encryption & Signature.

The underlying public key encryption scheme of our protocol must be semantically secure (some well-known algorithms are RSA-OAEP [57], Paillier [58], and ElGamal [59]). We utilize elliptic curve versions of ElGamal encryption scheme due to the underlying algorithmic suits which need to work compatibly.

- $(pk_{\mathcal{U}}, sk_{\mathcal{U}})$ denotes a public and private key pair of a user $\mathcal{U}$ (which is generated by a trusted third party (TTP)).
- $\mathsf{AsymEnc}_{pk_{\mathcal{U}}}(M)$ denotes an encryption of a message $M$ using the public key of $\mathcal{U}$ and $\mathsf{AsymDec}_{sk_{\mathcal{U}}}(C)$ denotes a decryption of a ciphertext using the private key of $\mathcal{U}$.
- We denote $\mathsf{Sign}_{sk_{\mathcal{U}}}(M)$ for the signature of a user $\mathcal{U}$ using its private key $sk_{\mathcal{U}}$.

Note that the functionality $\mathsf{Sign}$ is the traditional PKI signature scheme such as Digital Signature Standard (DSS), American National Standard (ANS) X9.31, and Public Key Cryptography Standard (PKCS) #1.

### 3.4 Anonymous Public-Key Certificates for Secure Deduplication and Privacy-Preserving Public Auditing.

Assume that a user $\mathcal{A}$ uploads a file $F$ to the $\mathsf{CSP}$ (in signed and encrypted form) and would like to get the services for both secure deduplication and public auditing. Assume also that another user $\mathcal{B}$ has also the same file $F$ and would like to upload to the $\mathsf{CSP}$. However, the user $\mathcal{B}$ should not send the file in encrypted or signed form, otherwise the client $\mathcal{B}$ and the $\mathsf{CSP}$ would not be able to benefit communication and/or storage advantages. Still, the user $\mathcal{B}$ should be able to delegate the auditing of the duplicated data without harming the privacy of the user $\mathcal{A}$. Namely, the $\mathsf{TPA}$ needs to verify the signatures without revealing the real identity of the user $\mathcal{A}$. Traditional PKI based architectures unfortunately cannot satisfy because of the inherent disclosure of the real identities.

We propose to use anonymous public-key certificates in our scheme to hide the real identities [17,18,20,21]. In an anonymous public-key certificate architecture, the certificate issuing protocol allows a user to apply to certificate authorities for generating anonymous certificates using a certificate already issued. Namely, pseudonyms will be in the anonymous certificate instead of the real identity of the users. Furthermore, in case of a malicious activity, an identity tracing protocol still enables a legal authority to trace the pseudonym in an anonymous certificate back to the real identity of the corresponding party. Note also that both Idemix [60] and uProve [61] offer complete anonymity even against to the certificate authorities. Furthermore, in case of a malicious user, their extensions can always extract the real identity of a user from an interaction with a relying party (by means of utilizing an external legal authority).

The Requests for Comments (RFC) document in [19] already defines a practical architecture to assure privacy of a user who requests and uses an X.509 certificate containing a pseudonym. This RFC document still provides the ability to link a certificate to the identity of the user who requested it.

We denote

$$\mathsf{AnonymSign}_{sk_{\mathcal{U}}}(M)$$

for the signature of an anonymous user $\mathcal{U}$ on a message $M$ using its private key $sk_{\mathcal{U}}$. Our proposed mechanism can easily utilize the anonymous public-key certificate architecture of the RFC in [19].

### 3.5 Oblivious Pseudorandom Functions (OPRF).

OPRF is a two-party protocol between a client C and a key server KS for privately computing a pseudorandom function $f_{k_{\mathsf{KS}}}(x)$ where $x$ is private key of C and $k_{\mathsf{KS}}$ is the private key of KS [62, 63]. At the end of the protocol, C obtains only $f_{k_{\mathsf{KS}}}(x)$ while KS learns no information. More concretely, a functionality for the OPRF can be realized as

$$\mathcal{F}_{\mathsf{OPRF}} : (k_{\mathsf{KS}}, x) \rightarrow (\perp, f_{k_{\mathsf{KS}}}(x))$$

for some pseudorandom function $f_{k_{\mathsf{KS}}}$. Chaum's *blind signatures* can be an example of OPRF [23, 33, 64].

## 4 Security Model

In this section, we aim at providing a security model for both privacy-preserving public auditing and secure deduplication.

### 4.1 Components

Our system involves five entities: A Certificate Authority (CA), a Key Server (KS), a CSP, a Trusted Party Auditor (TPA), and a user $U_i$. Their roles are described below:

- The CA is responsible for generating and issuing both traditional and anonymous public-key certificates. They are also responsible to trace the attackers back and identify their real identities (in case of a malicious activity). Note that this protocol is run only during the system setup.
- The KS and $U_i$ runs a two-party protocol OPRF. Note that this protocol is run only during the system setup.
- The CSP is a data storage owner which provides storage services for its clients to create, store, update and request data for retrievability. We assume that the CSP has a large storage, computation resources, and also provides host application services by the underlying virtual infrastructures. The users utilize these services in order to store and update their data in the CSP. Furthermore, the CSP supports secure client-side deduplication and privacy-preserving public auditing.
- The user ($U_i$) is a client of the CSP and has large amount of data to be stored. Moreover, $U_i$ utilizes the CSP's resources to store, retrieve, and share data with multiple users. $U_i$ uploads her encrypted data to the CSP, and deletes the original one from the local storage. As for auditing property, we assume that $U_i$ delegates the integrity property to the TPA to verify that her data indeed stays in the CSP correctly.
- The TPA is assumed to be semi-honest. It can check the cloud storage reliability and validity of cloud storage. Furthermore, the TPA is assumed to have a secure connection with the CSP in order to check the integrity and validity of users' data. The TPA also assists users and the CSP to perform the duplication control and support secure deduplication.
- A key server KS is assumed to be semi-honest. An oblivious pseudorandom function is run between the KS and a user to realize a $\mathcal{F}_{\mathsf{OPRF}}$.

### 4.2 Threat Model

First of all, TPA is assumed to be semi-honest, and all other components do not trust each other and are assumed to be malicious. Furthermore, we assume that the CA is trustworthy and do

not collude with other components. The security of the proposed system is proven based on the CDH assumption in the random oracle model [65].

For designing a secure client-side deduplication scheme, we consider the following adversary types: (1) Malicious outside adversary, (2) Malicious insider adversary CSP, (3) Malicious cloud user adversary, and (4) Semi-honest adversary TPA.

- **Malicious outside adversary.** All the communication channels are assumed to be secure to maintain the confidentiality and integrity of the data being transferred (the communication channel between the cloud storage provider and the client can be secured by TLS (which is already an industry standard) while uploading or downloading files.). The outside adversary is also assumed to possess its hash value for later to fool the CSP and to pass the proof of ownership of the data.
- **Malicious insider adversary CSP.** We assume that the CSP is malicious which may arbitrarily deviate from the protocol and intends to modify or delete the users' encrypted data contents in order to gain extra information over the outsourced sensitive data. Moreover, the CSP may also try to build links between user profiles and accessed data files. We also assume that the CSP can always detect and learn the existence of a deduplication. Furthermore, since the CSPs already knows their clients (i.e., their real identities) they know which users share the duplicated data but do not disclose their real identities. However, they cannot learn any information about the duplicated data.
- **Malicious cloud user adversary.** The goal of a malicious user is to convince the CSP that he is indeed a legitimate data owner. We assume that the malicious adversary has the short hash of the data. This information can then be used as an input to be able to fool the CSP and pass the proof of ownership successfully.
- **Semi-honest adversary TPA.** The TPA continuously executes the auditing scheme in a correct manner to assess the reliability of CSP on behalf of the users, but also tries to obtain some information about users' data.

We also assume that CSP and TPA are deployed by different organizations and they do not collude each other.

### 4.3 Security Goals

The proposed secure deduplication protocol with privacy-preserving public auditing model satisfies the following security objectives.

- **Public verifiability:** TPA can verify the correctness and the availability of the outsourced data without retrieving the entire data and without having online connections with the users.
- **Storage correctness:** A CSP can pass TPA's verification only if it indeed keeps the user's data.
- **Privacy:** No information about the outsourced data is leaked to TPA during the auditing process. Also, data itself is never leaked to the CSP except the information of duplication. Users' identities are also not disclosed when there is duplication.
- **Secure deduplication:** Secure deduplication is still possible on the CSP and does not leak any information to the CSP except the knowledge of duplication.

## 5 Our Secure Deduplication with Public Auditing Scheme

### 5.1 Setup

The following setup is based on Wang et al. [42] as we follow their secure auditing mechanism as a subalgorithm. The main difference is that the encrypted messages will be audited instead

of its plaintext form and the underlying signature will be anonymous and they are verified as described-above. We note that the scheme of Wang et al. [42] can adopt the use of the well-known optimization Merkle trees for proof of retrievability of very large files in an efficient manner (i.e., without having to read the entire file) [66].

First of all, a user $U_i$ runs a key generation protocol and obtains public and private inputs as follows. The signing key pair $(spk_i, ssk_i)$ is generated, and then the public key parameters of the user $U_i$ is computed, denoted by

$$pk_i = \{spk_i, g_i, h_i, v_i, e(v_i, h_i), q\}$$

where $x_i \in \mathbb{Z}_p$, $v_i \in \mathbb{G}_1$, $h_i = g_i^{x_i}$, $q$ is the order of $\mathbb{G}_1, \mathbb{G}_2 (q = |\mathbb{G}_1| = |\mathbb{G}_2|)$ and the private key of the user $U_i$ is denoted by

$$sk_i = \{x_i, ssk_i\}.$$

Note that the public and private key pairs $(pk_i, sk_i)$ of the $i$-th user is issued through an anonymous certificate protocol as described above (i.e., pseudonym is included in the certificate instead of the real identity). Furthermore, $(pk_{\mathsf{CSP}}, sk_{\mathsf{CSP}})$ denotes the public and private key pair of the CSP. Also, the CSP has the public keys $g_i, h_i$ for all users $U_i$s. Similarly, the user $U_i$ has also the public key of CSP denoted as $pk_{\mathsf{CSP}}$ and has also the his file

$$F = F_1 \| \cdots \| F_t \text{ where } F_i \in \{0,1\}, i = 1, \cdots, t.$$

$K_{U_i}$ also denotes the (private) symmetric key of the user $U_i$.

Furthermore, an oblivious pseudorandom function (OPRF) protocol is run between a key server (KS) and a user in order to provide semantic security during encryption key generation where $k_{\mathsf{KS}}$ is the private key belongs to the KS (e.g., blind signature protocols).

Finally, the security parameter $k$ is chosen large enough in order to make the brute force or other attacks computationally impossible on finding the pre-image message $m$ ($m \in \{0,1\}^k$) given $y$ and $k$ such that $y = \mathsf{H}_1(m)$.

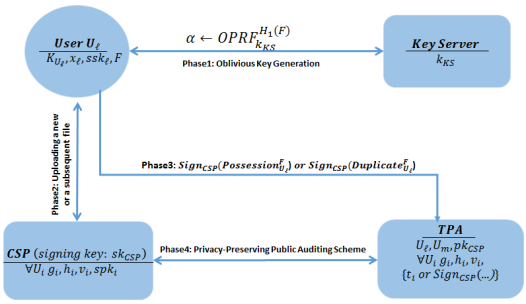


**Fig. 4.** Overview of Our Secure Deduplication Mechanism with Public Auditing

### 5.2 Overview of Our Mechanism

Before we present the detailed description of our system, we refer to Figure 4 to illustrate the main architecture. Roughly speaking, the user $U_\ell$ first interacts with a key server KS in order to obtain a secure and blinded key $\alpha = \mathsf{OPRF}_{k_{\mathsf{KS}}}^{\mathsf{H}_1(F)}$. Next, $U_\ell$ communicates with the CSP in order for uploading a new file. At this stage, $U_\ell$ first computes a secure hash value of the encryption of $F$ (i.e., $h = H_1(SymEnc_\alpha(F))$) and interrogates the CSP with $h$. If there is no duplication on the server, the protocol depicted in Figure 5 will be run; otherwise, the protocol

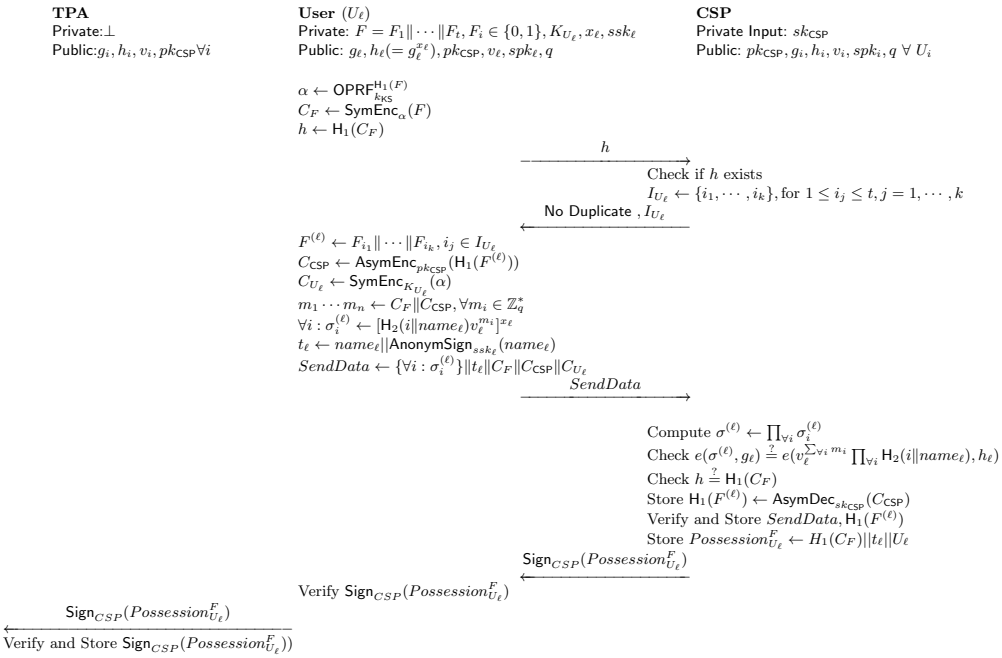


**Fig. 5.** The Scheme of Uploading a New File

of Figure 6 will be completed. Later, $U_\ell$ can easily retrieve his/her file(s) by running the protocol of Figure 7. Finally, using the protocol summarized on Figure 8, TPA would audit the uploaded new/subsequent file on the behalf of the user $U_\ell$ without further help of $U_\ell$.

### 5.3 Secure Data Upload Satisfying Deduplication with Public Auditing Scheme

The uploading algorithm is defined as follows (see Figure 5):

1. The $\ell$-th user $U_\ell$ runs an OPRF protocol to compute $\alpha = \mathsf{OPRF}_{k_{\mathsf{KS}}}^{\mathsf{H}_1(F)}$. Next, $U_\ell$ computes $C_F \leftarrow \mathsf{SymEnc}_\alpha(F)$, $h \leftarrow \mathsf{H}_1(C_F)$, and finally sends $h$ to CSP.
2. The CSP checks whether $h$ exists in the database. If it does not exist, it chooses a random set $I_{U_\ell} \leftarrow \{i_1, \cdots, i_k\}$ where $1 \le i_j \le t$ (bit length of $F$), $j = 1, \cdots, k$, and returns No Duplicate, $I_{U_\ell}$ to the $U_\ell$. Note that CSP will use this index set as challenges in Algorithm 6 when another user claims that she/he owns this file.
3. The $U_\ell$ first computes $F^{(\ell)} \leftarrow F_{i_1} \| \cdots \| F_{i_k}$ for $i_j \in I_{U_\ell}$. Note that $F^{(\ell)}$ is computed in order to ensure proof of retrievability (i.e., the data is available). Next, it encrypts it as $C_{\mathsf{CSP}} \leftarrow \mathsf{AsymEnc}_{pk_{\mathsf{CSP}}}(\mathsf{H}_1(F^{(\ell)}))$ and $C_{U_\ell} \leftarrow \mathsf{SymEnc}_{K_{U_\ell}}(\alpha)$. Let $m_1 \cdots m_n \leftarrow C_F \| C_{\mathsf{CSP}}$ where $\forall m_i \in \mathbb{Z}_q^*$. For $i = 1, \cdots, n$, it computes $\sigma_i^{(\ell)} \leftarrow \mathsf{AnonymSign}_{x_\ell}(\mathsf{H}_2(i \| \, name_\ell) \, v_\ell^{m_i}) = [\mathsf{H}_2(i \| name_\ell) v_\ell^{m_i}]^{x_\ell}$. Next, $U_\ell$ computes the file tag for $t_\ell \leftarrow name_\ell \, || \, \mathsf{AnonymSign}_{ssk_\ell}(name_\ell)$ to ensure the integrity of the unique file identifier $name_\ell$. Finally, it prepares $SendData \leftarrow \{\forall i : \sigma_i^{(\ell)}\} \| t_\ell \| C_F \| C_{\mathsf{CSP}} \| C_{K_\ell}$ and returns $SendData$ to the CSP.
4. The CSP first computes $\sigma^{(\ell)} \leftarrow \prod_{\forall i} \sigma_i^{(\ell)}$ and checks the signatures by $e(\sigma^{(\ell)}, g_\ell) \stackrel{?}{=} e(v_\ell^{\sum_{\forall i} m_i} \prod_{\forall i} \mathsf{H}_2(i \| name_\ell)$ Next, it also checks $h \stackrel{?}{=} \mathsf{H}_1(C_F)$. If they are correct, it decrypts $C_{\mathsf{CSP}}$ by computing $\mathsf{AsymDec}_{sk_{\mathsf{CSP}}}(C_{\mathsf{CSP}})$ and stores $\mathsf{H}_1(F^{(\ell)})$. It verifies and stores $SendData, \mathsf{H}_1(F^{(\ell)})$. Finally, CSP prepares $Possession_{U_\ell}^F \leftarrow H_1(C_F) || t_\ell || U_\ell$ and returns $Sign_{CSP}(Possession_{U_\ell}^F)$ to $U_\ell$.
5. The $U_\ell$ verifies the signature $Sign_{CSP}(Possession_{U_\ell}^F)$ and forwards it to TPA.
6. TPA verifies and stores $Sign_{CSP}(Possession_{U_\ell}^F)$.

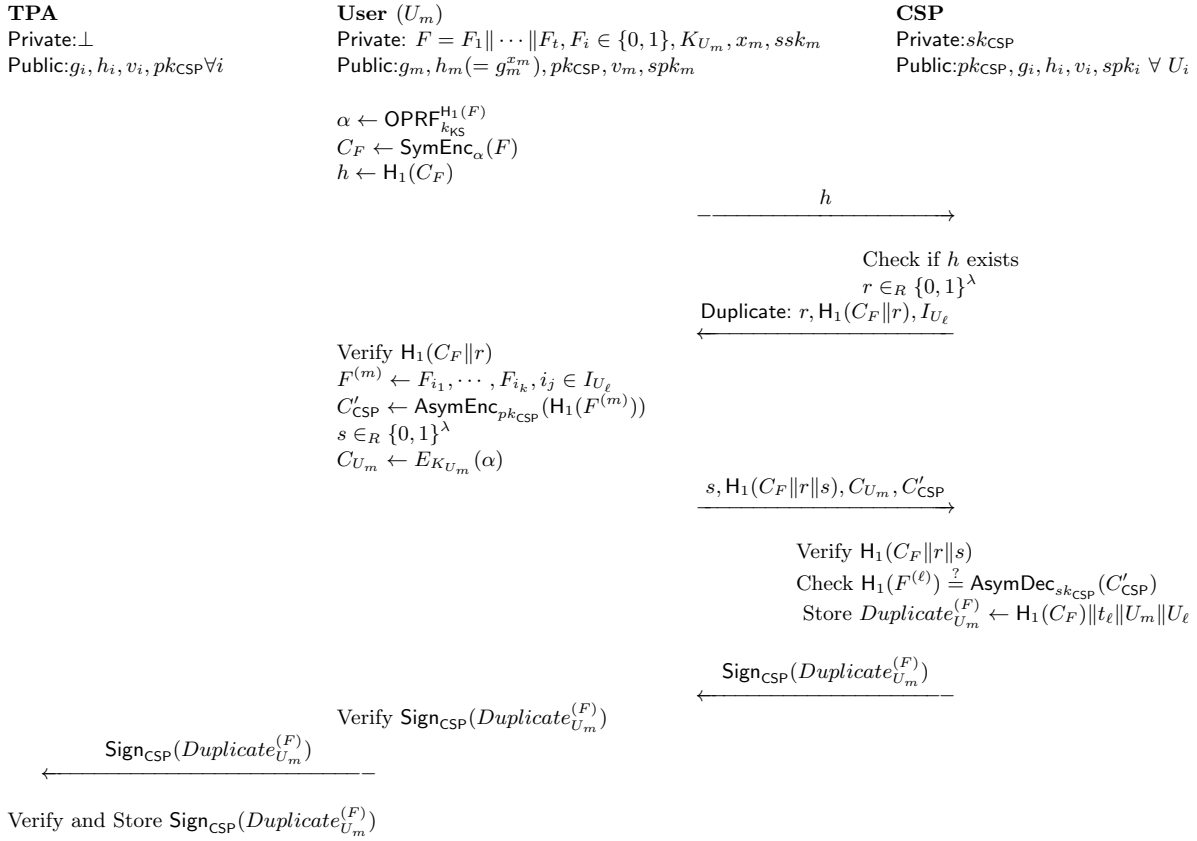


**Fig. 6.** The Scheme of Subsequent Upload

*Remark 1.* Note that, in the second round of the protocol, the cloud server searches the hash value $h$ in the database and decides whether the encrypted file $C_F$ is in the database. The security of this solution is based on collision-resistant hash functions $\mathsf{H}_1(.)$. However, if malicious users somehow obtain or compute collisions in practice then only looking for $h$ become not sufficient. Namely, if the malicious users somehow has the short hashes $h$ then they could fool the CSP that they indeed have the encrypted file $C_F$s. Therefore, in the third round of our protocol, we additionally request $F^{(\ell)}$ to eliminate these threats and to ensure the proof of correctness and retrievability.

### 5.4 The Scheme of Subsequent Upload

The following algorithm is executed if there is duplicate file on the CSP (see Figure 6):

1. The $m$-th user $U_m$ runs an OPRF protocol to compute $\alpha \leftarrow \mathsf{OPRF}_{k_{\mathsf{KS}}}^{\mathsf{H}_1(F)}$. Next, $U_m$ computes $C_F \leftarrow \mathsf{SymEnc}_\alpha(F)$, $h \leftarrow \mathsf{H}_1(C_F)$, and finally sends $h$ to CSP.
2. The CSP checks whether $h$ exists in the database. If it exists, it chooses a random number $r \in_R \{0,1\}^\lambda$ and returns Duplicate: $r, \mathsf{H}_1(C_F \| r), I_{U_\ell}$. The message pair $r, \mathsf{H}_1(C_F \| r)$ is sent to prove the possession of $C_F$ to the $U_m$. $I_{U_\ell}$ index set has been already created in the first protocol between the first owner of file $F$ (When no duplication occurs.). The same set $I_{U_\ell}$ is sent to prove that the current user indeed possess the file $F$.
3. The $U_m$ first verifies $\mathsf{H}_1(C_F \| r)$ to ensure that the CSP indeed possess the $C_F$. Next, it computes $F^{(m)} \leftarrow F_{i_1}, \cdots, F_{i_k}$, $C'_{\mathsf{CSP}} \leftarrow \mathsf{AsymEnc}_{pk_{\mathsf{CSP}}}(\mathsf{H}_1(F^{(m)}))$. As described in Remark 1 for the protocol in Figure 5, the goal of sending $F^{(m)}$ is to guarantee the proof of retrievability of

the encrypted file $C_F$ (i.e., data is indeed on the client-side). Next, it chooses random number $s \in_R \{0,1\}^\lambda$ and computes $C_{U_m} \leftarrow E_{K_{U_m}}(\alpha)$. Finally, it sends $s, \mathsf{H}_1(C_F \| r \| s), C_{U_m}, C'_{\mathsf{CSP}}$. Similarly as in the previous round, $s$ and $\mathsf{H}_1(C_F \| r \| s)$ are sent to prove that the $U_m$ indeed possesses $C_F$. Also, $C'_{\mathsf{CSP}}$ proves the possession of $F$.

4. The $\mathsf{CSP}$ verifies $\mathsf{H}_1(C_F \| r \| s)$ and checks $\mathsf{H}_1(F^{(\ell)}) \stackrel{?}{=} \mathsf{AsymDec}_{sk_{\mathsf{CSP}}}(C'_{\mathsf{CSP}})$, and stores $Duplicate^{(F)}_{U_m} \leftarrow \mathsf{H}_1(C_F) \parallel t_\ell \parallel U_m \parallel U_\ell$. Finally, it returns $\mathsf{Sign}_{\mathsf{CSP}}(Duplicate^{(F)}_{U_m})$ to $U_m$. This signature assures the user that the data is linked with the $U_m$. Also, it prevents a malicious $\mathsf{CSP}$ (the signer) from denying the signing. Note that the users' pseudonyms are used instead of their real identities.
5. The $U_m$ verifies the signature $\mathsf{Sign}_{\mathsf{CSP}}(Duplicate^{(F)}_{U_m})$ and forwards it to $\mathsf{TPA}$.
6. The $\mathsf{TPA}$ verifies and stores $\mathsf{Sign}_{\mathsf{CSP}}(Duplicate^{(F)}_{U_m})$. Note that $\mathsf{TPA}$ will process the auditing utilizing the signatures of $U_\ell$ on behalf of $U_m$. Because the signatures are anonymous no one will know any information about the real identity of $U_\ell$.

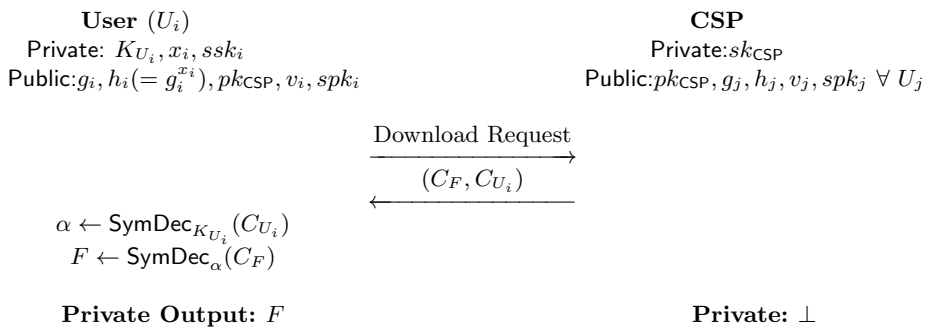


**Fig. 7.** The Scheme of Data Retrieval

*Remark 2.* We highlight that one needs to store the $\mathsf{CSP}$'s signatures $\mathsf{Sign}_{\mathsf{CSP}}(Duplicate^{(F)}_{U_m})$ for later auditing processing since otherwise one cannot later convince $\mathsf{CSP}$ that the deduplication was already occurred between the specific anonymous users. Therefore, in our proposal, the $\mathsf{CSP}$ sends these signatures to the users which forwards them to the $\mathsf{TPA}$. We highlight that, unlike the existing auditing mechanisms, the $\mathsf{TPA}$s in our proposal are not stateless (memorysless). In case of utilizing a stateless $\mathsf{TPA}$, a local copy of these signatures should be copied on the user side (instead of forwarding to the $\mathsf{TPA}$). However, in that case, the user must always forward these signatures to the $\mathsf{TPA}$ before it can start the auditing process.

**Charging each client in the case of deduplication.** In practice, cloud providers typically charge its clients based on the amount of data they stored. We would like to highlight that anonymous certificates are only used for duplicated users and cloud storage providers can identify their clients in our mechanism since there is already an authentication protocol between each client and the storage provider. Therefore, it is easy for a cloud provider to charge each client if there is already a deduplication.

### 5.5 The Scheme of Data Retrieval

Our data retrieval protocol is as follows (see Figure 7): the user $U_m$ sends a download request from $\mathsf{CSP}$ and receives $(C_F, C_{U_m})$. Next, $U_m$ then computes the encryption key $\alpha \leftarrow \mathsf{SymDec}_{K_{U_m}}(C_{U_m})$ and obtains the file by computing $F \leftarrow \mathsf{SymDec}_{\alpha}(C_F)$.

### 5.6 Privacy-Preserving Public Auditing Scheme

The privacy of our auditing mechanism follows from the Wang et al.'s scheme [42] (see Figure 8). Let $F$ be the file for auditing and $U_\ell$ is the first owner of the file. Let $U_m$ is the target user for auditing and $U_m$ also posses $F_m$.

In order for auditing the file of the user $U_m$, $\mathsf{TPA}$ gets either $sign_1 = \mathsf{Sign}_{\mathsf{CSP}}(Possession_{U_m}^{(F)})$ or $sign_2 = \mathsf{Sign}_{\mathsf{CSP}}(Duplicate_{U_m}^{(F)})$ where $Possesion_{U_m}^{(F)} = \mathsf{H}_1(C_F)\|t_\ell\|U_m$ and $Duplicate_{U_m}^{(F)} = \mathsf{H}_1(C_F)\|t_\ell\|U_m\|U_\ell$. Note that if $m \neq \ell$ then $sign_1$ is null (there is no duplication or $U_m$ is the first owner of the file), otherwise $signature_2$ is null. Note that the existence of $sign_2$ ensures the $\mathsf{CSP}$ that there was a deduplication with the user $U_\ell$, and in this case, the $\mathsf{CSP}$ needs to send necessary auditing information of the $U_\ell$. Then, the $\mathsf{TPA}$ next picks a random proper subset of $J \subset I = \{1, \cdots, n\}$ and for all $j \in J$, it sends $(j, \omega_j)$ to the $\mathsf{CSP}$. It also sends either $\mathsf{Sign}_{\mathsf{CSP}}(Possession_{U_m}^{(F)})$ or $\mathsf{Sign}_{\mathsf{CSP}}(Duplicate_{U_m}^{(F)})$.

The $\mathsf{CSP}$ first verifies the signatures and then computes $\sigma^{(\ell)} \leftarrow \prod_{j\in J}(\sigma_j^{(\ell)})^{\omega_j}$, $\mu' \leftarrow \sum_{j\in J}\omega_j \cdot m_j$, and for random $r \in_R \mathbb{Z}_p$ it computes $R \leftarrow e(v_\ell, h_\ell)^r$. Next, the $\mathsf{CSP}$ computes $\gamma \leftarrow \mathsf{H}_2(R)$ and $\mu \leftarrow r + \gamma\mu'$ and sends $\sigma^{(\ell)}, \mu, R$ back to the $\mathsf{TPA}$. The $\mathsf{TPA}$ first computes $\gamma \leftarrow \mathsf{H}_2(R)$ and then verifies $R \cdot e((\sigma^{(\ell)})^\gamma, g_\ell) \stackrel{?}{=} e((\prod_{j\in J}\mathsf{H}_2(j||name_\ell)^{\omega_j})^\gamma \cdot v_\ell^\mu, h_\ell)$.

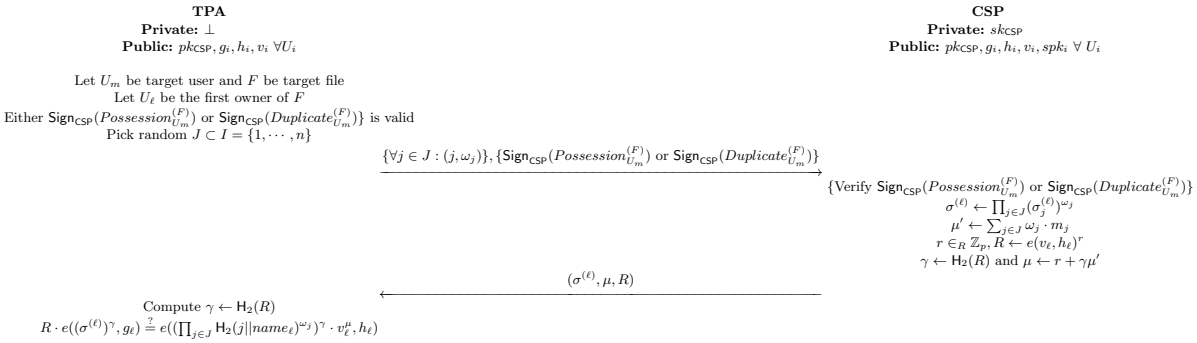


**Fig. 8.** Privacy-Preserving Public Auditing Scheme

## 6 Security Analysis

In this section, we prove that our proposed protocols in Section 5 satisfy the security model defined in Section 4.

In the following proof, we start by constructing a probabilistic polynomial-time simulator that produces a protocol transcript which is statistically indistinguishable from the one resulting from a real execution of the authentication protocol [67]. Namely, given a corrupted party (either a user $U_\ell$ or the $\mathsf{CSP}$), we construct an efficient simulator that has access to the public and private input of the corrupted party before the simulation is run and produces a view which is statistically indistinguishable from the view of that party interacting with the other honest party.

**Theorem 1.** *Assume that $\mathsf{OPRF}$ used is secure, $\mathsf{H}_1, \mathsf{H}_2$ are hash functions modeled as random oracles, and the $\mathsf{CDH}$ problem is hard. Then, our protocol in Figure 5 securely uploads a file $F$ in the presence of malicious adversaries.*

*Proof.* The security proof of the protocol is shown in the $\mathsf{OPRF}$-hybrid model, where the user and $\mathsf{SF}$ are assumed to have access to a trusted party computing the $\mathsf{OPRF}$ functionality following

the ideal model of our definition. Thus, the simulator will play the trusted party in the $\mathsf{OPRF}$, when simulating for the adversary.

We separately consider the different corruption cases (when no parties are corrupted, and when either one of the parties is corrupted). In the case that no parties are corrupted, the security reduces to the semi-honest case.

Assume that the user $U_\ell$ is corrupted: In this case, the simulator $\mathcal{A}$ has the $sk_{CSP}$ apart from the public parameters. First of all, the adversary $\mathcal{A}$ arbitrarily a set $I_{U_\ell}$. $\mathcal{A}$ then sends the $I_{U_\ell}$ to the user and obtains $SendData$. The simulator $\mathcal{A}$ then computes $\sigma^{(\ell)} \leftarrow \prod_{\forall i} \sigma_i^{(\ell)}$ and checks the signatures by $e(\sigma^{(\ell)}, g_\ell) \stackrel{?}{=} e(v_\ell^{\sum_{\forall i} m_i} \prod_{\forall i} \mathsf{H}_2(i \| name_\ell), h_\ell)$. Next, it also checks $h \stackrel{?}{=} \mathsf{H}_1(C_F)$. If they are correct, it simulates $\mathsf{AsymDec}_{sk_{\mathsf{CSP}}}(C_{\mathsf{CSP}})$ since it knows the $C_{\mathsf{CSP}}$, the encrypted message $\mathsf{AsymEnc}_{pk_{\mathsf{CSP}}}(C_{\mathsf{CSP}})$, and $pk_{\mathsf{CSP}}$.

Assume that the $\mathsf{CSP}$ is corrupted: In this case, the simulator $\mathcal{A}$ has the private values of the user $U_\ell$, i.e., $F$, $K_{U_\ell}$, $x_\ell$, $ssk_\ell$. The adversary $\mathcal{A}$ first arbitrarily chooses a file $F$ and calls the ideal functionality $\mathsf{OPRF}$ and learns $\alpha = \mathsf{OPRF}_{k_{\mathsf{KS}}}^{\mathsf{H}_1(F)}$. Next, $\mathcal{A}$ computes $C_F \leftarrow \mathsf{SymEnc}_\alpha(F)$, $h \leftarrow \mathsf{H}_1(C_F)$, and finally sends $h$ to $\mathsf{CSP}$. Upon receiving $\mathsf{No\ Duplicate}$ and $I_{U_\ell}$, the adversary $\mathcal{A}$ computes $F^{(\ell)} \leftarrow F_{i_1} \| \cdots \| F_{i_k}$ for $i_j \in I_{U_\ell}$. Next, it encrypts it as $C_{\mathsf{CSP}} \leftarrow \mathsf{AsymEnc}_{pk_{\mathsf{CSP}}}(\mathsf{H}_1(F^{(\ell)}))$ and $C_{U_\ell} \leftarrow \mathsf{SymEnc}_{K_{U_\ell}}(\alpha)$. Let $m_1 \cdots m_n \leftarrow C_F \| C_{\mathsf{CSP}}$ where $\forall m_i \in \mathbb{Z}_q^*$. For $i = 1, \cdots, n$, it computes $\sigma_i^{(\ell)} \leftarrow \mathsf{AnonymSign}_{x_\ell}(\mathsf{H}_2(i \| \ name_\ell)\ v_\ell^{m_i}) = [\mathsf{H}_2(i \| name_\ell) v_\ell^{m_i}]^{x_\ell}$. Next, $\mathcal{A}$ computes the file tag for $t_\ell \leftarrow name_\ell \ \| \ \mathsf{AnonymSign}_{ssk_\ell}$ to ensure the integrity of the unique file identifier $name_\ell$. Finally, it prepares $SendData \leftarrow \{\forall i : \sigma_i^{(\ell)}\} \| t_\ell \| C_F \| C_{\mathsf{CSP}} \| C_{K_\ell}$ and returns $SendData$ to the $\mathsf{CSP}$.

The transcript of the view of both parties is simulated, and it is consistent and statistically indistinguishable when interacting with the honest verifier.

| | **CSP** | | | **Client Side** | | | **TPA** | | |
|---|---|---|---|---|---|---|---|---|---|
| | **# Pairing** | **# EC Multiplication** | **# Asymmetric Enc/Sign** | **# Pairing** | **# EC Multiplication** | **# Asymmetric Enc/Sign** | **# Pairing** | **# EC Multiplication** | **# Asymmetric Enc/Sign** |
| The Scheme of Uploading a New File (Figure 5) | 2 | 1 | 1 Sign<br>1 AsymDec | 0 | $2n$ | 1 $\mathsf{OPRF}_{k_{\mathsf{KS}}}^{\mathsf{H}_1(F)}$<br>1 AsymEnc<br>1 AnonymSign | NA | NA | NA |
| The Scheme of Subsequent Upload (Figure 6) | 0 | 0 | 1 Sign<br>1 AsymDec | 0 | 0 | 1 $\mathsf{OPRF}_{k_{\mathsf{KS}}}^{\mathsf{H}_1(F)}$<br>1 AsymEnc | 0 | 0 | 0 |
| The Scheme of Data Retrieval (Figure 7) | 0 | 0 | 0 | 0 | 0 | 0 | NA | NA | NA |
| Privacy-Preserving Public Auditing Scheme (Figure 8) | 0 | $\|J\| + 1$ | 0 | NA | NA | NA | 2 | $\|J\| + 3$ | 0 |

**Table 1.** Computational Complexity of the Proposed Scheme. NA denotes "Not Applicable", $n$ denotes the number of blocks of the message $C_F || C_{\mathsf{CSP}}$, and $J \subset I = \{1, \cdots, n\}$. Note that, for the sake of easy notation, we use multiplicative notation for elements of $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_3$ in our protocol descriptions.

**Theorem 2.** *Assume that $\mathsf{H}_1$ is a hash function modeled as a random oracle and the $\mathsf{CDH}$ problem is hard. Then, our protocol for the subsequent upload in Figure 6 is secure against malicious users (uploaders) and malicius outside attackers.*

*Proof.* We provide a formal proof via the formal verification tool Scyther [68, 69] to ensure the proof of possession of $C_f$ and that a malicious user cannot get any information during the secure

deduplication part of the protocol in Figure 6. Before we give a formal proof, we would like to briefly sketch the idea on the proof: the objective of the malicious uploader to compromise the confidentiality of data uploaded by legitimate users. In particular, the goal of an adversary is to convince the CSP that it possesses a pre-stored data. The proof of possession and the proof of retrievability are given by the first three rounds of the protocol. Namely, proving the knowledge of $F^{(m)}$ indeed guarantees that the file $F$ is available on the client-side. Informally speaking, in the first step, an adversary $U_{adv}$ obtains the hash of the encrypted data. $U_{adv}$ then sends the hash of encrypted data which only allows him to learn whether there is deduplication. If there is a deduplication, the CSP challenges the adversary to assure the possession of data via $r, \mathsf{H}_1(C_F||r), I_{U_i}$ for some $i$. Note that because of the underlying secure hash function (in the random oracle model) the adversary cannot upload a fake data (which compromises the integrity of data) and it does not give any sensitive information to the adversary. The adversary has to return $\mathsf{H}_1(C_F||r||s), C_{U_{adv}}$ which is only possible if the adversary possess the data $F$. The CSP can only be ensured if the verifications are passed successfully. Hence, the adversary cannot link himself to a legitimate data without possessing it.

The formal proof via a formal verification tool Scyther [68, 69] is given in Appendix A. The modeling of this part is given in a language called is Security Protocol Description Language (SPDL) in Appendix A including the codes of the validation of the protocol specifications. In this way, we show that no malicious adversary can get a valuable information during the deduplication part of the protocol.

If the proof of possession of the ciphertext $C_f$ is guaranteed then the user must ensure the knowledge of the message $F^{(m)}$ by checking $\mathsf{H}_1(F^{(\ell)}) \stackrel{?}{=} \mathsf{AsymDec}_{sk_{\mathsf{CSP}}}(C'_{\mathsf{CSP}})$. Furthermore, the signature $\mathsf{Sign}_{\mathsf{CSP}}$ $(Duplicate^{(F)}_{U_m})$ links anonymously the $U_\ell$ with the $U_m$. Furthermore, the CSP returns $\mathsf{Sign}_{\mathsf{CSP}}(Duplicate^{(F)}_{U_m})$ to the $U_m$ which forwards it to the TPA. Because the signatures are anonymous no one will know any information about the real identity of $U_\ell$. Hence, the proposed protocol is secure against malicious adversaries.

**Theorem 3.** *Our protocol for the data retrieval in Figure 7 is secure against malicious users (uploaders) and malicious outside attackers.*

*Proof.* It is trivial to show that the data retrieval protocol does not lead to a security breach because the only value that the CSP sends are the encryptions $C_F$ and $C_{U_i}$. That is, since the adversary does not know the symmetric key of the encryptions it cannot get any information about the data.

| | **Efficient Realizability (in terms of Deduplication and Auditing)** | **Privacy-Preserving Public Auditing** |
|---|---|---|
| Alkhojandi and Miri [52] | ✗ (Loss of deduplication advantages) | ✗ |
| Li et al. [53] | ✗ (Loss of bandwidth advantages) | ✗ (achieves weak privacy since the file is uploaded to the TPA) |
| This paper | ✓ | ✓ |

**Table 2.** Comparison with the Existing Schemes.

**Theorem 4.** *Assume that $\mathsf{H}_2$ is a hash function modeled as a random oracle and the CDH problem is hard. Then, our protocol for the privacy-preserving public auditing in Figure 8 is secure against semi-honest TPAs and malicious CSPs.*

*Proof.* First of all, the TPA obtains the tags $t_\ell$ and verifies their corresponding signatures. If they are correct, the TPA next picks a random proper subset of $J \subset I = \{1, \cdots, n\}$ and for all $j \in J$, it sends $(j, \omega_j)$ to the CSP to start the auditing process. The security, the privacy-preserving public verifiability, and the storage correctness against corrupted TPAs and CSPs basically follow the underlying auditing mechanisms of Wang et al. [42]. Note that the proof of Wang et al. only ensures the integrity of the data and do not hide the identity of the users.

In our scheme, there are two cases.

- $U_m$ is the first owner of the file and TPA would have the signature of possession $\mathsf{Sign}_{\mathsf{CSP}}(Possession_{U_m}^{(F)})$ where $Possession_{U_m}^{(F)} = \mathsf{H}_1(C_F) \parallel t_\ell \parallel U_m$. This signature prevents a malicious CSP (the signer) from denying the signing (i.e., assures *non-repudiation*). On the other hand, they prevent malicious users from claiming false possession.
- $U_m$ is owner of a deduplicated file and TPA would have the signature of duplication $\mathsf{Sign}_{\mathsf{CSP}}(Duplicate_{U_m}^{(F)})$ where $Duplicate_{U_m}^{(F)} = \mathsf{H}_1(C_F) \parallel t_\ell \parallel U_m \parallel U_\ell$. Note that these signatures ensure that there is a deduplication with the user $U_\ell$. On the one hand, the signatures $\mathsf{Sign}_{\mathsf{CSP}}(Duplicate_{U_m}^{(F)})$ assures the user $U_m$ that her data is linked to the user $U_\ell$, and therefore, it prevents a malicious CSP (the signer) from denying the signing (i.e., assures *non-repudiation*). On the other hand, they prevent malicious users from claiming false duplications.

Furthermore, the privacy of users during the auditing process is guaranteed by the underlying anonymous public-key certificates, i.e., users' pseudonyms are used in the signatures instead of their real identities. In the case of a malicious activity, the CA can also easily trace the malicious users back and can detect their real identities.

Hence, the modified version of the auditing scheme provides both secure deduplication privacy of users simultaneously.

## 7 Complexity Analysis

In Table 1, we provide the number of pairings, Elliptic Curve (EC) multiplications, and Asymmetric operations (in terms of public key encryption and signatures) of our system. The comparison has been given for the user, the CSP, and the TPA separately.

Furthermore, in Table 2, we compare our protocol with the only existing protocols of [52] and [53] which aim to satisfy public auditing and secure deduplication simultaneously. Note that the main drawback of [52] is that deduplication is performed by an mediator within the enterprise. Therefore, not many deduplication would occur on a small size of enterprises instead of performing deduplication on the large CSPs.

On the other hand, the main drawback of [53] is that the file is uploaded to the semi-honest TPA in plaintext form. Therefore, privacy will not be achieved against malicious TPAs. And more importantly, client-side deduplication solution of [53] would not eliminate the bandwidth advantage for a user. We also note that the protocol of [53] is also too inefficient because for a secure implementation one needs supersingular elliptic curves over large prime fields (see for further details [54]).

## 8 Conclusion

In this paper, we proposed a solution for the dilemma where the CSPs are willing to use deduplication to save communication and storage costs while the users are willing to encrypt their data before uploading and also to ensure that their data has never been modified without being detected. In this respect, we propose an efficient solution in order to fulfill privacy-preserving data

integrity and secure duplication simultaneously. We prove the security of the proposed scheme in the random oracle model, assuming that the CDH problem is hard. We leave open the question of constructing efficient cryptographic protocols that do not need certificate authorities or anonymous certificates.

***Acknowledgments.*** This work is supported by a grant from Ministry of Development of Turkey provided to the Cloud Computing and Big Data Research Lab Project (the project id: 2014K121030). The authors would like to thank Osmanbey Uzunkol, Devrim Ünal, Ziya Alper Genç, and anonymous referees for their valuable comments and suggestions.

## References

1. Sean Quinlan and Sean Dorward. Venti: A new approach to archival storage. In *Proceedings of the Conference on File and Storage Technologies*, FAST '02, pages 89–101, Berkeley, CA, USA, 2002. USENIX Association.
2. John R. Douceur, Atul Adya, William J. Bolosky, Dan Simon, and Marvin Theimer. Reclaiming space from duplicate files in a serverless distributed file system. In *Proceedings of the 22 Nd International Conference on Distributed Computing Systems (ICDCS'02)*, ICDCS '02, Washington, DC, USA, 2002. IEEE Computer Society.
3. Alex Osuna, Eva Balogh, Alexandre Ramos Galante de Carvalho, Rucel F. Javier, and Zohar Mann. Implementing ibm storage data deduplication solutions. 2011.
4. Fatema Rashid. *Secure Data Deduplication in Cloud Environments*. PhD thesis, Ryerson University, Toronto, Ontario, Canada, 2015.
5. Jesus Diaz Vico. Study of the security in cloud storage services: Analysis of dropbox and mega, 2016 (Accessed June 2016). `https://www.incibe.es/CERT_en/publications/Studies/`.
6. Danny Harnik, Benny Pinkas, and Alexandra Shulman-Peleg. Side channels in cloud services: Deduplication in cloud storage. *IEEE Security and Privacy*, 8(6):40–47, November 2010.
7. Alex Osuna, Eva Balogh, Alexandre Ramos Galante de Carvalho, Rucel F. Javier, and Zohar Mann. *Implementing IBM Storage Data Deduplication Solutions*, pages 1–322. march 2011.
8. Kate Miller. Cloud deduplication, on-demand: Storreduce, an apn technology partner. 2015 (Accessed July 2016).
9. Frederik Armknecht, Jens-Matthias Bohli, Ghassan O. Karame, Zongren Liu, and Christian A. Reuter. Outsourced proofs of retrievability. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 831–843, New York, NY, USA, 2014. ACM.
10. Richard Chow, Philippe Golle, Markus Jakobsson, Elaine Shi, Jessica Staddon, Ryusuke Masuoka, and Jesus Molina. Controlling data in the cloud: Outsourcing computation without outsourcing control. In *Proceedings of the 2009 ACM Workshop on Cloud Computing Security*, CCSW '09, pages 85–90, New York, NY, USA, 2009. ACM.
11. Guangwei Xu, Chunlin Chen, Hongya Wang, Zhuping Zang, Mugen Pang, and Ping Jiang. Two-level verification of data integrity for data storage in cloud computing. In Gang Shen and Xiong Huang, editors, *Advanced Research on Electronic Commerce, Web Application, and Communication*, volume 143 of *Communications in Computer and Information Science*, pages 439–445. Springer Berlin Heidelberg, 2011.
12. Wenjun Luo and Guojing Bai. Ensuring the data integrity in cloud data storage. In *Cloud Computing and Intelligence Systems (CCIS), 2011 IEEE International Conference on*, pages 240–243, Sept 2011.
13. Hovav Shacham and Brent Waters. Compact proofs of retrievability. *Journal of Cryptology*, 26(3):442–483, 2013.
14. Sara Bouchenak, Gregory Chockler, Hana Chockler, Gabriela Gheorghe, Nuno Santos, and Alexander Shraer. Verifying cloud services: Present and future. *SIGOPS Oper. Syst. Rev.*, 47(2):6–19, July 2013.
15. B. Wang, B. Li, and H. Li. Panda: Public auditing for shared data with efficient user revocation in the cloud. *IEEE Transactions on Services Computing*, 8(1):92–106, Jan 2015.
16. Mehmet Sabır Kiraz. A comprehensive meta-analysis of cryptographic security mechanisms for cloud computing. *Journal of Ambient Intelligence and Humanized Computing*, pages 1–30, june 2016.
17. N. Zhang, Q. Shi, and M. Merabti. Anonymous public-key certificates for anonymous and fair document exchange. *IEE Proceedings - Communications*, 147(6):345–350, Dec 2000.
18. D. Critchlow and N. Zhang. Revocation invocation for accountable anonymous pki certificate trees. In *Computers and Communications, 2004. Proceedings. ISCC 2004. Ninth International Symposium on*, volume 1, pages 386–392 Vol.1, June 2004.
19. S. Park, H. Park, Y. Won, J. Lee KISA, and S. Kent. Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1. RFC 5636, August 2009.

20. Jesus Diaz, David Arroyo, and Francisco B. Rodriguez. *Anonymity Revocation through Standard Infrastructures*, pages 112–127. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.
21. Jesus Diaz, David Arroyo, and Francisco B. Rodriguez. New x.509-based mechanisms for fair anonymity management. *Computers & Security*, 46:111–125, 2014.
22. J.R. Douceur, A. Adya, W.J. Bolosky, P. Simon, and M. Theimer. Reclaiming space from duplicate files in a serverless distributed file system. In *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on*, pages 617–624, 2002.
23. Mihir Bellare, Sriram Keelveedhi, and Thomas Ristenpart. Dupless: Server-aided encryption for deduplicated storage. In *Proceedings of the 22Nd USENIX Conference on Security*, SEC'13, pages 179–194. USENIX Association, 2013.
24. Jan Stanek, Alessandro Sorniotti, Elli Androulaki, and Lukas Kencl. *A Secure Data Deduplication Scheme for Cloud Storage*, pages 99–118. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.
25. Nagapramod Mandagere, Pin Zhou, Mark A Smith, and Sandeep Uttamchandani. Demystifying data deduplication. In *Proceedings of the ACM/IFIP/USENIX Middleware '08 Conference Companion*, Companion '08, pages 12–17. ACM, 2008.
26. Dirk Meister and André Brinkmann. Multi-level comparison of data deduplication in a backup scenario. In *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*, SYSTOR '09, pages 8:1–8:12. ACM, 2009.
27. Jia Xu, Ee-Chien Chang, and Jianying Zhou. Weak leakage-resilient client-side deduplication of encrypted data in cloud storage. In *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*, ASIA CCS '13, pages 195–206, New York, NY, USA, 2013. ACM.
28. D. Harnik, B. Pinkas, and A. Shulman-Peleg. Side channels in cloud services: Deduplication in cloud storage. *Security Privacy, IEEE*, 8(6):40–47, Nov 2010.
29. Shai Halevi, Danny Harnik, Benny Pinkas, and Alexandra Shulman-Peleg. Proofs of ownership in remote storage systems. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11, pages 491–500. ACM, 2011.
30. Roberto Di Pietro and Alessandro Sorniotti. Boosting efficiency and security in proof of ownership for deduplication. In *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '12, pages 81–82. ACM, 2012.
31. Wenjin Hu, Tao Yang, and Jeanna N. Matthews. The good, the bad and the ugly of consumer cloud storage. *SIGOPS Oper. Syst. Rev.*, 44(3):110–115, August 2010.
32. Martin Mulazzani, Sebastian Schrittwieser, Manuel Leithner, Markus Huber, and Edgar Weippl. Dark clouds on the horizon: Using cloud storage as attack vector and online slack space. In *Proceedings of the 20th USENIX Conference on Security*, SEC'11, pages 5–5. USENIX Association, 2011.
33. Mihir Bellare, Sriram Keelveedhi, and Thomas Ristenpart. Message-locked encryption and secure deduplication. In Thomas Johansson and PhongQ. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 296–312. Springer Berlin Heidelberg, 2013.
34. Jin Li, Xiaofeng Chen, Mingqiang Li, Jingwei Li, P.P.C. Lee, and Wenjing Lou. Secure deduplication with efficient and reliable convergent key management. *Parallel and Distributed Systems, IEEE Transactions on*, 25(6):1615–1625, June 2014.
35. Jan Stanek, Alessandro Sorniotti, Elli Androulaki, and Lukas Kencl. A secure data deduplication scheme for cloud storage. In Nicolas Christin and Reihaneh Safavi-Naini, editors, *Financial Cryptography and Data Security*, volume 8437 of *Lecture Notes in Computer Science*, pages 99–118. Springer Berlin Heidelberg, 2014.
36. Jian Liu, N. Asokan, and Benny Pinkas. Secure deduplication of encrypted data without additional independent servers. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 874–885, New York, NY, USA, 2015. ACM.
37. Yifeng Zheng, Xingliang Yuan, Xinyu Wang, Jinghua Jiang, Cong Wang, and Xiaolin Gui. Enabling encrypted cloud media center with secure deduplication. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, ASIA CCS '15, pages 63–72. ACM, 2015.
38. Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary Peterson, and Dawn Song. Provable data possession at untrusted stores. Cryptology ePrint Archive, Report 2007/202, 2007. `http://eprint.iacr.org/2007/202`.
39. Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Osama Khan, Lea Kissner, Zachary Peterson, and Dawn Song. Remote data checking using provable data possession. *ACM Trans. Inf. Syst. Secur.*, 14(1):12:1–12:34, June 2011.
40. Huaqun Wang. Proxy provable data possession in public clouds. *Services Computing, IEEE Transactions on*, 6(4):551–559, Oct 2013.
41. Jiawei Yuan and Shucheng Yu. Proofs of retrievability with public verifiability and constant communication cost in cloud. In *Proceedings of the 2013 International Workshop on Security in Cloud Computing*, Cloud Computing '13, pages 19–26, New York, NY, USA, 2013. ACM.

42. Cong Wang, S.S.M. Chow, Qian Wang, Kui Ren, and Wenjing Lou. Privacy-preserving public auditing for secure cloud storage. *Computers, IEEE Transactions on*, 62(2):362–375, Feb 2013.
43. Mehdi Sookhak, Hamid Talebian, Ejaz Ahmed, Abdullah Gani, and Muhammad Khurram Khan. A review on remote data auditing in single cloud server: Taxonomy and open issues. *Journal of Network and Computer Applications*, 43(0):121 – 141, 2014.
44. Alshaimaa Abo-alian, N.L. Badr, and M.F. Tolba. Auditing-as-a-service for cloud storage. In P. Angelov, K.T. Atanassov, L. Doukovska, M. Hadjiski, V. Jotsov, J. Kacprzyk, N. Kasabov, S. Sotirov, E. Szmidt, and S. Zadrożny, editors, *Intelligent Systems'2014*, volume 322 of *Advances in Intelligent Systems and Computing*, pages 559–568. Springer International Publishing, 2015.
45. Mehmet Sabır Kiraz, İsa Sertkaya, and Osmanbey Uzunkol. An Efficient ID-Based Message Recoverable Privacy-Preserving Auditing Scheme. In *Privacy, Security and Trust (PST), 2015 13th Annual Conference on*, pages 117–124. IEEE, July 2015.
46. Hovav Shacham and Brent Waters. Compact proofs of retrievability. In Josef Pieprzyk, editor, *Advances in Cryptology - ASIACRYPT 2008*, volume 5350 of *Lecture Notes in Computer Science*, pages 90–107. Springer Berlin Heidelberg, 2008.
47. Qian Wang, Cong Wang, Jin Li, Kui Ren, and Wenjing Lou. Enabling public verifiability and data dynamics for storage security in cloud computing. In *Proceedings of the 14th European Conference on Research in Computer Security*, ESORICS'09, pages 355–370. Springer-Verlag, 2009.
48. Jia Xu and Ee-Chien Chang. Towards efficient proofs of retrievability. In *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '12, pages 79–80. ACM, 2012.
49. Nathalie Baracaldo, Elli Androulaki, Joseph Glider, and Alessandro Sorniotti. Reconciling end-to-end confidentiality and data reduction in cloud storage. In *Proceedings of the 6th Edition of the ACM Workshop on Cloud Computing Security*, CCSW '14, pages 21–32. ACM, 2014.
50. Youngjoo Shin, Dongyoung Koo, Junbeom Hur, and Joobeom Yun. Secure proof of storage with deduplication for cloud storage systems. *Multimedia Tools and Applications*, pages 1–16, 2015.
51. Jiawei Yuan and Shucheng Yu. Secure and constant cost public cloud storage auditing with deduplication. In *Communications and Network Security (CNS), 2013 IEEE Conference on*, pages 145–153, Oct 2013.
52. Naelah Alkhojandi and Ali Miri. *Foundations and Practice of Security: 7th International Symposium, FPS 2014, Montreal, QC, Canada, November 3-5, 2014. Revised Selected Papers*, chapter Privacy-Preserving Public Auditing in Cloud Computing with Data Deduplication, pages 35–48. Springer International Publishing, Cham, 2015.
53. J. Li, J. Li, D. Xie, and Z. Cai. Secure auditing and deduplicating data in cloud. *IEEE Transactions on Computers*, PP(99), 2015.
54. Mehmet Sabır Kiraz and Osmanbey Uzunkol. Still wrong use of pairings in cryptography. Cryptology ePrint Archive, Report 2016/223, 2016. `https://eprint.iacr.org/2016/223`.
55. I. Blake, G. Seroussi, N. Smart, and J. W. S. Cassels. *Advances in Elliptic Curve Cryptography (London Mathematical Society Lecture Note Series)*. Cambridge University Press, New York, NY, USA, 2005.
56. W. Diffie and M. Hellman. New directions in cryptography. *IEEE Trans. Inf. Theor.*, 22(6):644–654, September 2006.
57. Mihir Bellare and Phillip Rogaway. Optimal asymmetric encryption – how to encrypt with rsa. In *Advances in Cryptology, EUROCRYPT 94*, volume 950, pages 92–111. Springer-Verlag, May 1995.
58. Pascal Paillier and David Pointcheval. Efficient public-key cryptosystems provably secure against active adversaries. In *Proc. of Asiacrypt'99, Lecture Notes in Computer Science*, pages 165–179. Springer Verlag, 1999.
59. Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proceedings of CRYPTO 84 on Advances in cryptology*, pages 10–18, 1985.
60. Jan Camenisch and Els Van Herreweghen. Design and implementation of the idemix anonymous credential system. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, CCS '02, pages 21–30, New York, NY, USA, 2002. ACM.
61. Christian Paquin. U-prove technology overview v1.1 (revision 2). April 2013.
62. Michael J. Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. Keyword search and oblivious pseudorandom functions. In *Theory of Cryptography: Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005. Proceedings*, pages 303–324, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.
63. Stanislaw Jarecki and Xiaomin Liu. Efficient oblivious pseudorandom function with applications to adaptive ot and secure computation of set intersection. In *Proceedings of the 6th Theory of Cryptography Conference on Theory of Cryptography*, TCC '09, pages 577–594, Berlin, Heidelberg, 2009. Springer-Verlag.
64. David Chaum. *Blind Signatures for Untraceable Payments*, pages 199–203. Springer US, Boston, MA, 1983.
65. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, CCS '93, pages 62–73, New York, NY, USA, 1993. ACM.

66. R. C. Merkle. Protocols for public key cryptosystems. In *1980 IEEE Symposium on Security and Privacy*, pages 122–122, April 1980.
67. Oded Goldreich. *The Foundations of Cryptography - Volume 1, Basic Techniques.* Cambridge University Press, 2001.
68. C.J.F. Cremers. *Scyther - Semantics and Verification of Security Protocols.* Ph.D. dissertation, Eindhoven University of Technology, 2006.
69. Cas J. Cremers. The scyther tool: Verification, falsification, and analysis of security protocols. In *Proceedings of the 20th International Conference on Computer Aided Verification*, CAV '08, pages 414–418, Berlin, Heidelberg, 2008. Springer-Verlag.

# A Formal Security Analysis Using Scyhter

## A.1 Security Protocol Description Language (SPDL) Code

Because it is very effective for finding errors, its modeling adversaries with different capabilities, and it is easy to implement the protocol we chose Scyther [68, 69] as an automatic evaluation tool. Scyther's output also ensures that no information is leaked during the protocol.

```
# C_f is modeled as k(I,R) because of private shared information
hashfunction h;
protocol hashed-protocol(I,R) {

role I {
var r: Nonce;
fresh s: Nonce;

send_1(I, R, h(k(I,R))); # U_m sends H_1(C_f) to CSP
recv_2(R, I, r, h(k(I,R), r)); # CSP sends r, H_1(C_f||r) to U_m
send_3(I, R, s, h(k(I,R), r, s) ); # U_m sends s, H_1(C_f||r||s) to CSP

claim_i1(I, Niagree); # checks the integrity of the transmitted data
claim_i2(I, Secret, k(I,R)); # checks the confidentiality of the data
}

role R {
fresh r: Nonce;
var s: Nonce;

recv_1(I, R, h(k(I,R)));
send_2(R, I, r, h(k(I,R), r));
recv_3(I, R, s, h(k(I,R), r, s) );

claim_r1(R, Niagree); # checks the integrity of the transmitted data
claim_r2(R, Secret, k(I,R)); # checks the confidentiality of the data

}
}
```