Security weakness in the Proof of Storage with Deduplication

Youngjoo Shin[1], Junbeom Hur[2], Kwangjo Kim[1]

[1] Department of Computer Science, Korea Advanced Institute of Science and Technology,
{s.youngjoo, kkj}@kaist.ac.kr
[2] School of Computer Science and Engineering, Chung-Ang University
jbhur@cau.ac.kr

Abstract. Achieving both security and efficiency is the challenging issue for a data outsourcing service in the cloud computing. Proof of Storage with Deduplication (POSD) [1] is the first solution that addresses the issue for the cloud storage. However, the validity of the POSD scheme stands on the strong assumption that all clients are honest in terms of generating their keys. We present insecurity of the scheme under new attack model that malicious clients exploit dishonestly manipulated keys. We also propose an improvement of the POSD scheme to mitigate our attack.

1 Introduction

Data outsourcing to a cloud storage brings forth one of new challenges for the efficient resource utilization as well as keeping security for the outsourced data simultaneously. Recently, Zheng and Xu proposed a Proof of Storage with Deduplication (POSD) scheme for a secure and efficient cloud storage service [1]. Exploiting the public verifiability [2], the POSD scheme couples two notions of Proof of Data Possession (PDP) [2][3] and Proof of Data Ownership (POW) [4] and provides a solution to achieve both of security and efficiency. Using the POSD scheme, a client can be assured the integrity of its outsourced data. In addition, a storage server can take advantage of deduplication techniques in a secure manner. That is, the storage server can efficiently utilize resources such as storage space and network bandwidth while preventing information leakage [5][6].

In the POSD scheme, the verification of auditing and deduplication protocol entirely depends on public keys, which are created and provided by clients [1]. Hence, the validity of the scheme is implicitly based on an assumption, which we call random key assumption, 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 random key assumption may cause storage systems that utilize the POSD scheme to face a new security threat not considered before. Unfortunately, the scheme

has a serious security breach under new attack model allowing malicious clients to make dishonestly manipulated keys.

In this paper, we present the security weakness of the POSD scheme. More specifically, we show 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 clients-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 paper is organized as follows: In Section 2, we briefly review the POSD scheme. New attack model and some attack scenarios are presented in Section 3, and countermeasure against the attack is described in Section 4. Finally, we conclude this paper in Section 5.

2 A review of the Proof of Storage with Deduplication

In this section, we briefly review the POSD scheme. For more details, refer to Zheng and Xu's original paper [1].

2.1 Security requirements

The POSD scheme is required to satisfy the following two security properties:
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.
(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.

2.2 The POSD scheme

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 PPT 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 :

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

and verifies

$$\mathrm{X} \stackrel{?}{=} v_1^{\mathrm{Y}_1} v_2^{\mathrm{Y}_2} \prod_{j=1}^{m} z_j^{\mu_j} \bmod p \tag{1}$$

$$e(\mathrm{T}, g) \stackrel{?}{=} e(\mathrm{W} u^{\sum_{i \in \mathrm{I}} \beta_i H_2(x_i)}, z_g) \quad (\in \mathbb{G}_T)\,. \tag{2}$$

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 $\mathrm{I}=\{\alpha_1, \alpha_2, \ldots, \alpha_c\}$ 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

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

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

$$\mathrm{X} \stackrel{?}{=} v_1^{\mathrm{Y}_1} v_2^{\mathrm{Y}_2} \prod_{j=1}^{m} z_j^{\mu_j} \bmod p \tag{3}$$

$$e(\mathrm{T}, g) \stackrel{?}{=} e(\mathrm{W} u^{\sum_{i \in \mathrm{I}} \beta_i H_2(x_i)}, z_g) \quad (\in \mathbb{G}_T)\,, \tag{4}$$

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

3 Weakness of the POSD scheme

In this section, we describe new attack model against the POSD scheme and present some attack scenarios.

3.1 Weak key attack

A malicious client may create the manipulated keys of his/her own dishonestly rather than executing the KEYGEN algorithm under the random key assumption. 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 $(1 \le j \le m)$ are assigned as follows:

$$v_1 = \xi,\ v_2 = \xi^{-1},$$
$$s_{j1} = s_{j2} = \psi_j \ (1 \le j \le m).$$

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. We describe details 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 $\widehat{\mathrm{Tag}}_{int}$:

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

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

$$\begin{aligned}
& v_1^{\mathrm{Y}_1} v_2^{\mathrm{Y}_2} \prod_{j=1}^{m} z_j^{\mu'_j} \\
&= v_1^{\sum_{i\in \mathrm{I}} \beta_i y_{1i}} v_2^{\sum_{i\in \mathrm{I}} \beta_i y_{12}} \prod_{j=1}^{m} \Big(v_1^{-s_{j1}} v_2^{-s_{j2}}\Big)^{\sum_{i\in \mathrm{I}} \beta_i \mathsf{F'}_{ij}} \\
&= v_1^{\sum_{i\in I} \beta_i y_{1i}} v_2^{\sum_{i\in \mathrm{I}} \beta_i y_{12}} \prod_{j=1}^{m} \xi^{(\psi_j - \psi_j)\sum_{i\in \mathrm{I}} \beta_i \mathsf{F'}_{ij}} \\
&= v_1^{\sum_{i\in \mathrm{I}} \beta_i \left(r_{i1} + \sum_{j=1}^{m} \mathsf{F}_{ij} S_{j1}\right)} v_2^{\sum_{i\in \mathrm{I}} \beta_i \left(r_{i2} + \sum_{j=1}^{m} \mathsf{F}_{ij} S_{j2}\right)} \\
&= \Big(v_1^{\sum_{i\in \mathrm{I}} \beta_i r_{i1}} v_2^{\sum_{i\in \mathrm{I}} \beta_i r_{i2}}\Big) \xi^{\sum_{i\in \mathrm{I}} \sum_{j=1}^{m} \beta_i \mathsf{F}_{ij} \psi_j - \sum_{i\in \mathrm{I}} \sum_{j=1}^{m} \beta_i \mathsf{F}_{ij} \psi_j} \\
&= (v_1^{r_{i1}} v_2^{r_{i2}})^{\sum_{i\in \mathrm{I}} \beta_i} \\
&= \prod_{i\in \mathrm{I}} x_i^{\beta_i} \\
&= \mathrm{X}. \qquad (5)
\end{aligned}$$

Eq. (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 $\mathsf{resp} = \big(\{\mu'_j\}_{1\le j\le m}\big)$ to the server. In the verification phase, Eqs. (3) and (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.

3.2 Attack scenario

Exploiting the weak keys, several practical attacks against storage systems using the POSD scheme are feasible. Below we list some plausible attack scenarios.

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′, $\widehat{\text{Tag}_{int}}$), where fid and $\widehat{\text{Tag}_{int}}$ 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 DEDUP 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 AUDITINT 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 [4], 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).

4 Countermeasure

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.

4.1 Modified POSD scheme

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. The modified POSD scheme is described as follows:
KEYGEN, AUDITINT, DEDUP: The key generation algorithm and the auditing and the deduplication protocols are same as those of the original scheme.
UPLOAD: The modified file uploading protocol is described as follows:

1-2. The procedures are same as described in steps 1-2 of the original **UPLOAD** protocol.

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}.

4.2 Security analysis

Now we show that our modified scheme satisfies two security requirements, server unforgeability and (κ, θ)-uncheatability [1], even under the weak key attack.

Theorem 1. *The modified* POSD *scheme is server unforgeable and (κ, θ)-uncheatable in the random oracle model without the random key assumption.*

Proof. We present our proof 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 [1]. 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

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

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

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

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 [1].

– 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.

5 Conclusion

We have addressed a security weakness of recently proposed proof of storage with deduplication (POSD) scheme [1]. Under new attack model allowing malicious clients to exploit dishonestly manipulated keys, the POSD scheme fails to guarantee required security. To mitigate the attack, we proposed an improved scheme blending client's keys with the server contributed random values.

References

1. Zheng, Q., Xu, S.: Secure and efficient proof of storage with deduplication. In: Proceedings of the second ACM conference on Data and Application Security and Privacy. CODASPY '12 (2012) 1–12
2. Ateniese, G., Burns, R., Curtmola, R., Herring, J., Kissner, L., Peterson, Z., Song, D.: Provable data possession at untrusted stores. In: Proceedings of the ACM Conference on Computer and Communications Security. (2007) 598–610
3. Ateniese, G., Pietro, R.D., Mancini, L.V., Tsudik, G.: Scalable and efficient provable data possession. In: Proceedings of the 4th International Conference on Security and Privacy in Communication Networks, SecureComm'08. (2008) 9:1–9:10

4. Halevi, S., Harnik, D., Pinkas, B., Shulman-Peleg, A.: Proofs of ownership in remote storage systems. In: Proceedings of the ACM Conference on Computer and Communications Security. (2011) 491–500
5. Harnik, D., Pinkas, B., Shulman-Peleg, A.: Side channels in cloud services: Deduplication in cloud storage. IEEE Security and Privacy Magazine **8** (November 2010) 40–47
6. Mulazzani, M., Schrittwieser, S., Leithner, M., Huber, M., Weippl, E.: Dark clouds on the horizon: using cloud storage as attack vector and online slack space. Proc. USENIX Conf. Security (SEC'11) (2011)