Improvement of a Remote Data Possession Checking Protocol from Algebraic Signatures*

Yong Yu[1,2], Jianbing Ni[1], Jian Ren[3], Wei Wu[4], Lanxiang Chen[4], and Qi Xia[1]

[1] School of Computer Science and Engineering,
University of Electronic Science and Technology of China, Chengdu, 611731, China
{yyucd2012,nimengze,weiwu81,xiaqi0769}@gmail.com
[2] State Key Laboratory of Information Security, Institute of Information Engineering,
Chinese Academy of Sciences, Beijing 100093, China
[3] Department of Electrical and Computer Engineering,
Michigan State University, MI, 4882, USA
renjian@egr.msu.edu
[4] School of Mathematics and Computer Science,
Fujian Normal University, Fuzhou, 350007, China
lxiangchen@fjnu.edu.cn

Abstract. Cloud storage allows cloud users to enjoy the on-demand and high quality data storage services without the burden of local data storage and maintenance. However, the cloud servers are not necessarily fully trusted. As a consequence, whether the data stored on the cloud are intact becomes a major concern. To solve this challenging problem, recently, Chen proposed a remote data possession checking (RDPC) protocol using algebraic signatures. It achieves many desirable features such as high efficiency, small challenges and responses, non-block verification. In this paper, we find that the protocol is vulnerable to replay attack and deletion attack launched by a dishonest server. Specifically, the server can either fool the user to believe that the data is well maintained but actually only a proof of the challenge is stored, or can generate a valid response in the integrity checking process after deleting the entire file of the user. We then propose an improved scheme to fix the security flaws of the original protocol without losing the desirable features of the original protocol.

1 Introduction

Cloud storage provides a novel service model wherein data are maintained, managed and backed up remotely and accessed by the users over the network at

* This work is supported by the NSFC of China under Grants 61003232, 61370203, 61202450, the National Research Foundation for the Doctoral Program of Higher Education of China under Grants 20100185120012, 20123503120001, the NSFC of China for International Young Scientists under Grant 61250110543, Department of Education, Fujian Province, A-Class Project under Grant JA12076, and the Fundamental Research Funds for the Central Universities under Grant ZYGX2011J067.

X. Huang and J. Zhou (Eds.): ISPEC 2014, LNCS 8434, pp. 359–372, 2014.

anytime and from anywhere [1]. Although cloud storage is targeted to take up much of the workload from the client, it is fraught with security risks [2]. On the one hand, frequent data access increases the probability of disk corruption, as a result, loss of the data may occur constantly. Simultaneously, cloud service providers may try to hide data loss incidents in order to maintain their reputation. On the other hand, cloud providers are not fully trusted and thus, they might discard the data that have not been or are rarely accessed for monetary reasons. Therefore, whether the stored data keeps virgin is a major concern of the cloud users.

In order to check the data integrity at untrusted stores, in 2007, Ateniese et al. [3, 4] proposed the notion of provable data possession (PDP) for the first time and presented two efficient and provably-secure PDP schemes based on homomorphic verifiable tags. In their protocols, users are allowed to verify data integrity without accessing the entire file. At the same time, Juels et al. [5] formalized the model of proof of retrievability (PoR) which enables the server to produce a concise proof that a user can retrieve data, and then, presented a sentinel-based PoR scheme using error-correcting code. In 2008, Shacham and Waters [6, 7] described two efficient and compact PoR schemes. In 2009, Ateniese et al. [8] provided a framework for building public-key homomorphic linear authenticators from any identification protocol, and then described how to turn any public-key homomorphic linear authenticator into a publicly-verifiable PDP scheme with an unbounded number of verifications. Subsequently, a number of data auditing protocols [9–16] from some efficient PDP and PoR schemes [5–7, 17–19], were proposed to ensure the integrity of users' data. In 2013, Chen [20] proposed an algebraic signature based remote data possession checking (RDPC) protocol, which is a similar notion inherited from PDP, but the number of verifications in their basic protocol is limited. To overcome this drawback, an improved scheme supporting to refresh tags after t verifications was also proposed in [20]. Both protocols provide a number of desirable features of a remote data possession checking protocol such as high efficiency, small challenges and responses, no-block verification and were suggested to be adopted to the cloud storage scenario.

Our Contribution. In this paper, we identify several security flaws in the RDPC protocols in [20]. Firstly, neither the basic protocol nor the improved one is secure against the replay attack, in which the server is able to generate a valid proof from the previous proofs, without accessing the actual data. Consequently, the server needs only to store a previous proof and replay it as a valid response when required. Secondly, the improved protocol is vulnerable to a malicious server's deletion attack; namely, the server can generate a valid response in the integrity checking process after deleting the original data file. Then, we propose a new RDPC protocol to fix these security problems. Finally, we show the fixed protocol is secure based on the security model due to Ateniese et al. [3] and maintains the desirable features of the original protocol on performance.

Organization. Section 2 gives some preliminaries used in this paper. Section 3 reviews the RDPC protocols in [20] and discusses the security of the protocols. Section 4 describes our new RPDC protocol. Section 5 provides security proofs for the new RDPC protocol and Section 6 concludes the paper.

2 Preliminaries

In this section, we review basic knowledge of the RDPC protocols, including security model, components and security requirements of a RDPC protocol.

2.1 System Model

The remote data possession checking architecture for cloud storage involves two kinds of entities: a cloud server and its users. The cloud server, which has significant storage space and computation resources, stores users' data and provides data access service. The users have large amount of data to be stored on the cloud in order to eliminate the overhead of local storage. As users no longer possess the entire data locally and the cloud server is not fully-trusted, it is of critical importance for users to ensure their data are correctly stored and maintained in the cloud. Therefore, the users should be able to efficiently check the integrity and correctness of their outsourced data.

2.2 Components of a RDPC Protocol

A remote data possession checking protocol, which can be used to verify the integrity of the users' data, consists of five phases: **Setup**, **TagBlock**, **Challenge**, **ProofGen** and **ProofVerify** [3, 4].

- **Setup** is a probabilistic algorithm run by the user to setup the protocol. It takes a security parameter κ as input and returns k as the secret key of the user.
- **TagBlock** is a probabilistic algorithm that is run by the user to generate tags for a file. It takes the secret key k and a file F as input and returns the set of tags T for file F.
- **Challenge** is a probabilistic algorithm that is run by the user to generate a challenge. It takes the security parameter κ as input and returns the challenge *chal*.
- **ProofGen** is a deterministic algorithm that is run by the cloud server in order to generate a proof of possession. It takes the blocks of file F and the set of tags T as input and returns a proof of possession R for the challenged blocks in F.
- **ProofVerify** is a deterministic algorithm that is run by the user in order to evaluate a proof of possession. It takes his secret key k, the challenge *chal* and the proof of possession R as input, and returns whether the proof is a correct proof of possession for the blocks challenged by *chal*.

2.3 Security Requirements

In cloud storage, the cloud server could be self-interested and might hide data corruption incidents to maintain its reputation. So a practical remote data possession checking protocol should be secure against the internal attacks a cloud server can launch, namely replace attack, forge attack, replay attack and deletion attack [14].

- Replace attack: The server may choose another valid pair of data block and tag (F_i, T_i) to replace a challenged pair of data block and tag (F_j, T_j), when it has already discarded F_j or T_j.
- Forge attack: The server may forge valid tags of data blocks to deceive the user.
- Replay attack: The server may generate a valid proof of possession R from previous proofs or other information, without accessing the outsourced data.
- Deletion attack: The server may generate a valid proof R making use of the tags T or other information, even the user's entire file has been deleted.

The security game due to Ateniese et al. [4] covers all the attacks mentioned above by capturing that an adversary cannot produce a valid proof without possessing all the blocks corresponding to a given challenge, unless it guesses all the missing blocks. The details of the game are as follows:

- Setup: The challenger runs **Setup** algorithm to generate a secret key k and keeps it secret.
- Query: The adversary chooses some data blocks $F_i (i = 1, \cdots, n)$ and makes tag queries adaptively. The challenger computes the corresponding tags $T_i (i = 1, \cdots, n)$ for the blocks and sends them back to the adversary.
- Challenge: The challenger generates a challenge *chal* and requests the adversary to respond a proof of possession R for the challenged blocks.
- Forge: The adversary computes a proof R for the challenged blocks and returns it to the challenger.

The adversary wins the game if $\mathbf{VerifyProof}(k, chal, T_i (i = 1, \cdots, n), R)$ holds. A RDPC protocol is secure against a malicious server if for any (probabilistic polynomial-time) adversary the probability that it wins the security game on a set of file blocks is negligible.

3 On the Security of the RDPC Protocols

In this section, we review the basic RDPC protocol and the improved one in [20] and show that both protocols are insecure against the replay attack, and the improved scheme is also susceptible to the deletion attack.

3.1 A Brief Review of the RDPC Protocols

The following symbols are used in the RDPC protocols in [20].

- t: the number of verifications;
- c: the number of blocks challenged in each challenge;
- $E_{kt}(\cdot)$, $D_{kt}(\cdot)$: the encryption and decryption algorithms of a symmetric cryptosystem, where kt is the symmetric key;
- $F = F[1], \cdots, F[n]$: F denotes a file name, and $F[i]$ denotes the ith data block of the file F;
- $T = T_1, \cdots, T_t$: T denotes the set of block tags and T_i denotes the ith tag;
- $f(\cdot)$: $\{0,1\}^\kappa \times \{0,1\}^l \rightarrow \{0,1\}^l$, denotes a pseudo-random function (PRF);
- $\sigma(\cdot)$: $\{0,1\}^\kappa \times \{1, \cdots, n\} \rightarrow \{1, \cdots, n\}$, denotes a pseudo-random permutation (PRP);
- $AS_g(\cdot)$: denotes an algebraic signature algorithm. Here the algebraic signature on a block $s_0, s_1, \cdots, s_{n-1}$ is defined as:

$$AS_g(s_0, s_1, \cdots, s_{n-1}) = \textstyle\sum_{i=0}^{n-1} s_i \cdot g^i,$$

where g is a primitive element of a Galois field [21].

The details of the basic RDPC protocol in [20] are described in Figure 1.

Setup: The user generates a master key $k \xleftarrow{R} \{0,1\}^\kappa$, an encryption key $kt \xleftarrow{R} \{0,1\}^\kappa$, and two random values $r_1, r_2 \xleftarrow{R} \{0,1\}^\kappa$.	**Challenge**: For the ith challenge, the user computes $k_i = f_k(r_1 + i)$, and sends $< r_2, k_i >$ to the server.
TagBlock: $0 < i \leq t$ $k_i = f_k(r_1 + i)$ $s = 0$ for $0 < j \leq c$ $l_j = \sigma_{k_i}(r_2 + j)$ $s = s + F[l_j]$ $\delta_i = AS_g(s)$ $T_i = E_{kt}(\delta_i)$ The user sends $< F, T >$ to the server.	**ProofGen**: $F'_i = 0$ for $0 < j \leq c$ $l_j = \sigma_{k_i}(r_2 + j)$ $F'_i = F'_i + F[l_j]$ return $< F'_i, T'_i >$. **ProofVerify**:The user checks whether $AS_g(F'_i) = D_{kt}(T'_i)$ holds.

Fig. 1. The basic RDPC protocol in [20]

An improved scheme using challenge updating was proposed as well to overcome the drawback of limited number of data verifications in the basic protocol. The new tag generation and challenge updating are shown in Figure 2 and the other processes are the same as those of the basic protocol.

Setup: The user picks a master key $k \xleftarrow{R} \{0,1\}^{\kappa}$, the encryption key $kt \xleftarrow{R} \{0,1\}^{\kappa}$, and three random numbers $r_1, r_2, r_3 \xleftarrow{R} \{0,1\}^{\kappa}$.

TagBlock:

for $0 < i \leq n$
$\tau_i = AS_g(F[i])$
for $0 < i \leq t$
$s = 0$
$k_i = f_k(r_1 + i)$
for $0 < j \leq c$
$l_j = \sigma_{k_i}(r_2 + j)$
$s = s + \tau_{l_j}$
$T_i = E_{kt}(s)$
Forward $< F, T, \tau >$ to the server.

Challenge-updating:

For the mth updating
$k_m^u = f_k(r_3 + m)$
for $0 < i \leq t$
s=0
$k_i = f_{k_m^u}(r_1 + i)$
for $0 < j \leq c$
$l_j = \sigma_{k_i}(r_2 + j)$
$s = s + \tau_{l_j}$
$T_i'' = E_{kt}(s)$
Send $< T'' >$ to the server.

Fig. 2. The improved RDPC protocol in [20]

3.2 Replay Attacks on the Protocols

The User		The Server
1.For the first challenge, compute $k_1 = f_k(r_1 + 1)$	$< r_2, k_1 >$ →	
		2.Compute $l_j = \sigma_{k_1}(r_2 + j)$ for $1 \leq j \leq c$ and $F_1' = \sum_{j=1}^{c} F[l_j]$
	← $< F_1', T_1' >$	
3.Verify $AS_g(F_1') = D_{kt}(T_1')$		
		4.If the proof is available, discard $< F, T >$, and keep $< F_1', T_1' >$
5.For the ith challenge, compute $k_i = f_k(r_1 + i)$	$< r_2, k_i >$ →	
	← $< F_1', T_1' >$	6.Replay $< F_1', T_1' >$
7.Verify $AS_g(F_1') = D_{kt}(T_1')$		

Fig. 3. Replay attack on the RDPC protocols

The replay attack, a serious security threat to RDPC protocols, says that the server can generate a valid proof from previous proofs or other information

without accessing the actual data of the user. In the RDPC protocols in [20], since the replayed proof $< F_1', T_1' >$ can always make the equation $AS_g(F_1') = D_{kt}(T_1')$ hold and as a consequence, the server only needs to keep $< F_1', T_1' >$ instead of the entire file and verifiable tags $< F, T >$ of the user. Thus, the RPDC protocols in [20] are insecure against replay attack as shown in Figure 3.

3.3 Deletion Attack on the Improved Protocol

The deletion attack enables the server to generate a valid proof from the block tags or other information after deleting all the stored data of the user. In the improved protocol in [20], the server can launch deletion attack to fool the user to believe that the data in the cloud are well maintained, while actually only the block tags are stored. The details of the attack are shown below:

- Receiving the stored file $< F, T, \tau >$ from the user, the server keeps the values $< T, \tau >$ and discards the file F.
- When receiving the ith challenge $< r_2, k_i >$ from the user, the server computes $l_j = \sigma_{k_i}(r_2 + j)$ for each $j \in [1, c]$, and generates the l_j-th data block $F^*[l_j] = s^*_{l_j,0}, \cdots, s^*_{l_j,n-1}$ using τ_{l_j} as follows: pick $n-1$ random values $s^*_{l_j,1}, \cdots, s^*_{l_j,n-1}$, and compute $s^*_{l_j,0}$ as:

$$s^*_{l_j,0} = \tau_{l_j} - \sum_{j=1}^{n-1} s^*_{l_j,j} \cdot g^j.$$

- After generating all the challenged data blocks $\{F^*_{l_1}, \cdots, F^*_{l_c}\}$, the server computes $F_i^* = \sum_{j=1}^{c} F^*_{l_j}$ and responds the proof $< F_i^*, T_i >$ to the user.

The verification equation $AS_g(F_i^*) = D_{kt}(T_1)$ holds since F_i^* is equal to F_i and thus, the user believes that the data in the cloud are well maintained. But in fact, the server stores only the block tags of the file $< T, \tau >$ instead of the whole data $< F, T, \tau >$. As a consequence, the server can delete the file F and rent the storage space to other cloud users without being detected by the user in data possession checking process.

4 Our RDPC Protocol

To enhance the security of original RDPC protocols in [20], we incorporate the basic RDPC scheme and the tricks due to Shacham and Waters [6, 7], namely, we involve the name of the file F_{id} and the block sequence numbers i in generating block tags. Besides, since algebraic signatures in [20] are non-cryptographic encoding methods rather than digital signatures, the server can generate a valid proof using the block tags τ after deleting all the data of the user. In our protocol, we enhance the algebraic signature algorithm by involving pseudo-random functions. Moreover, to improve the efficiency of the RDPC protocol, we make use of the random sampling technique to challenge the server. It's not necessary

for the user to update the tags after t times verifications in the new protocol. Besides, the communication flows of our RDPC protocol should be transmitted via an authenticated and reliable channel in order to avoid the attack by Ni et al. [22]. The details of **Setup**, **TagBlock**, **Challenge**, **ProofGen** and **ProofVerify** are shown below.

- **Setup**: κ denotes the security parameter which determines the size of a prime q. Let G_1 be a cyclic group generated by g with order q. The user generates a secret key $k \xleftarrow{R} Z_q^*$, and defines two pseudo-random functions (PRF): δ: $\{0,1\}^* \times Z_q^* \to Z_q^*$ and ϕ: $Z_q^* \times Z_q^* \to Z_q^*$. π: $Z_q^* \times \{1, \cdots, n\} \to \{1, \cdots, n\}$ represents a pseudo-random permutation (PRP); H: $\{0,1\}^* \to Z_q^*$ stands for a hash function. Choose k_{enc} as the secret key of a symmetric encryption scheme $Enc(\cdot)$ and $Dec(\cdot)$.
- **TagBlock**: Given a file F, the user firstly splits F into m blocks $F = \{F[1], \cdots, F[m]\}$, further divides each block say $F[i]$ into n sectors $\{s_{i,1}, \cdots, s_{i,n}\}$. Next, the user picks n random values $\{\alpha_1, \cdots, \alpha_n\}$ in Z_q^* and generates $\tau = F_{id}||m||n||Enc_{k_{enc}}(\alpha_1||\cdots||\alpha_n)$ as the file tag of the file F where F_{id} is the name of the file F. Then, the user computes the verifiable tag of $F[i]$ as

$$T_i = \sum_{j=1}^{n} (\alpha_j \cdot s_{i,j} + H(F_{id}||g_{id}||i)) \cdot g_{id}^j,$$

  where g_{id} is computed as $g_{id} = \delta_k(F_{id})$. Finally, the user sets $T = \{T_1, \cdots, T_m\}$ and sends (τ, F, T) to the server.
- **Challenge**: The user chooses a value c as the number of the blocks challenged, and generates two random numbers $k_1 \xleftarrow{R} Z_q^*$, $k_2 \xleftarrow{R} Z_q^*$, then sends the challenge $chal = (c, k_1, k_2)$ to the server.
- **ProofGen**: Upon receiving the challenge from the user, the server computes the $l_t = \pi_{k_1}(t)$ and $a_t = \phi_{k_2}(t)$ for $1 \le t \le c$. Then the server generates $\sigma = \sum_{t=1}^{c} a_t \cdot T_{l_t}$ and $\rho_j = \sum_{t=1}^{c} a_t s_{l_t,j}$ for $1 \le j \le n$. Finally, the server sets $\rho = \{\rho_1, \rho_2, \cdots, \rho_n\}$ and responds (τ, σ, ρ) to the user.
- **ProofVerify**: Upon receiving the proof from the server, the user computes $l_t = \pi_{k_1}(t)$, $a_t = \phi_{k_2}(t)$ for $1 \le t \le c$ and $g_{id} = \delta_k(F_{id})$, then decrypts $\alpha_1||\cdots||\alpha_n = Dec_{k_{enc}}(Enc_{k_{enc}}(\alpha_1||\cdots||\alpha_n))$ and checks whether the identity holds:

$$\sigma = \sum_{j=1}^{n} (\alpha_j \cdot \rho_j + \sum_{t=1}^{c} (a_t \cdot H(F_{id}||g_{id}||l_t))) \cdot g_{id}^j.$$

  If the equation holds, it indicates the user's data are well maintained; Otherwise, the data have been corrupted. The protocol is illustrated in Figure 4.

The User		The Server
1.Compute τ and $T_i = \sum_{j=1}^{n}(\alpha_j s_{i,j} + H(F_{id}\|\|g_{id}\|\|i))g_{id}^j$ for $F[i]$ where $g_{id} = \delta_k(F_{id})$.	$\xrightarrow{(\tau, F, T)}$	2.Store (τ, F, T).
3.Generate a random challenge: $chal = (c, k_1, k_2)$	$\xrightarrow{(c, k_1, k_2)}$	4.Compute $l_t = \pi_{k_1}(t)$ and $a_t = \phi_{k_2}(t)$ for $1 \le t \le c$.
		5.Generate $\sigma = \sum_{t=1}^{c} a_t \cdot T_{l_t}$.
		6.Generate $\rho_j = \sum_{t=1}^{c} a_t s_{l_t,j}$ for $1 \le j \le n$.
	$\xleftarrow{(\tau, \sigma, \rho)}$	
7.Compute $l_t = \pi_{k_1}(t)$ and $a_t = \phi_{k_2}(t)$ for $1 \le t \le c$ and $g_{id} = \delta_k(F_{id})$.		
8.Decrypt τ and check whether the equation $\sigma = \sum_{j=1}^{n}(\alpha_j \cdot \rho_j + \sum_{t=1}^{c}(a_t \cdot H(F_{id}\|\|g_{id}\|\|l_t))) \cdot g_{id}^j$ holds.		

Fig. 4. Our new RDPC protocol

The correctness of the protocol is elaborated as follows:

$$\sigma = \sum_{t=1}^{c} a_t \cdot T_{l_t} \tag{1}$$

$$= \sum_{t=1}^{c} a_t \cdot \sum_{j=1}^{n}(\alpha_j \cdot s_{l_t,j} + H(F_{id}||g_{id}||l_t)) \cdot g_{id}^j \tag{2}$$

$$= \sum_{j=1}^{n}\sum_{t=1}^{c} a_t(\alpha_j \cdot s_{l_t,j} + H(F_{id}||g_{id}||l_t)) \cdot g_{id}^j \tag{3}$$

$$= \sum_{j=1}^{n}(\sum_{t=1}^{c} a_t \cdot \alpha_j \cdot s_{l_t,j} + \sum_{t=1}^{c} a_t \cdot H(F_{id}||g_{id}||l_t)) \cdot g_{id}^j \tag{4}$$

$$= \sum_{j=1}^{n}(\alpha_j \cdot \rho_j + \sum_{t=1}^{c}(a_t \cdot H(F_{id}||g_{id}||l_t))) \cdot g_{id}^j \tag{5}$$

5 Security Proofs

In this section, we prove that our RDPC protocol is secure under the security model of Ateniese et al. [3] using the tricks due to Shacham and Waters [6, 7]. Intuitively, without maintaining the whole file, an adversary cannot generate a valid response to a challenge. That is, we will prove that the **ProofVerify**

algorithm will reject except when the prover's ρ_j are computed correctly, i.e. are such that $\rho_j = \sum_{t=1}^{c} a_t s_{l_t,j}$.

Theorem 1. *If the pseudo-random function is secure and the symmetric encryption scheme is semantically secure, then there exists no adversary to break our RDPC protocol, that cause the user to accept a corrupted proof in the remote data possession checking process, within non-negligible probability, except the responding proof (σ, ρ) is computed correctly by* ***ProofGen*** *phase.*

In order to prove theorem 1, we construct a series of games and interleave the game description by limiting the difference in adversary's behavior between successive games.

Game 0. The first game, Game 0, is defined the same as the security game defined in Section 2.3.

Game 1. In Game 1, the challenger uses the a random bit-string of the same length as encryption of $\alpha_1||\cdots||\alpha_n$ to replace the ciphertext. When given a challenged tag, the adversary can distinguish the encrypted value of the tag, rather than attempting to decrypt the ciphertext, the challenger declares failure and aborts.

Analysis. In Game 1, the challenger keeps a table of plaintexts $\alpha_1||\cdots||\alpha_n$ and their tags to respond queries in decryption oracles. The challenger can break the semantic security of the symmetric encryption scheme employing the adversary if the probability of the adversary's success between Game 0 and Game 1 is non-negligible. In order to bridge the gap between Game 0 and Game 1, we must use a hybrid argument between "all valid encryption" and "no valid encryption", which will cause the reduction suffer a $1/q_s$ security loss, where q_s is the number of queries made by the adversary.

Specifically, the challenger interacts with the adversary $\mathcal{A}$ following the security game in section 2.3 and keeps track of the files stored by $\mathcal{A}$. Then, if $\mathcal{A}$ succeeds in some data integrity checking interaction with a proof that is different from that would be generated by the **ProofGen** algorithm, the challenger aborts and outputs 1; Otherwise, outputs 0. Assume the challenger outputs 1 with some non-negligible probability ϵ_0 if its behavior is as specified in Game 0, and the challenger outputs 1 with some non-negligible probability ϵ_1 if its behavior is as specified in Game 1, we will show that the gap between ϵ_0 and ϵ_1 is negligible as long as the symmetric encryption scheme is semantic secure.

In game 0, the challenger uses the ciphertext of $\alpha_1||\cdots||\alpha_n$ to generate each tag. In Game 1, the challenger encrypts a random string of the same length in generating each tag. Suppose that $|\epsilon_1 - \epsilon_0|$ is non-negligible. Consider the hybrid argument in which the challenger generates the first i tags using random ciphertexts, and the remaining $q_s - i$ tags involving random values. Thus, there must be a value of i such that the difference between the challenger's outputs in hybrid i and hybrid $i+1$ is at least $|\epsilon_1 - \epsilon_0|/q_s$, which is non-negligible. According to this, we will construct an algorithm $\mathcal{B}$ to break the security of the symmetric encryption scheme.

The encryption oracle for k_{enc} is accessible to $\mathcal{B}$, as well as a left-or-right oracle which given strings m_0 and m_1 of the same length, outputs the encryption of m_b, where b is a random bit. $\mathcal{B}$ interacts with $\mathcal{A}$ acting as the challenger. In answering $\mathcal{A}$'s first i queries, $\mathcal{B}$ uses its encryption oracle to obtain the encryption of $\alpha_1||\cdots||\alpha_n$, which includes in the tag. In answering $\mathcal{A}$'s the $(i+1)$th query, $\mathcal{B}$ generates the correct plaintext $m_0 = \alpha_1||\cdots||\alpha_n$ and a random plaintext m_1 of the same length and submits both to its left-or-right oracle. In answering $\mathcal{A}$'s remaining queries, $\mathcal{B}$ encrypts a random plaintext which has the same length as the correct plaintext using its encryption oracles and includes the result in the tags. $\mathcal{B}$ keeps track of the files stored by the adversary. If $\mathcal{A}$ succeeds in some data possession checking interaction but the proof is different from that would be generated by the **ProofGen** algorithm, the challenger aborts and outputs 1; Otherwise, outputs 0.

If the left-or-right oracle receives its left input, $\mathcal{B}$ is interacting with $\mathcal{A}$ according to hybrid i. If the left-or-right oracle receives its right input, $\mathcal{B}$ is interacting with $\mathcal{A}$ according to hybrid $i+1$. There is a non-negligible difference in $\mathcal{A}$'s behavior and therefore in $\mathcal{B}$'s, which breaks the security of the symmetric encryption scheme. Note that, since the values $\alpha_1||\cdots||\alpha_n$ are randomly chosen and independent with each file, the values given by $\mathcal{B}$ to its left-or-right oracle are consistent with a query it makes to its encryption oracle with negligible probability.

Game 2. In Game 2, the challenger uses truly random values in Z_p^* instead of the outputs of the pseudo-random function δ, remembering these values to use in verifying the validation of the adversary's responses in data possession checking instances. More specifically, the challenger evaluates g_{id} not by applying the PRF $g_{id} = \delta_k(F_{id})$, but by generating a random value $r \leftarrow Z_q^*$ and inserting an entry (k, F_{id}, r) in a table; it queries this table when evaluating the PRF δ to ensure consistency.

Analysis. In Game 2, the challenger uses random values to replace the outputs of the PRF δ and then keeps a table of (k, F_{id}, r) to ensure the verification of the adversary's proof. If there is a difference in the adversary's success probability between Games 1 and 2, we can use the adversary to break the security of the PRF δ. This means that if the adversary can distinguish random values from the outputs of PRF, the challenger can break the security of the PRF involving the adversary.

As in the analysis of Game 2, the difference in behavior we use to break the security of PRF is the event that the adversary succeeds in a data possession checking interaction but responded values (σ, ρ) are different from those that would be by the **ProofGen** algorithm. Similar to the analysis of Game 1, a hybrid argument is necessary to proof Game 2 with a security loss $1/(mq_s)$ in the reduction, where m is a bound on the number of blocks in any file the adversary requests to have stored.

Game 3. In Game 3, the challenger handles RDPC protocol executions initiated by the adversary differently than in Game 2. In each such RDPC protocol execu-

tion, the challenger issues a challenge as before. However, the challenger verifies the adversary's response differently from what it specified in **ProofVerify** phase.

The challenger keeps a table of the **TagBlock** queries made by the adversary and the corresponding responses to maintain consistency; the challenger knows the values ρ_j and σ that the sever would have produced in response to the query it issued. If the values the adversary sent were exactly these values, the challenger accepts the adversary's response. If in any of these interactions the adversary responds in such a way that (1) passes the verification algorithm but (2) is not what would have been computed by an honest server, the challenger declares failure and aborts.

Analysis. The adversary's view is different in Game 3 and Game 2 only when the response of adversary (1) can make the verification algorithm satisfied but (2) is not what would have been computed by the challenger, which acts as an honest server, in some RDPC protocol interaction. We show that the probability that this happens is negligible.

Before analyzing the difference in probabilities between Game 3 and Game 2, we firstly describe the notion and draw a few conclusions. Suppose the file F that causes the abort is divided into m data blocks $F = F[1], \cdots, F[m]$, further divides each block into n sectors $F[i] = s_{i,1}, \cdots, s_{i,n}$. F_{id} denotes the name of the file F and i represents the block number of $F[i]$. Assume $chal = (c, k_1, k_2)$ is the query that causes the challenger to abort and the adversary's response to that query is (σ^*, ρ^*). If the adversary's response satisfies the verification–i.e., if

$$\sigma^* = \sum_{j=1}^{n}(\alpha_j \cdot \rho_j^* + \sum_{t=1}^{c}(a_t \cdot H(F_{id}||r||l_t))) \cdot r^j,$$

where $l_t = \pi_{k_1}(t)$ and $a_t = \phi_{k_2}(t)$ for $1 \leq t \leq c$ and r is the random value substituted by Game 2 for g_{id}. Let the expected response, which would have been obtained from an honest prover, be (σ, ρ), where $\sigma = \sum_{t=1}^{c} a_t \cdot T_{l_t}$ and $\rho_j = \sum_{t=1}^{c} a_t s_{l_t,j}$. Because of the correctness of the protocol, the expected response can pass the verification equation, that is

$$\sigma = \sum_{j=1}^{n}(\alpha_j \cdot \rho_j + \sum_{t=1}^{c}(a_t \cdot H(F_{id}||r||l_t))) \cdot r^j.$$

Observe that if $\rho_j^* = \rho_j$ for each j, the value of σ^* should be equal to σ, which contradicts our assumption above. Therefore, let us define $\Delta\sigma = \sigma^* - \sigma$ and $\Delta\rho_j = \rho_j^* - \rho_j$ for $1 \leq j \leq n$ and subtracte the verification equation for σ from that for σ^*, we have

$$\Delta\sigma = \sum_{j=1}^{n} \alpha_j \cdot \Delta\rho_j \cdot r^j.$$

The bad event occurs exactly when some $\Delta\rho_j$ is not zero, which means that the adversary's submitting a convincing response is different from an honest server's response.

However, the values $\{\alpha_1, \cdots, \alpha_n\}$ for every file are randomly chosen and therefore independent of the adversary's view. There is no other needed to consider the encryption in generating the tags, and the appearance is in computing $T_i = \sum_{j=1}^{n}(\alpha_j \cdot s_{i,j} + H(F_{id}||r||i)) \cdot r^j$, where $H(F_{id}||r||i)$ is a secure hash function. Since the output of δ is replaced by a random value r, T_i is independent of $\{\alpha_1, \cdots, \alpha_n\}$. Therefore, the probability that the bad event happens if the challenger first picks the random values $\{\alpha_1, \cdots, \alpha_n\}$ for each stored file and then undertakes the data possession checking interactions is the same as the probability that the bad event happens if the challenger first undertakes the data possession checking interactions and then chooses the value $\{\alpha_1, \cdots, \alpha_n\}$ for each file.

Consider the values $\Delta\rho_j$ and $\Delta\sigma$ in responses from the adversary and the choice of $\{\alpha_1, \cdots, \alpha_n\}$. The probability makes the equation $\Delta\sigma = \sum_{j=1}^{n} \alpha_j \cdot \Delta\rho_j \cdot r^j$ hold for a specific entry in an interaction is $1/p$. Therefore, the probability that the equation holds for a nonzero number of entries is at most q_P/p, where q_P is the number of RDPC protocol interactions initiated by the adversary. Thus, except with negligible probability q_P/p, the adversary never generates a convincing response which is different from an honest server's response, so the probability of the challenger aborts is negligible.

Wrapping Up. Yet we have argued that, assuming the PRF is secure and the symmetric encryption is semantic security, there is only a negligible difference in the success probability of the adversary in Game 3 compared to Game 0, where the adversary is not constrained in this manner. This completes the proof of Theorem 1.

6 Conclusion

In this paper, we presented a security analysis on a remote data possession checking protocol using algebraic signature in [20], and showed that it suffers from the replay attack and deletion attack. We also proposed an improved protocol by using the techniques of Shacham and Waters [6,7] to fix these security flaws. Involving the security model due to Ateniese et al [3], we provided formal security proofs of our new RDPC protocol.

References

1. Buyyaa, R., Yeoa, C., Broberga, J., Brandicc, I.: Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Generation Computer Systems 25(6), 599–616 (2009)
2. Zissis, D., Lekkas, D.: Addressing cloud computing security issues. Future Generation Computer Systems 28(3), 583–592 (2012)
3. Ateniese, G., Burns, R.C., Curtmola, R., Herring, J., Kissner, L., Peterson, Z.N.J., Song, D.: Provable data possession at untrusted stores. In: Proceeding of ACM CCS 2007, Alexandria, Virginia, USA, pp. 598–609. ACM (2007)
4. Ateniese, G., Burns, R.C., Curtmola, R., Herring, J., Kissner, L., Peterson, Z.N.J., Song, D.: Remote data checking using provable data possession. ACM Trans. Inf. Syst. Security 14(1), 12 (2011)

5. Juels, A., Kaliski, B.S.: PORs: proofs of retrievability for large files. In: Proceeding of ACM CCS 2007, Alexandria, Virginia, USA, pp. 584–597. ACM (2007)
6. Shacham, H., Waters, B.: Compact proofs of retrievability. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 90–107. Springer, Heidelberg (2008)
7. Shacham, H., Waters, B.: Compact proofs of retrievability. Journal of Cryptology 26(3), 442–483 (2013)
8. Ateniese, G., Kamara, S., Katz, J.: Proofs of storage from homomorphic identification protocols. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 319–333. Springer, Heidelberg (2009)
9. Wang, Q., Wang, C., Li, J., Ren, K., Lou, W.: Enabling public verifiability and data dynamics for storage security in cloud computing. In: Backes, M., Ning, P. (eds.) ESORICS 2009. LNCS, vol. 5789, pp. 355–370. Springer, Heidelberg (2009)
10. Wang, Q., Wang, C., Ren, K., Lou, W., Li, J.: Enabling public auditability and data dynamics for storage security in cloud computing. IEEE Trans. Parallel Distrib. Syst. 22(5), 847–859 (2012)
11. Wang, C., Ren, K., Lou, W., Li, J.: Toward public auditable secure cloud data storage services. IEEE Network 24(4), 19–24 (2010)
12. Zhu, Y., Hu, H., Ahn, G.J., Stephen, S.: Yau: efficient audit service outsourcing for data integrity in clouds. Journal of Systems and Software 85(5), 1083–1095 (2012)
13. Zhu, Y., Hu, H., Ahn, G.J., Yu, M.: Cooperative provable data possession for integrity verification in multicloud storage. IEEE Trans. Parallel Distrib. Syst. 23(12), 2231–2244 (2012)
14. Yang, K., Jia, X.: An efficient and secure dynamic auditing protocol for data storage in cloud computing. IEEE Trans. Parallel Distrib. Syst. 24(9), 1717–1726 (2013)
15. Zhu, Y., Wang, S.B., Hu, H., Ahn, G.J., Ma, D.: Secure collaborative integrity verification for hybrid cloud environments. Int. J. Cooperative Inf. Syst. 21(3), 165–198 (2012)
16. Wang, C., Chow, S.S.M., Wang, Q., Ren, K., Lou, W.: Privacy-preserving public auditing for secure cloud storage. IEEE Trans. Computers 62(2), 362–375 (2013)
17. Curtmola, R., Khan, O., Burns, R.: Robust remote data checking. In: Proceeding of Storage SS 2008, Fairfax, Virginia, USA, pp. 63–68. ACM (2008)
18. Bowers, K.D., Juels, A., Oprea, A.: Proofs of retrievability: theory and implementation. In: Proceeding of CCSW 2009, Chicago, Illinois, USA, pp. 43–54. ACM (2009)
19. Dodis, Y., Vadhan, S., Wichs, D.: Proofs of retrievability via hardness amplification. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 109–127. Springer, Heidelberg (2009)
20. Chen, L.: Using algebraic signatures to check data possession in cloud storage. Future Generation Computer Systems 29(7), 1709–1715 (2013)
21. Schwarz, T., Miller, E.: Store, forget, and check: using algebraic signatures to check remotely administered storage. In: Proceeding of ICDCS 2006, Lisbon, Portugal, p. 12. IEEE Computer Society (2006)
22. Ni, J., Yu, Y., Mu, Y., Xia, Q.: On the security of an efficient dynamic auditing protocol in cloud storage. IEEE Transactions on Parallel and Distributed Systems (2013), doi:10.1109/TPDS.2013.199
23. Erway, C., Kupcu, A., Papamanthou, C., Tamassia, R.: Dynamic provable data possession. In: Proceeding of ACM CC 2009, Hyatt Regency Chicago, Chicago, IL, USA, pp. 213–222. ACM (2009)
24. Ateniese, G., Pietro, R.D., Mancini, L.V., Tsudik, G.: Scalable and efficient provable data possession. In: Proceeding of SecureComm 2008, Stanbul, Turkey, pp. 1–10. IEEE Computer Society (2008)