Security enhancement of an auditing scheme for shared cloud data

Reyhaneh Rabaninejad* and
Mahmoud Ahmadian Attari

Department of Electrical Engineering,
K.N. Toosi University of Technology,
Tehran, Iran
Email: rabaninejad@ee.kntu.ac.ir
Email: m_ahmadian@kntu.ac.ir
*Corresponding author

Maryam Rajabzadeh Asaar

Department of Electrical and Computer Engineering,
Science and Research Branch,
Islamic Azad University,
Tehran, Iran
Email: asaar@srbiau.ac.ir

Mohammad Reza Aref

Department of Electrical Engineering,
Sharif University of Technology,
Tehran, Iran
Email: aref@sharif.edu

Abstract: In cloud storage services, public auditing mechanisms allow a third party to verify integrity of the outsourced data on behalf of data owners without the need to retrieve data from the cloud server. In some applications, the identity of data users should be kept private from the third party auditor. Oruta is a privacy preserving public auditing scheme for shared data in the cloud which exploits ring signatures to protect the identity privacy. In this paper, we propose two attacks and demonstrate that the scheme is insecure and a dishonest server can arbitrarily tamper the outsourced data without being detected by the auditor. We also propose a solution to remedy this weakness with the minimum overhead and without losing any desirable features of the scheme. Performance evaluation demonstrates acceptable efficiency of improved scheme in comparison to the original protocol.

Keywords: cloud storage; shared data; public auditing; security analysis.

Reference to this paper should be made as follows: Rabaninejad, R., Attari, M.A., Asaar, M.R. and Aref, M.R. (2022) 'Security enhancement of an auditing scheme for shared cloud data', *Int. J. Internet Protocol Technology*, Vol. 15, No. 1, pp.60–68.

Biographical notes: Reyhaneh Rabaninejad received the BSc and MSc degrees in electrical engineering from the Sharif University of Technology, Tehran, Iran. She is currently working toward the PhD degree in communications engineering, and is a member of Information Systems and Security Lab, Sharif University of Technology. Her research interests include network security and cloud computing security.

Mahmoud Ahmadian Attari received the PhD degree from the University of Manchester, UK, in electrical engineering. He has been a professor at K.N. Toosi University of Technology since 1979 where he founded the Cryptography and Coding Lab (CCL), in 2003. His research interests include multifunctional coding, and network security.

Maryam Rajabzadeh Asaar received the MSc and PhD degrees in electrical engineering from the Sharif University of Technology. She is an assistant professor at Department of Electrical and Computer Engineering, Science and Research Branch, Islamic Azad University. Her research interests include cryptographic protocols and network security.

Mohammad Reza Aref received the MSc and PhD degrees from Stanford University, in electrical engineering. He has been a professor of electrical engineering at the Sharif University of Technology since 1995. His research interests include information theory and cryptography, in which he has published more than 290 technical papers.

1 Introduction

Nowadays, efficient and scalable data storage services provided by cloud computing have caused a vast data movement from local storage systems to the cloud. However, security concerns including data integrity are a serious challenge for data owners after outsourcing. Provable Data Possession (PDP) protocols introduced by Ateniese et al. (2007) are mechanisms for efficient data integrity verification, in which a Third Party Auditor (TPA) chooses a random subset of data blocks and sends it as a challenge to the cloud server. The server generates a proof based on the challenged data blocks and their signatures and finally the TPA checks the proof that the server returns. PDP protocols have attracted significant research in recent years (e.g. Shacham and Waters, 2008; Wang, 2013; Wang et al., 2014; Tian et al., 2015; Li et al., 2016; Rabaninejad et al., 2020; Li et al., 2017; Wang et al., 2017; Yu et al., 2017; Rabaninejad et al., 2019; Rabaninejad et al., 2020).

Supporting efficient *dynamic* data operations is an important property in PDP protocols. The first dynamic PDP scheme was introduced by Ateniese et al. (2008). However, their scheme restricts number of auditing requests and also it only supports private verification. Wang et al. (2009) and Zhang et al. (2017) were the first who introduced PDP schemes with fully dynamic operations based on Merkle Hash Tree. Dynamic PDP scheme using the concept of rank information was researched by Erway et al. (2015). Index Hash Table was another tool to achieve dynamic data operations in PDP schemes presented by Zhu et al. (2011, 2013). Other very recent dynamic PDP proposals include (Shen et al., 2017; Yan et al., 2017; Cash et al., 2017; Rabaninejad et al., 2019).

Since the users may store sensitive data on the cloud, *data privacy* is also another important feature in PDP schemes. In other words, the TPA should not learn any information about the data content when he is auditing data integrity on behalf of users. Random masking technique proposed by Wang et al. (2013) addresses this problem. Yu et al. also designed an Identity-based PDP scheme with zero-knowledge data privacy (Yu et al., 2017).

With *data sharing* services such as Dropbox and Google Drive, data owners are able to conveniently share their data with a group of users. Oruta is a protocol proposed by Wang et al. (2014) which enables shared-data auditing. Oruta is based on ring signatures (Boneh et al., 2003) to provide *identity privacy*, which means that not only the content of shared-data, but also the identities of users are kept private from the public verifier. Yu et al. (2014) proposed an attack on Oruta. In their attack, an active adversary acts as the man in the middle attack and arbitrarily alters the data and also accordingly modifies the auditing proof generated by server such that the proof is verified by the TPA, although the data have been polluted. To fix this problem, they suggested that the server securely signs its proof, so that the active adversary is no more able to modify the proof generated by the server and therefore the corruption is detected by the TPA. Panda (Wang et al., 2013), is another shared-data auditing protocol proposed by Wang et al. which provides efficient user revocation. However, it lacks data and identity privacy properties which were provided by Oruta. Another shared data auditing scheme utilising polynomial-based authentication tags was proposed by Yuan and Yu (2014, 2015). In their scheme, user revocation is inefficient since it involves three parties including the group manager, the cloud server and the TPA (Yuan and Yu, 2015). Yu et al. (2016) showed that the scheme in Yuan and Yu (2015) is insecure against collusion of the cloud server and a revoked user. Jiang et al. (2016) addressed the problem of secure user revocation in shared data auditing by employing the group signatures (Boneh and Shacham, 2004). However, the scheme is inefficient due to the expensive computation cost of generating group signatures and costly auditing operations. Recently, Rabaninejad et al. proposed a lightweight public shared data auditing protocol which employs collusion resistant proxy re-signature and preserves users' identity privacy, enables efficient user revocation, and is secure against collusion of the server and revoked users (Rabaninejad et al., 2019).

In this paper, we propose two attacks on Oruta (Wang et al., 2014), named as *replace and replay attacks*, and show that a *dishonest server* can forge a proof on the corrupted data which passes the verification phase. In the replace attack that some data blocks are corrupted, the server can replace the auditing message such that the challenged set of blocks does not include the corrupted ones. Also, in the replay attack that some blocks are not fresh, the server can generate the proof based on the old data blocks and in both cases the generated proof passes the verification equation. We note that the improvement suggested in Yu et al. (2014) does not make Oruta secure against our attacks. Specifically, since the cloud server is the attacker itself, therefore it can sign the forged proof generated based on corrupted data and the TPA still verifies the forged proof. In the next step, we develop a simple improvement in the scheme without losing any desirable features and demonstrate that this improvement makes Oruta secure against the proposed attacks.

2 The Oruta protocol

In this section, we get a glimpse of the Oruta scheme (Wang et al., 2014). Three parties are involved in the protocol: cloud server, third party auditor (TPA) and data users. Let U denote the group of authorised users with d members who have access to the shared data and can modify it. Shared data blocks and their signatures are outsourced to the cloud server, and the TPA, on behalf of users, audits the integrity of the data stored in the cloud.

Setup. Let G_1, G_2 and G_T be multiplicative cyclic groups of prime order p, with g_1 and g_2 as generators of G_1 and G_2, respectively. Let $e: G_1 \times G_2 \to G_T$ be a bilinear map, and $\psi: G_2 \to G_1$ be an isomorphism such that $\psi(g_2) = g_1$. Also three hash functions $H_1: \{0,1\}^* \to G_1$, $H_2: \{0,1\}^* \to Z_q$ and $h: G_1 \to Z_p$ are used in the scheme. We note that q is a prime much smaller than p. In order to outsource shared data M, it is divided into n blocks as $M = (m_1, ..., m_n)$. Then, each block m_j is also divided into k sectors in Z_p. So, M can be represented as a $n \times k$ matrix. Oruta consists of five algorithms which are reviewed in the following:

keyGen. For user $u_i \in U$, the public and private key pair is: $(pk, sk) = (w_i = g_2^{x_i}, x_i)$, where x_i is chosen randomly from Z_p. The original user also generates a public aggregate key $pak = (\eta_1, ..., \eta_k)$, where $\eta_l \in_r G_1$.

SigGen. Using this algorithm, a user u_s generates a ring signature on a block m_j. Consider block $m_j = (m_{j,1}, ..., m_{j,k})$ with identifier id_j. The signer u_s, using her private key sk_s, public aggregate key pak, and the public key of the other $d-1$ users, does the following steps:

1 Aggregate block m_j using pak:

$$\beta_j = H_1(id_j)\prod_{l=1}^{k}\eta_l^{m_{j,l}} \in G_1. \quad (1)$$

2 The ring signature of block m_j is $\sigma_j = (\sigma_{j,1}, ..., \sigma_{j,d}) \in G_1^d$. For all $i \neq s$, u_s chooses random $a_{j,i} \in Z_p$, and sets $\sigma_{j,i} = g_1^{a_{j,i}}$. For $i = s$, u_s calculates $\sigma_{j,s}$ using equation (2).

$$\sigma_{j,s} = \left(\frac{\beta_j}{\psi(\prod_{i \neq s} w_i^{a_{j,i}})}\right)^{1/x_s} \in G_1 \quad (2)$$

Modify. A user in U can insert, delete or update the j-th block of the data as described below. The identifier of block m_j is defined as, $id_j = \{v_j, r_j\}$, where v_j is the virtual index of the block and $r_j = H_2(m_j \| v_j)$.

- **Insert.** The user, in order to insert a new block $m_{j'}$ into the outsourced data, computes $v_{j'} = (v_{j-1} + v_j)/2$, $r_{j'} = H_2(m_{j'} \| v_{j'})$, and sets the new block identifier as $id_{j'} = \{v_{j'}, r_{j'}\}$. Next, he signs $m_{j'}$ and sends $\{m_{j'}, id_{j'}, \sigma_{j'}\}$ to the server.
- **Delete.** To delete a block m_j, the user only requires to send the block index to the server. Accordingly, the server deletes $\{m_j, id_j, \sigma_j\}$ from its storage.
- **Update.** The user, in order to update the j-th block with a new value $m_{j'}$, computes $r_{j'} = H_2(m_{j'} \| v_j)$ and sets $id_{j'} = \{v_j, r_{j'}\}$, where the virtual index v_j remains unchanged. Next, he signs $m_{j'}$ and sends $\{m_{j'}, id_{j'}, \sigma_{j'}\}$ to the server.

ProofGen. In order to audit the data integrity, the TPA first chooses a random c-element subset $J \subset [1, n]$ as the block indices to be challenged in the auditing process. Then for each $j \in J$, the TPA chooses a random value $y_j \in Z_q$ and sends the auditing message $\{(j, y_j)\}_{j \in J}$ to the server.

The server, after receiving the auditing message $\{(j, y_j)\}_{j \in J}$, generates an auditing proof through the following procedure:

1 For $l \in [1, k]$, the server calculates $\lambda_l = \eta_l^{r_l} \in G_1$, where $r_l \in_r Z_q$ and η_l is a part of public aggregate key pak.

2 For $l \in [1, k]$, the server computes a linear function of challenged blocks as $\mu_l = \sum_{j \in J} y_j m_{j,l} + r_l h(\lambda_l) \in Z_p$.

3 The server, aggregates the challenged block's signatures as $\phi_i = \prod_{j \in J} \sigma_{j,i}^{y_j}$, for $i \in [1, d]$.

Finally, the server sends back the auditing proof $\{\lambda, \mu, \phi, \{id_j\}_{j \in J}\}$ to the TPA, where $\lambda = (\lambda_1, ..., \lambda_k)$, $\mu = (\mu_1, ..., \mu_k)$ and $\phi = (\phi_1, ..., \phi_d)$.

ProofVerify. The TPA verifies the server's proof $\{\lambda, \mu, \phi, \{id_j\}_{j \in J}\}$ through equation (3). If the equation holds, the verification passes, otherwise it fails. The verification is done using the auditing message $\{(j, y_j)\}_{j \in J}$, the public aggregate key and all the users public keys.

$$e(\prod_{j \in J} H_1(id_j)^{y_j} . \prod_{l=1}^{k} \eta_l^{\mu_l}, g_2) \overset{?}{=} (\prod_{i=1}^{d} e(\phi_i, w_i)).e(\prod_{l=1}^{k} \lambda_l^{h(\lambda_l)}, g_2) \quad (3)$$

3 Security analysis of Oruta

Two kinds of threats related to the integrity of shared data are considered in Wang et al. (2014). First, an adversary may try to corrupt the integrity of shared data. Second, the cloud server may inadvertently corrupt (or even remove) data in its storage due to hardware failures or human errors. Making matters worse, the cloud server tries to hide such data corruptions from the users in order to save its reputation and avoid losing profits of its services. In this section, we propose two attacks and show that a dishonest server can forge a proof on the corrupted data which passes verification in equation (3). We then propose a simple fix to make Oruta secure against the proposed attacks.

3.1 Replace attack

Assume that a block m_t of the data stored at the cloud server is corrupted. Upon receiving the auditing message $\{(j, y_j)\}_{j\in J}$, where $t \in J$, the server replaces the set J with a set J^* such that $t \notin J^*$ –i.e., replaces the corrupted block m_t with another intact block in J^* –, and generates auditing proof for the auditing message $\{(j, y_j)\}_{j\in J^*}$, with the same y_j values as the original auditing message. More specifically, the auditing proof is generated as below:

1 For $l \in [1, k]$, λ_l values are calculated as before.

2 For $l \in [1, k]$, the server computes $\mu_l^* = \sum_{j\in J^*} y_j m_{j,l} + r_l h(\lambda_l)$.

3 For $i \in [1, d]$, $\phi_i^* = \prod_{j\in J^*} \sigma_{j,i}^{y_j}$.

Therefore, the only difference with the normal auditing proof explained in Section 2, is that J is replaced with J^* in Steps 2 and 3. Finally, the server returns $\{\lambda, \mu^*, \phi^*, \{id_j\}_{j\in J^*}\}$ as the auditing proof to the TPA. Next, the TPA checks equation (4):

$$e\Big(\prod_{j\in J} H_1(id_j)^{y_j} . \prod_{l=1}^{k} \eta_l^{\mu_l^*}, g_2\Big) \overset{?}{=} \Big(\prod_{i=1}^{d} e(\phi_i^*, w_i)\Big) . e\Big(\prod_{l=1}^{k} \lambda_l^{h(\lambda_l)}, g_2\Big) \quad (4)$$

It can be seen that because the indices $j \in J$ are not directly used in the verification equation, and the coefficients y_j used by the server are the same as the original auditing message, the proof generated for the set J^* passes the TPA's verification. So, the server could successfully make the equation hold while the data have been corrupted.

3.2 Replay attack

As mentioned before, Oruta supports dynamic operations and a user can insert, delete or update a block in shared data stored at the cloud server. The second attack arises here when a user wants to update the t^{th} block with a new value $m_{t'}$ and sends $\{m_{t'}, id_{t'}, \sigma_{t'}\}$ to the cloud server. Consider a scenario that an adversary interrupts the user–server line and this update request is not received by the server. Also, consider a scenario that the server inadvertently does not follow data update requests due to hardware failures or human errors or even denial of service happens when a burst of update requests are received by the cloud. These scenarios are possible according to threat model of Oruta paper (Wang et al., 2014). Now, we show that in case these scenarios occur, the dishonest server can generate a valid proof in response to the TPA based on the old values $\{m_t, id_t, \sigma_t\}$. Assume that the auditing message $\{(j, y_j)\}_{j\in J}$, where $t \in J$ is sent to the server. The server generates a proof for this auditing message, but based on the old version of t^{th} block as follows:

1 For $l \in [1, k]$, the server calculates λ_l values as before.

2 For $l \in [1, k]$, $\mu_l^* = \sum_{j\in J} y_j m_{j,l} + r_l h(\lambda_l)$; where for $j = t$, $m_{t,l}$ is considered (instead of $m'_{t,l}$).

3 For $i \in [1, d]$, $\phi_i^* = \prod_{j\in J} \sigma_{j,i}^{y_j}$; where for $j = t$, $\sigma_{t,i}$ is considered (instead of $\sigma'_{t,i}$).

The auditing proof returned by the server is $\{\lambda, \mu^*, \phi^*, \{id_j\}_{j\in J}\}$, where for $j = t$, id_t is sent (instead of id'_t). The TPA checks equation (5).

$$e\Big(\prod_{j\in J} H_1(id_j)^{y_j} . \prod_{l=1}^{k} \eta_l^{\mu_l^*}, g_2\Big) \overset{?}{=} \Big(\prod_{i=1}^{d} e(\phi_i^*, w_i)\Big) . e\Big(\prod_{l=1}^{k} \lambda_l^{h(\lambda_l)}, g_2\Big) \quad (5)$$

Based on the above equation, the TPA is convinced by the server that the data integrity is not violated; while in fact the data is not fresh and the update process in not followed properly.

4 An improved protocol

In this section, we first describe an improved protocol to resolve the weakness discussed in the previous section. Next, we discuss how the improved protocol withstands the aforementioned attacks. Finally, we evaluate the performance of improved protocol.

4.1 Improved Oruta

The weakness causes these attacks work, is that in Oruta, the TPA has no information about the verification metadata and this enables the server to fool the TPA in the verification process. Here, we suggest a simple fix to overcome the weakness. The idea is to leverage an authenticated data structure like Merkle Hash Tree (MHT) (Wang et al., 2011; Rabaninejad et al., 2019), where the updated MHT root is sent to the TPA. In the following, we first review MHT authentication structure and then provide a detailed explanation of scheme modifications.

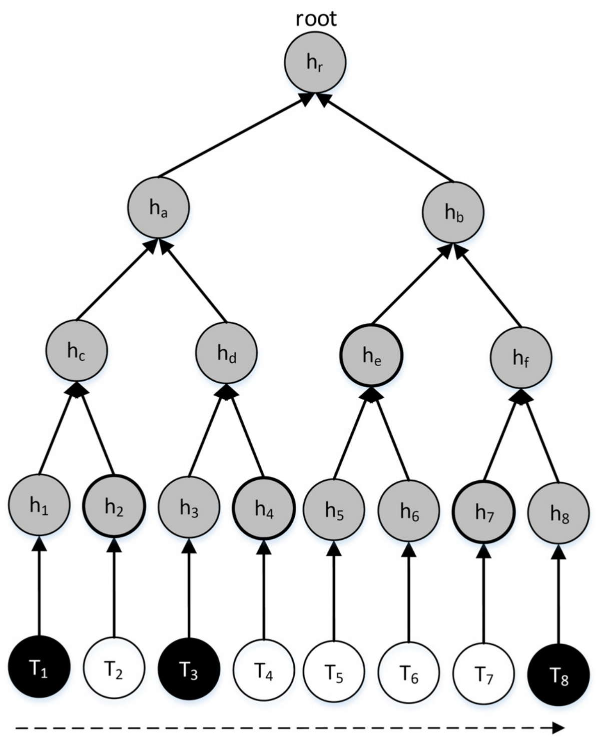
– Merkle Hash Tree

A Merkle Hash Tree (MHT) (Merkle, 1980) is a tree in which leaf nodes are labelled with the hashes of authentic elements. Also, every non-leaf node is the cryptographic hash of its child nodes, as shown in Figure 1. For example, consider a verifier who possess the root h_r requests to authenticate the values $\{T_1, T_3, T_8\}$. The prover returns $\{T_1, T_3, T_8\}$ together with $\Delta_1 = h_2$, $\Delta_3 = h_4$, and $\Delta_8 = \{h_e, h_7\}$ as Auxiliary Authentication Information (AAI). The verifier uses AAI to check the integrity of $\{T_1, T_3, T_8\}$ as follows:

$$h_1 = h(T_1), h_3 = h(T_3), h_8 = h(T_8)$$
$$h_c = h(h_1, h_2), h_d = h(h_3, h_4), h_f = h(h_7, h_8)$$
$$h_a = h(h_c, h_d), h_b = h(h_e, h_f)$$
$$h_{r'} = h(h_a, h_b)$$

Finally, if $h_{r'} = h_r$, it is proved that $\{T_1, T_3, T_8\}$ are unaltered. It should be noted that MHT leaves are considered as a left-to-right sequence, which makes it possible to authenticate both values and positions of the data blocks.

Figure 1 An example of data authentication using MHT



– **Scheme modifications.** The improved protocol is as follows. Here, we only emphasise on modifications and the unmodified parts are referred to the original protocol to avoid repetition:

Setup. $H(.)$ is a hash function employed in MHT authenticated data structure and is defined as $H:\{0,1\}^* \rightarrow Z_q$. Other parameters are chosen the same as Setup algorithm of the original protocol.

keyGen. This algorithm is the same as the original protocol.

SigGen. The signature $\sigma_j = (\sigma_{j,1}, ..., \sigma_{j,d}) \in G_1^d$ on block m_j with identifier id_j is generated exactly the same as the original protocol. The difference here is that a MHT authenticated data structure is produced with the nodes as hash of identifiers $\{id_j\}_{j\in[1,n]}$ related to the data file $M = (m_1, ..., m_n)$. Also, $(root, \sigma_{root})$ respectively denotes MHT root and its signature, which are simultaneously forwarded to both the server and the TPA. For each new block (m_j, id_j), the MHT root is re-computed and the new σ_{root} is forwarded to both the server and the TPA, so that they replace σ_{root} as the last version of the MHT root in their storage.

Modify.

- **Insert.**

(a) The user, in order to insert a new block $m_{j'}$ into j^{th} position of the outsourced data, computes $v_{j'} = (v_{j-1} + v_j)/2$, $r_{j'} = H_2(m_{j'} \| v_{j'})$, and sets the new block identifier as $id_{j'} = \{v_{j'}, r_{j'}\}$. Next, he signs $m_{j'}$ and sends $\{m_{j'}, id_{j'}, \sigma_{j'}\}$ to the server. This step was the same as the original protocol. But other steps are different.

(b) The server inserts $\{m_{j'}, id_{j'}, \sigma_{j'}\}$ in the j^{th} row and also inserts $H(id_{j'})$ as the j^{th} leaf of the MHT,

(c) The server regenerates MHT root $root^*$ and forwards $(\Delta_j, id_j, \sigma_{root}, root^*)$ to the user.

(d) The user first uses id_j and its AAI Δ_j to compute $root$, and checks whether the returned σ_{root} is a valid signature on $root$. Second, he computes $root^{*\prime}$ and checks whether $root^{*\prime} = root^*$. If it is so, he generates σ_{root^*} and forwards $(root^*, \sigma_{root^*})$ to both the server and the TPA. The insert operation is accomplished.

- **Delete.**

(a) The user sends a request to delete a block m_j from the outsourced data.

(b) Upon receiving delete request, the server deletes j^{th} row $\{m_j, id_j, \sigma_j\}$ and updates the MHT using the same procedure explained in Insert operation.

- **Update.**

(a) The user sends $\{m_{j'}, id_{j'}, \sigma_{j'}\}$ to the server, as a request to update the j-th block with a new value $m_{j'}$.

(b) Upon receiving update request, the server updates the j^{th} row and also updates the MHT using the same procedure explained in Insert operation.

ProofGen. In order to audit the data integrity, the TPA first chooses a random c-element subset $J \subset [1,n]$ as the block indices to be challenged in the auditing process. Then for each $j \in J$, the TPA chooses a random value $y_j \in Z_q$ and sends the auditing message $\{(j, y_j)\}_{j\in J}$ to the server.

Upon receiving the auditing message $\{(j, y_j)\}_{j\in J}$ from the TPA, the server generates an auditing proof through the following procedure:

1 For $l \in [1,k]$, the server calculates $\lambda_l = \eta_l^{r_l} \in G_1$, where $r_l \in_r Z_q$ and η_l is a part of public aggregate key pak.

2 For $l \in [1,k]$, the server computes a linear function of challenged blocks as $\mu_l = \sum_{j\in J} y_j m_{j,l} + r_l h(\lambda_l) \in Z_p$.

3 The server, aggregates the challenged block's signatures as $\phi_i = \prod_{j\in J} \sigma_{j,i}^{y_j}$, for $i \in [1,d]$.

Finally, the server sends back the auditing proof $\{\lambda, \mu, \phi, \{id_j, \Delta_j\}_{j\in J}\}$ to the TPA, where $\lambda = (\lambda_1,\ldots,\lambda_k)$, $\mu = (\mu_1,\ldots,\mu_k)$ and $\phi = (\phi_1,\ldots,\phi_d)$. Besides, Δ_j is the set of Auxiliary Authentication Information (AAI) corresponding to the challenged nodes in MHT.

ProofVerify. This algorithm includes two following steps:

1 In this step, the TPA uses parameters $\{id_j, \Delta_j\}_{j\in J}$ which are returned by the server as part of the proof. Specifically, the TPA calculates MHT root $root'$ using the nodes $\{H(id_j)\}_{j\in J}$ and its AAI set $\{\Delta_j\}_{j\in J}$. If $root'$ is equal to $root$ (the last MHT root stored by the TPA), it means that the block identifiers $\{id_j\}_{j\in J}$ returned by the server are the exact values corresponding to the challenged blocks. Consequently, this step prevents replace/replay attacks which will be discussed in the security subsection.

2 The TPA checks equation (6) to verify the correctness of the proof $\{\lambda, \mu, \phi, \{id_j\}_{j\in J}\}$. If the equation holds, the verification passes, otherwise it fails.

$$e\Big(\prod_{j\in J} H_1(id_j)^{y_j}.\prod_{l=1}^{k}\eta_l^{\mu_l}, g_2\Big) \overset{?}{=} \Big(\prod_{i=1}^{d} e(\phi_i, w_i)\Big).e\Big(\prod_{l=1}^{k}\lambda_l^{h(\lambda_l)}, g_2\Big) \quad (6)$$

We note that the second step is the same as ProofVerify algorithm of the original scheme but the first step is different. More precisely, in the first step, the TPA uses parameters $\{id_j, \Delta_j\}_{j\in J}$ exactly in the same indices $\{j\}_{j\in J}$ it had challenged the server, in order to generate MHT root $root'$. In case the values $\{id_j, \Delta_j\}_{j\in J}$ are not related to the indices $\{j\}_{j\in J}$ (replace attack) or are not the updated values (replay attack), root $root'$ calculated by the TPA is is different from $root$ (the last MHT root stored by the TPA) and the verification fails. In other words, Step 1 fixes the weakness in the basic scheme since Merkle Hash Tree is the verification metadata that helps the TPA to detect a forged proof.

4.2 *Security of improved Oruta*

Here, we discuss how improved Oruta resists against replace/replay attacks proposed in Section 3.

Security against replace Attack. In this attack, the cloud server replaced corrupted block m_t with another intact block, lets name it m_{t^*}, and generated auditing proof for the auditing message $\{(j, y_j)\}_{j\in J^*}$. So, the set J^* is equal to set J except that index t is replaced with t^*. The server returns $P = \{\lambda, \mu^*, \phi^*, \{id_j, \Delta_j\}_{j\in J^*}\}$ as the auditing proof to the TPA. Now, we show that Step 1 of ProofVerify algorithm described above fails: the TPA calculates MHT root $root'$ using the values $\{H(id_j)\}_{j\in J^*}$ as the MHT nodes and the AAI set $\{\Delta_j\}_{j\in J^*}$, which are returned by the server. Technically, the TPA puts the values $\{H(id_j)\}_{j\in J^*}$ in $j \in J$ positions of the merkle hash tree nodes. Therefore, $H(id_t^*)$ is placed as the t^{th} node. Obviously, $root'$ is not equal to $root$ (the real MHT root), since $H(id_t^*)$ is not equal to $H(id_t)$ (the correct value for t^{th} node of the MHT). Therefore, before the TPA gets to check equation (6) to verify the correctness of the proof in Step 2, Step 1 of ProofVerify algorithm fails and the replace attack is detected in the improved protocol.

Security against replay Attack. The argument here is the same with slight differences. Replay attack considered a case when a block m_t is not properly updated according to scenarios explained in 3.2. In case that the cloud server received an auditing message $\{(j, y_j)\}_{j\in J}$, where $t \in J$, the server generated a proof based on the old version of block m_t and returned auditing proof $\{\lambda, \mu^*, \phi^*, \{id_j, \Delta_j\}_{j\in J}\}$, where for $j = t$, id_t is sent (instead of updated id_t'). Now, we show that Step 1 of ProofVerify algorithm described above fails: the TPA calculates MHT root $root'$ using the values $\{H(id_j)\}_{j\in J}$ as the MHT nodes and the AAI set $\{\Delta_j\}_{j\in J}$, which are returned by the server. Obviously, $root'$ is not equal to $root$ (the real MHT root), since $H(id_t)$ (the old value) is not equal to $H(id_t')$ (the correct updated value for t^{th} node of the MHT). Therefore, before the TPA gets to check equation (6) to verify the correctness of the proof in

Step 2, Step 1 of ProofVerify algorithm fails and the replay attack is detected in the improved protocol.

To summarise, if the server replaces the challenged blocks with other blocks (replace attack) or replays the old versions of the blocks (replay attack) in generating the auditing proof, the MHT root computed by the TPA will be different from the real MHT root, and the verification fails. Therefore, the improved protocol is secure against the proposed attacks without losing any desirable features.

4.3 Performance evaluation

In terms of performance, the improved Oruta protocol has employed Merkle Hash Tree (MHT) in its construction in comparison to original Oruta protocol. Here, through a detailed overhead analysis, we demonstrate that the computation/ communication costs caused by MHT are acceptable for the users, the server, and the TPA. Hence, improved Oruta achieves comparable efficiency to the original Oruta protocol, but with enhanced security level.

Table 1 presents computation overhead for the users, the cloud server and the TPA and also communication overhead, for both Oruta and Improved Oruta schemes. In this table, the computational complexity of hash function, modular multiplications, exponentiations, and pairings are denoted by H, Mul, Exp, and $Pair$, respectively. Also, the index of the operation denotes the group that the operation is defined in, for example Mul_{G_1} means multiplication over G_1. Furthermore, as additions have negligible cost, they are omitted from overhead analysis.

First column of Table 1 represents overhead analysis of the original Oruta protocol. An explanation on how these overheads are calculated can be found in Rabaninejad et al. (2019). For improved Oruta which is denoted in the second column, the terms 'Sign. Gen. Complexity' (Row 1) and 'Server Comp. Complexity' (Row 2) are equal to the ones for Oruta protocol, which is a direct result of protocol description in Subsection 4.1, since these parts are exactly the same as the original protocol. For 'Verifier Comp. Complexity' (Row 3), Step 1 is added to ProofVerify algorithm of the improved protocol, in which the TPA computes MHT root. For a tree with n leaves, each root computation includes about $log_2 n$ hash evaluations, where n is the total number of data blocks and $log_2 n$ equals to the tree depth. Since cost of computing an ordinary hash function which maps an arbitrary string to elements in $\mathbb{Z}_q$, is negligible in comparison to the other operations, this term has not a big impact on computational cost of TPA. Technically, assume the schemes are implemented on a personal computer (Intel I5-3470 3.20 GHz processor, 4 GB memory and Windows 7 operating system) using MIRACL library (see http://www.shamus.ie/index.php?page=home), an ordinary hash function takes 0.053 milliseconds, while pairing takes 11.515 milliseconds. For example, for 1000,000 blocks, the root computation cost is about $log_2 1000,000 \times 0.053 = 20 \times 0.053 = 1.06$, which is ignorable to $(d+2)Pair$ operations required in the verification process. Finally, for 'Comm. Complexity' (Row 4), in improved protocol the server responds the values of challenged nodes and their siblings $\{id_j, \Delta_j\}_{j\in J}$ for MHT root authentication in the auditing process. Hence, Com_{MHT} is the communication cost introduced by the term $\{id_j, \Delta_j\}_{j\in J}$. For a tree with n leaves, each Δ_j contains at most $log_2 n$ nodes equal to the tree depth. Therefore, the maximum cost of this term is $Com_{MHT} = c(|q|\,log_2 n + |q|)$. All in all, it can be observed from Table 1 that the overall performance of the two schemes are comparable to each other and the slight extra overhead in improved protocol is the cost paid for enhanced security level.

Table 1 Performance comparison

Scheme / Metric	*Oruta (Wang et al., 2014)*	*Improved Oruta (this paper)*
Sign. Gen. Complexity	$1H+(d+k)Mul_{G_1}+(2d+k)Exp_{G_1}$	$1H+(d+k)Mul_{G_1}+(2d+k)Exp_{G_1}$
Server Comp. Complexity	$kH+(ck)Mul_{\mathbb{Z}_q}$ $+(k+dc)Exp_{G_1}+(dc)Mul_{G_1}$	$kH+(ck)Mul_{\mathbb{Z}_q}+(k+dc)Exp_{G_1}+(dc)Mul_{G_1}$
Verifier Comp. Complexity	$cH+dMul_{G_2}+(d+2)Pair$ $+(2k+c)Exp_{G_1}+(2k+c)Mul_{G_1}$	$cH+dMul_{G_2}+(d+2)Pair+(2k+c)Exp_{G_1}+(2k+c)Mu$
Comm. Complexity	$(2k+d)\|p\|+c(2\|q\|+\|n\|)$	$(2k+d)\|p\|+c(2\|q\|+\|n\|)+Com_{MHT}$

Notes: Parameters *d*, *k* and *c* denote size of the group *U*, number of elements per block and number of challenged blocks in an auditing task, respectively.

5 Conclusion

In this paper, we investigated Oruta, a privacy preserving public auditing protocol for shared data in the cloud, and showed that by applying the replace and replay attacks, a dishonest server can cheat and hide data corruption from the TPA's view. We next improved Oruta to fix the weakness and made it secure against the proposed attacks. Our solution enjoys the desirable features of the original protocol with acceptable computation and communication overheads according to our performance evaluation.

Acknowledgement

This work was partially supported by Iran NSF under Grant No. 96.53979.

References

Ateniese, G., Burns, R., Curtmola, R., Herring, J., Kissner, L., Peterson, Z. and Song, D. (2007) 'Provable data possession at untrusted stores', *Proceedings of the 14th ACM conference on Computer and communications security, ACM 2007*, pp.598–609.

Ateniese, G., Di Pietro, R., Mancini, L.V. and Tsudik, G. (2008) 'Scalable and efficient provable data possession', *Proceedings of the 4th international conference on Security and privacy in communication netowrks*, ACM.

Boneh, D. and Shacham, H. (2004) 'Group signatures with verifier-local revocation', *Proceedings of the 11th ACM conference on Computer and communications security*, ACM, pp.168–177.

Boneh, D., Gentry, C., Lynn, B. and Shacham, H. (2003) 'Aggregate and verifiably encrypted signatures from bilinear maps', *International Conference on the Theory and Applications of Cryptographic Techniques: Advances in Cryptology (EUROCRYPT'03)*, Springer, pp.416–432.

Cash, D., Küpçü, A. and Wichs, D. (2017) 'Dynamic proofs of retrievability via oblivious ram', *Journal of Cryptology*, Vol. 30, No. 1, pp.22–57.

Erway, C., Küpçü, A., Papamanthou, C. and Tamassia, R. (2015) 'Dynamic provable data possession', *ACM Transactions on Information and System Security (TISSEC)*, Vol. 17, No. 4, p.15.

Jiang, T., Chen, X. and Ma, J. (2016) 'Public integrity auditing for shared dynamic cloud data with group user revocation', *IEEE Transactions on Computers*, Vol. 65, No. 8, pp.2363–2373.

Li, F., Xie, D., Gao, W., Chen, K., Wang, G. and Metere, R. (2017) 'A certificateless signature scheme and a certificateless public auditing scheme with authority trust level 3+', *Journal of Ambient Intelligence and Humanized Computing*, pp.1–10.

Li, J., Zhang, L., Liu, J.K., Qian, H. and Dong, Z. (2016) 'Privacy-preserving public auditing protocol for low-performance end devices in cloud', *IEEE Transactions on Information Forensics and Security*, Vol. 11, No. 11, pp.2572–2583.

Merkle, R.C. (1980) 'Protocols for public key cryptosystems', *Security and Privacy, 1980 IEEE Symposium on*, pp.122–122.

Rabaninejad, R., Ahmadian, M., Asaar, M.R. and Aref, M.R. (2019) 'A lightweight auditing service for shared data with secure user revocation in cloud storage', *IEEE Transactions on Services Computing*, doi:10.1109/TSC.2019.2919627.

Rabaninejad, R., Asaar, M.R., Attari, M.A. and Aref, M.R. (2019) 'An identity-based online/offline secure cloud storage auditing scheme', *Cluster Computing*, pp.1–14, doi:10.1007/s10586-019-03000-5.

Rabaninejad, R., Attari, M.A., Asaar, M.R. and Aref, M.R. (2019) 'Comments on a lightweight cloud auditing scheme: Security analysis and improvement', *Journal of Network and Computer Applications*, Vol. 139, pp.49–56, doi:10.1016/j.jnca.2019.04.012.

Rabaninejad, R., Attari, M.A., Asaar, M.R. and Aref, M.R. (2020) 'A lightweight identity-based provable data possession supporting users' identity privacy and traceability', *Journal of Information Security and Applications*, Vol. 51, 102454, doi:10.1016/j.jisa.2020.102454.

Shacham, H. and Waters, B. (2008) 'Compact proofs of retrievability', *International Conference on the Theory and Application of Cryptology and Information Security*, pp.90–107.

Shen, J., Shen, J., Chen, X., Huang, X. and Susilo, W. (2017) 'An efficient public auditing protocol with novel dynamic structure for cloud data', *IEEE Transactions on Information Forensics and Security*, Vol. 12, No. 10, pp.2402–2415.

Tian, H., Chen, Y., Chang, C., Jiang, H., Huang, Y., Chen, Y. and Liu, J. (2015) 'Dynamic-hash-table based public auditing for secure cloud storage', *IEEE Transactions on Services Computing*, Vol. 10, No. 5, pp.701–714.

Wang, B., Li, B. and Li, H. (2013) 'Panda: public auditing for shared data with efficient user revocation in the cloud', *IEEE Transactions on Services Computing*, Vol. 8, No. 1, pp.92–106.

Wang, B., Li, B. and Li, H. (2014) 'Oruta: privacy-preserving public auditing for shared data in the cloud', *IEEE Transactions on Cloud Computing*, Vol. 2, No. 1, pp.43–56.

Wang, B., Li, H., Liu, X., Li, F. and Li, X. (2014) 'Efficient public verification on the integrity of multi-owner data in the cloud', *Journal of Communications and Networks*, Vol. 16, No. 6, pp.592–599.

Wang, C., Chow, S.S.M., Wang, Q., Ren, K. and Lou, W. (2013) 'Privacy-preserving public auditing for secure cloud storage', *IEEE Transactions on computers*, Vol. 62, No. 2, pp.362–375.

Wang, H. (2013) 'Proxy provable data possession in public clouds', *IEEE Transactions on Services Computing*, Vol. 6, No. 4, pp.551–559.

Wang, Q., Wang, C., Li, J., Ren, K. and Lou, W. (2009) 'Enabling public verifiability and data dynamics for storage security in cloud computing', *Computer Security–ESORICS 2009*, pp.355–370.

Wang, Q., Wang, C., Ren, K., Lou, W. and Li, J. (2011) 'Enabling public auditability and data dynamics for storage security in cloud computing', *IEEE Transactions on Parallel and Distributed Systems*, Vol. 22, No. 5, pp.847–859.

Wang, Y., Wu, Q., Qin, B., Tang, S. and Susilo, W. (2017) 'Online/offline provable data possession', *IEEE Transactions on Information Forensics and Security*, Vol. 12, No. 5, pp.1182–1194.

Yan, H., Li, J., Han, J. and Zhang, Y. (2017) 'A novel efficient remote data possession checking protocol in cloud storage', *IEEE Transactions on Information Forensics and Security*, Vol. 12, No. 1, pp.78–88.

Yu, Y., Au, M.H., Ateniese, G., Huang, X., Susilo, W., Dai, Y. and Min, G. (2017) 'Identity-based remote data integrity checking with perfect data privacy preserving for cloud storage', *IEEE Transactions on Information Forensics and Security*, Vol. 12, No. 4, pp.767–778.

Yu, Y., Li, Y., Ni, J., Yang, G., Mu, Y. and Susilo, W. (2016) 'Comments on public integrity auditing for dynamic data sharing with multiuser modification', *IEEE Transactions on Information Forensics and Security*, Vol. 11, No. 3, pp.658–659.

Yu, Y., Niu, L., Yang, G., Mu, Y. and Susilo, W. (2014)' On the security of auditing mechanisms for secure cloud storage', *Future Generation Computer Systems*, Vol. 30, pp.127–132.

Yuan, J. and Yu, S. (2014) 'Efficient public integrity checking for cloud data sharing with multi-user modification', *INFOCOM, 2014 Proceedings IEEE*, pp.2121–2129.

Yuan, J. and Yu, S. (2015) 'Public integrity auditing for dynamic data sharing with multiuser modification', *IEEE Transactions on Information Forensics and Security*, Vol. 10, No. 8, pp.1717–1726.

Zhang, H., Tu, T. et al. (2017) 'Dynamic outsourced auditing services for cloud storage based on batch-leaves-authenticated merkle hash tree', *IEEE Transactions on Services Computing*.

Zhu, Y., Ahn, G., Hu, H., Yau, S.S., An, H.G. and Hu, C. (2013) 'Dynamic audit services for outsourced storages in clouds', *IEEE Transactions on Services Computing*, Vol. 6, No. 2, pp.227–238.

Zhu, Y., Wang, H., Hu, Z., Ahn, G., Hu, H. and Yau, S.S. (2011) 'Dynamic audit services for integrity verification of outsourced storages in clouds', *Proceedings of the 2011 ACM Symposium on Applied Computing*, pp.1550–1557.