Future Generation Computer Systems 30 (2014) 127–132

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

On the security of auditing mechanisms for secure cloud storage

Yong Yu[a,b,*], Lei Niu[a], Guomin Yang[b], Yi Mu[b], Willy Susilo[b]

[a] School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu, 610054, PR China

[b] Centre for Computer and Information Security Research, School of Computer Science and Software Engineering, University of Wollongong, Wollongong, NSW 2522, Australia

HIGHLIGHTS

- Identify a new kind of attack on secure cloud auditing protocols.
- Show two identity privacy-preserving auditing mechanisms called Oruta and Knox are insecure against this new attack.
- Discuss the security of a distributed storage integrity auditing mechanism in our attack.

ARTICLE INFO

Article history:
Received 31 December 2012
Received in revised form
3 April 2013
Accepted 17 May 2013
Available online 27 May 2013

Keywords:
Cloud storage
Privacy-preserving
Auditing
Cryptanalysis

ABSTRACT

Cloud computing is a novel computing model that enables convenient and on-demand access to a shared pool of configurable computing resources. Auditing services are highly essential to make sure that the data is correctly hosted in the cloud. In this paper, we investigate the active adversary attacks in three auditing mechanisms for shared data in the cloud, including two identity privacy-preserving auditing mechanisms called Oruta and Knox, and a distributed storage integrity auditing mechanism. We show that these schemes become insecure when active adversaries are involved in the cloud storage. Specifically, an active adversary can arbitrarily alter the cloud data without being detected by the auditor in the verification phase. We also propose a solution to remedy the weakness without sacrificing any desirable features of these mechanisms.

1. Introduction

Cloud storage, an important service of cloud computing, allows users to move data from their local storage systems to the cloud and enjoy the on-demand high quality cloud services. It offers great convenience to users since they do not have to care about the complexities of direct hardware and software managements. Besides, with cloud storage, data sharing is realized efficiently among a large number of users in a group and it becomes a standard feature in most cloud storage offerings, including Dropbox and Google Docs.

Although cloud storage provides many appealing benefits for users, it also prompts a number of security issues towards the outsourced data [1,2]. The data stored on the cloud is easily be corrupted, modified or deleted due to hardware failure or human errors, thus, protecting the correctness and integrity of the data in the cloud is highly essential. To achieve this goal, two novel approaches called provable data possession (PDP) [3] and proofs of retrievability (POR) [4] were proposed. In 2007, Ateniese et al. [3] proposed, for the first time, the notion of PDP to check the integrity of the data stored at untrusted servers, and presented a public auditing scheme using RSA-based homomorphic linear authenticators. They also described a publicly verifiable scheme, which allows any third party to challenge the server for data possession. To support dynamic data operations, Ateniese et al. proposed a scalable PDP [5] based on hash function and symmetric key encryption. However, in this scheme, the numbers of update and challenge are limited and need to be prefixed and block insertion is not allowed. Subsequently, Erway et al. developed two dynamic PDP protocols [6] based on hash trees. Juels et al. [4] proposed a POR model to ensure both data possession and retrievability. Unfortunately, this mechanism prevents efficient extension for updating data. Shacham and Waters [7] described two solutions for ensuring the integrity of remote data. The first scheme makes use of pseudorandom functions and supports private auditing, while the second one allows public auditing and is based on BLS short signature [8]. Based on the BLS short signature, Wang et al. [9] presented data integrity checking approaches to achieve public auditability, storage correctness, privacy-preserving, batch auditing,

* Corresponding author at: Centre for Computer and Information Security Research, School of Computer Science and Software Engineering, University of Wollongong, Wollongong, NSW 2522, Australia. Tel.: +61 242214043.
E-mail address: yyucd2012@gmail.com (Y. Yu).

0167-739X/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.future.2013.05.005

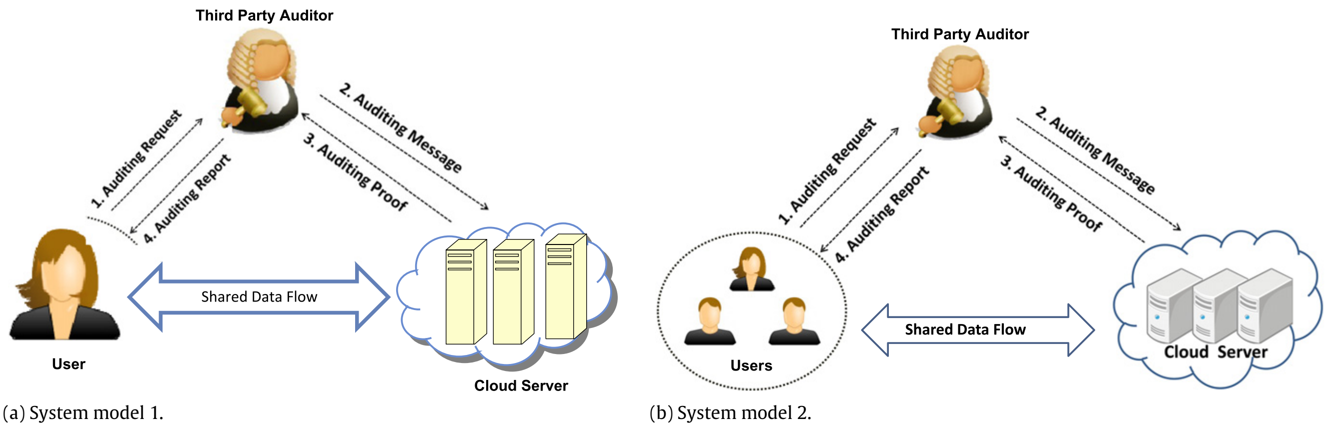

Fig. 1. The system model of auditing mechanisms for secure cloud storage.

lightweight, dynamic data support and error location and recovery. Since then, several other auditing mechanisms such as [10–14] have been proposed for protecting the integrity of the outsourced data.

Most of the existing solutions only focus on auditing the integrity of the remote data. However, privacy preserving is highly essential during the auditing process. Wang et al. [15] proposed a privacy-preserving public auditing mechanism, in which the content of users' data is not disclosed to the auditor. Recently, Wang et al. observed that preserving identity privacy from the auditor during the auditing process is also essential since the identities of users may indicate that a particular user in the group or a special block in the shared data is a more valuable target than others. They also proposed two identity privacy-preserving auditing mechanisms, called Oruta [16] and Knox [17], for secure cloud storage. In Oruta, ring signatures [18] based homomorphic authenticators are employed such that the auditor can verify the integrity of the shared data for a group of users without retrieving the entire data, while the identity of the user on each block on the shared data is kept confidential from the auditor. A drawback of Oruta is that the size of the signatures and auditing proofs are linearly increasing with the number of the users in a group. Moreover, when a new user is added to the group, all the signatures have to be re-generated. Knox leveraged group signature [19] based homomorphic authenticators to solve the problem, and also shortened the size of authenticators and the auditing proofs while preserving the properties of identity privacy-preserving, public auditing and batch auditing.

In this paper, we revisit three auditing mechanisms for secure cloud storage, including two identity privacy-preserving mechanisms [16,17] and a distributed storage integrity auditing mechanism [9]. We show that the property of correctness cannot be achieved when active adversaries are involved in these auditing systems. More specifically, an active adversary can arbitrarily tamper the cloud data and produce a valid auditing response to pass the auditor's verification. As a consequence, the adversary can fool the auditor to believe that the data in the cloud are well-maintained while in fact the data have been corrupted. We also propose a solution to resolve the weakness in these schemes.

2. Preliminaries

In this section, we review some basic knowledge, including the bilinear map and the system models.

2.1. Bilinear maps

Let $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ be three multiplicative cyclic groups of prime order p, g_1 and g_2 be the generators of $\mathbb{G}_1$ and $\mathbb{G}_2$. ψ is a computable isomorphism from G_2 to G_1, with $\psi(g_2) = g_1$. The map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is said to be an admissible bilinear pairing if the following conditions hold true [20].

(1) e is bilinear, i.e. $e({g_1}^a, {g_2}^b) = e(g_1, g_2)^{ab}$ for all $a, b \in Z_p$.
(2) e is non-degenerate, i.e. $e(g_1, g_2) \neq 1_{\mathbb{G}_T}$.
(3) e is efficiently computable.

2.2. Review of the system model

Three participants, namely the cloud server, the third party auditor and users are involved in a cloud auditing system [16,17]. The cloud server has ample storage space and provides data storage and sharing services for users. The auditor has expertise and capabilities to provide data auditing service based on requests from users, without downloading the entire file. In a general cloud storage system model, we only consider a client, an auditor and an untrusted cloud server as shown in Fig. 1(a). While in an identity privacy-preserving auditing mechanisms for secure cloud storage, there are two types of users in a group as shown in Fig. 1(b): an original user who is the creator of the shared data, and a number of group users who can access and modify the data. When a user wishes to check the integrity of the outsourced data, she sends an auditing request to the auditor. Upon receiving the request, the auditor generates an auditing message to the cloud server, and gets an auditing proof of the data from the server. After verifying the correctness of the proof, the auditor forwards an auditing report to inform the user the integrity of the outsourced data.

3. On the security two identity privacy-preserving auditing mechanisms

In this section, we briefly review two identity privacy-preserving auditing mechanisms called Oruta and Knox, and present the security analysis of these auditing mechanism in the presence of active adversaries.

3.1. Review and analysis of Oruta

The shared data M is divided into n blocks, and each block m_j is further split into k sectors, which are elements in Z_p. Thus, M can be denoted as $M = \{m_{j,l}\}_{j\in[1,n],l\in[1,k]}$. Three secure hash functions $H_1 : \{0, 1\}^* \to G_1$, $H_2 : \{0, 1\}^* \to Z_p$ and $h : G_1 \to Z_p$ are employed. The global parameters are $(e, \psi, p, G_1, G_2, G_T, g_1, g_2, H_1, H_2, h)$. Let U denote the user group that has d users.

KenGen: a user $u_i \in U$ randomly picks $x_i \in Z_p$ as his secret key and computes the corresponding public key $w_i = g_2^{x_i}$. The original user also randomly generates a public aggregate key $pak = (\eta_1, \dots, \eta_k)$, where η_l for $l \in [1, k]$ are random elements in G_1.

SigGen: given d group members' public keys $(w_1, \ldots, w_d)$, a block $m_j = (m_{j,1}, \ldots, m_{j,k})$, the identifier id_s and a private key sk_s for some user u_s. u_s computes a ring signature based homomorphic authenticator of block m_j as follows:

(1) compute the aggregation of block m_j as

$$\beta_j = H_1(\mathrm{id}_j) \cdot \prod_{l=1}^{k} \eta_l^{m_{j,l}} \in G_1.$$

(2) for $i \neq s$, randomly pick $a_{j,i} \in Z_p$ and set $\sigma_{j,i} = g_1^{a_{j,i}}$. Then compute $\sigma_{j,s} = (\frac{\beta_j}{\psi(\prod_{i\neq s} w_i^{a_{j,i}})})^{1/x_s} \in G_1$. The ring signature based homomorphic authenticator of block m_j is $\sigma_j = (\sigma_{j,1}, \ldots, \sigma_{j,d})$.

Modify: this algorithm performs the insert, delete, update operations for a user in the group to modify the j-th block in the shared data. We refer readers to [16] for the details of these operations.

ProofGen: the auditor first picks a random c-element subset $\mathcal{J}$ of $[1, n]$. For each $j \in \mathcal{J}$, the auditor picks a random $y_j \in Z_p$ and sends an auditing message $\{(j, y_j)\}_{j\in\mathcal{J}}$ to the cloud server.

Upon receiving an auditing message $\{(j, y_j)\}_{j\in\mathcal{J}}$, the cloud server generates a responding proof for the selected blocks as follows.

(1) Choose a random element $r_l \in Z_q$, and compute $\lambda_l = \eta_l^{r_l} \in G_1$, for $l \in [1, k]$.
(2) Compute $\mu_l = \sum_{j\in\mathcal{J}} y_j m_{j,l} + r_l h(\lambda_l) \in Z_p$, for $l \in [1, k]$.
(3) Aggregate the signatures as $\phi_i = \prod_{j\in\mathcal{J}} \sigma_{j,i}^{y_j}$, for $i \in [1, d]$.

The cloud server sends an auditing proof $\{\lambda, \mu, \phi, \{\mathrm{id}_j\}_{j\in\mathcal{J}}\}$ to the auditor, where $\lambda = (\lambda_1, \ldots, \lambda_k)$, $\mu = (\mu_1, \ldots, \mu_k)$ and $\phi = (\phi_1, \ldots, \phi_d)$.

ProofVerify: with the auditing proof $\{\lambda, \mu, \phi, \{\mathrm{id}_j\}_{j\in\mathcal{J}}\}$, the auditing message $\{(j, y_j)\}_{j\in\mathcal{J}}$, the public aggregate key *pak* and all the group members' public keys $(pk_1, \ldots, pk_d)$, the auditor verifies the validity of the proof by checking the following equation:

$$e\left(\prod_{j\in\mathcal{J}} H_1(\mathrm{id}_j)^{y_j} \cdot \prod_{l=1}^{k} \eta_l^{\mu_l}, g_2\right) \stackrel{?}{=} \left(\prod_{i=1}^{d} e(\phi_i, w_i)\right) \cdot e\left(\prod_{l=1}^{k} \lambda_l^{h(\lambda_l)}, g_2\right).$$

If the equation holds, the auditor outputs 1; Otherwise, outputs 0. And then, the auditor forwards the auditing results to the user.

Security analysis of Oruta: As the first identity privacy-preserving public auditing mechanism for shared data in the cloud, Oruta enjoys many desirable properties of a cloud auditing system including correctness, unforgeability, identity privacy and data privacy, and it can also be extended to support batch auditing. Informally, correctness means that the auditor is able to correctly detect whether there is any corrupted block in shared data [16]. Regarding the correctness of the outsourced data, two kinds of threats were considered in Oruta. First, an adversary may try to corrupt the integrity of the shared data and prevent users from using data correctly. Second, the cloud server may inadvertently corrupt or even remove data in its storage due to hardware failures and human errors. However, below we show that when an active adversary, such as a bug planted in the software running on the cloud server by a malicious programmer or a hacker, is involved in the auditing process, Oruta would fail to achieve the property of correctness. Specifically, the adversary can arbitrarily modify or tamper the outsourced data and fool the auditor to believe the data are well preserved in the cloud. All the information the adversary has to know is how the data are modified. The details of this analysis are presented below.

Assume that the adversary modifies each sector $m_{j,l}$ to $m^*_{j,l} = m_{j,l}+d_{j,l}$ for $j \in [1, n]$, $l \in [1, k]$ and records the values $d_{j,l}$, i.e., how the shared data are modified. In the auditing process, the auditor sends the auditing message $\{(j, y_j)\}_{j\in\mathcal{J}}$ to the cloud server. Upon receiving $\{(j, y_j)\}_{j\in\mathcal{J}}$, the cloud server honestly executes the auditing mechanism to compute $\{\lambda, \mu^*, \phi\}$, where $\lambda = (\lambda_1, \ldots, \lambda_k)$, $\mu^* = (\mu^*_1, \ldots, \mu^*_k)$ and $\phi = (\phi_1, \ldots, \phi_d)$ in the following way.

For $l \in [1, k]$, the cloud server firstly chooses a random $r_l \in Z_q$, and then computes $\lambda_l = \eta_l^{r_l}$ and

$$\begin{aligned} \mu^*_l &= \sum_{j\in\mathcal{J}} y_j m^*_{j,l} + r_l h(\lambda_l) \\ &= \sum_{j\in\mathcal{J}} y_j (m_{j,l} + d_{j,l}) + r_l h(\lambda_l) \\ &= \sum_{j\in\mathcal{J}} y_j m_{j,l} + r_l h(\lambda_l) + \sum_{j\in\mathcal{J}} y_j d_{j,l} \\ &= \mu_l + \sum_{j\in\mathcal{J}} y_j d_{j,l}. \end{aligned}$$

The cloud server aggregates the signatures as $\phi_i = \prod_{j\in\mathcal{J}} \sigma_{j,i}^{y_j}$, for $i \in [1, d]$, and sends the auditing proof $\{\lambda, \mu^*, \phi, \{\mathrm{id}_j\}_{j\in\mathcal{J}}\}$ to the auditor.

The adversary intercepts the auditing proof $\{\lambda, \mu^*, \phi, \{\mathrm{id}_j\}_{j\in\mathcal{J}}\}$ from the cloud server to the auditor, and modifies each μ^*_l to

$$\mu_l = \mu^*_l - \sum_{j\in\mathcal{J}} y_j d_{j,l}$$

for $l \in [1, k]$. By performing such a modification, the adversary derives a correct proof with respect to the original data blocks m_j for $\{(j, y_j)\}_{j\in\mathcal{J}}$, and sends it to the auditor. As a result, in *ProofVerify* phase, the modified auditing proof can make the equation hold and, thus the auditor believes that the blocks in shared data are all well-maintained, while the data have been polluted by the adversary. In this way, the adversary can make Oruta lose correctness.

3.2. Knox and its security analysis

In this section, we briefly review Knox [17], another identity privacy-preserving auditing mechanism for secure cloud storage based on homomorphic authenticable group signature (HAGS) [17]. We show that it also fails to meet the correctness property in the presence of an active adversary and readers are referred to [17] for the full description of the scheme.

Brief review of Knox: let $M = \{m_{j,l}\}_{j\in[1,n],l\in[1,k]}$ denote the outsourced data and d the number of users in the group. $H_1 : \{0, 1\}^* \to Z_p$ and $H_2 : \{0, 1\}^* \to G_1$ are two cryptographic hash functions. Knox also employs a pseudorandom generator PRG: $\mathcal{K}_{\mathrm{prg}} \to Z_p^k$ and a pseudorandom function PRF: $\mathcal{K}_{\mathrm{prf}} \times \mathcal{I} \to Z_p$, where $\mathcal{K}_{\mathrm{prg}}$ and $\mathcal{K}_{\mathrm{prf}}$ are the key spaces of the PRG and the PRF respectively, and $\mathcal{I}$ is the set of all data block identifiers in the index hash table of M. The global parameters of Knox are $(e, \psi, p, G_1, G_2, G_T, g_1, g_2, H_1, H_2,$ PRG, PRF$)$.

KeyGen: the original user randomly picks a secret key pair $skp = (sk_{\mathrm{prg}}, sk_{\mathrm{prf}}) \in \mathcal{K}_{\mathrm{prg}} \times \mathcal{K}_{\mathrm{prf}}$ which will be shared with the auditor. The group public key $gpk = (g_1, g_2, h, w, v, w, \rho, \eta)$ and the original user's master key $gmsk = (\xi_1, \xi_2, \gamma)$ are defined as in HAGS [17].

Join: the private key of user i is $gsk[i] = (A_i, x_i, \pi, skp)$, generated as in HAGS. The original user forwards $gsk[i]$ to user i in a secure manner and logs the user into the group user list.

Sign: given a group public key gpk, a private key $gsk[i]$, a block $m_j \in Z_p^k$ and this block's identifier $\mathrm{id}_j \in \mathcal{I}$, user i computes the signature σ_j as follows.

(1) Select $\alpha_j, \beta_j, r_{j,\alpha}, r_{j,\beta}, r_{j,x}, r_{j,\gamma_1}, r_{j,\gamma_2}$ and compute $T_{j,1}, T_{j,2}, T_{j,3}, \gamma_{j,1}, \gamma_{j,2}, R_{j,1}, R_{j,2}, R_{j,3}, R_{j,4}, R_{j,5}$ as in HAGS.
(2) Compute $\delta = (\delta_1, \ldots, \delta_k) \leftarrow \mathrm{PRG}(sk_{\mathrm{prg}}) \in Z_p^k$ and $b_j \leftarrow \mathrm{PRF}(sk_{\mathrm{prf}}, \mathrm{id}_j) \in Z_p$, then calculate the homomorphic MAC of block $m_j = (m_{j,1}, \ldots, m_{j,k})$ as:

$$t_j = \sum_{l=1}^{k} \delta_l \cdot m_{j,l} + b_j \in Z_p.$$

(3) Compute a challenge c_j for m_j as:

$$c_j = \eta^{t_j} \cdot H_1(T_{j,1}, \ldots, R_{j,5}) \in Z_p.$$

(4) Compute $s_{j,\alpha}, s_{j,\beta}, s_{j,x}, s_{j,\gamma_1}, s_{j,\gamma_2}$ as in HAGS.
(5) Compute a tag θ_j as $\theta_j = [H_2(\mathrm{id}_j)g_1^{t_j}]^{\pi} \in G_1$.
(6) Output a signature σ_j of this block m_j as $\sigma_j = (T_{j,1}, T_{j,2}, T_{j,3}, \theta_j, R_{j,3}, c_j, s_{j,\alpha}, s_{j,\beta}, s_{j,x}, s_{j,\gamma_1}, s_{j,\gamma_2})$.

ProofGen: the auditor firstly picks a random c-element subset $\mathcal{J}$ of set $[1, n]$. For $j \in \mathcal{J}$, the auditor picks a random $y_j \in Z_p$ and sends an auditing message $\{(j, y_j)\}_{j\in\mathcal{J}}$ to the cloud server. Upon receiving $\{(j, y_j)\}_{j\in\mathcal{J}}$, the cloud server generates a proof for the selected blocks as follows.

(1) Compute $\mu_l = \sum_{j\in\mathcal{J}} y_j m_{j,l} \in Z_p$, for $l \in [1, k]$, and aggregate the selected tags as $\Theta = \prod_{j\in\mathcal{J}} \theta_j^{y_j} \in G_1$.
(2) Output Φ and $\phi_j = (T_{j,1}, T_{j,2}, T_{j,3}, R_{j,3}, c_j, s_{j,\alpha}, s_{j,\beta}, s_{j,x}, s_{j,\gamma_1}, s_{j,\gamma_2})$ based on σ_j, where $j \in \mathcal{J}$ and Φ is the set of all ϕ_j.
(3) Generate an auditing proof $\{\mu, \Theta, \Phi, \{\mathrm{id}_j\}_{j\in\mathcal{J}}\}$, and send it to the auditor, where $\mu = (\mu_1, \ldots, \mu_k)$.

ProofVerify: given an auditing proof $\{\mu, \Theta, \Phi, \{\mathrm{id}_j\}_{j\in\mathcal{J}}\}$, an auditing message $\{(j, y_j)\}_{j\in\mathcal{J}}$, a group public key gpk, a secret key pair skp, the auditor verifies the correctness of this proof as follows.

(1) Generate $\delta = (\delta_1, \ldots, \delta_k) \leftarrow \mathrm{PRG}(sk_{\mathrm{prg}}) \in Z_p^k$ and $b_j \leftarrow \mathrm{PRF}(sk_{prf}, \mathrm{id}_j) \in Z_p$, where $j \in \mathcal{J}$.
(2) Re-compute $R_{j,1}, R_{j,2}, R_{j,4}, R_{j,5}$ as in HAGS.
(3) Compute $\lambda = \sum_{l=1}^{k} \delta_l \mu_l + \sum_{j\in\mathcal{J}} y_j b_j \in Z_p$.
(4) Check the following equations:

$$\prod_{j\in\mathcal{J}} R_{j,3}^{y_j} \overset{?}{=} e\left(\prod_{j\in\mathcal{J}} (T_{j,3}^{s_{j,x}} \cdot h^{-s_{j,\gamma_1}-s_{j,\gamma_2}} \cdot g_1^{-c_j})^{y_j}, g_2\right) \cdot e\left(\prod_{j\in\mathcal{J}} (h^{-s_{j,\alpha}-s_{j,\beta}} \cdot T_{j,3}^{c_j})^{y_j}, w\right).$$

$$\prod_{j\in\mathcal{J}} c_j^{y_j} \overset{?}{=} \eta^{\lambda} \cdot \prod_{j\in\mathcal{J}} H_1(T_{j,1}, \ldots, R_{j,5})^{y_j}.$$

$$e(\Theta, g_2) \overset{?}{=} e\left(\prod_{j\in\mathcal{J}} H_2(\mathrm{id}_j)^{y_j} \cdot g_1^{\lambda}, \rho\right).$$

If all the equations hold, the auditor outputs 1; Otherwise, outputs 0. The auditor then forwards the auditing result to the user.

Open: given a block m_j and a signature σ, the original user can reveal the identity of the signer on this block using his private key $gmsk$ as in HAGS.

Security analysis of Knox: Similar to the analysis for Oruta, we show that Knox is also vulnerable to the pollution attacks from active adversaries, who can tamper the outsourced data without being detected by the auditor. Assume that the adversary modifies each element $m_{j,l}$ to $m_{j,l}^* = m_{j,l} + d_{j,l}$ for $j \in [1, n]$, $l \in [1, k]$ and records the modifications $d_{j,l}$. In the auditing process, after receiving the auditing message $\{(j, y_j)\}_{j\in\mathcal{J}}$ from the auditor, the cloud server firstly computes μ_l^* for $l \in [1, k]$ as follows,

$$\mu_l^* = \sum_{j\in\mathcal{J}} y_j(m_{j,l} + d_{j,l}) = \mu_l + \sum_{j\in\mathcal{J}} y_j d_{j,l},$$

and $\Theta = \prod_{j\in\mathcal{J}} \theta_j^{y_j}$. Then it sends the auditing proof $\{\mu^*, \Theta, \Phi, \{\mathrm{id}_j\}_{j\in\mathcal{J}}\}$ back to the auditor, in which $\mu^* = (\mu_1^*, \ldots, \mu_k^*)$ and Φ is the set of all $\phi_j = (T_{j,1}, T_{j,2}, T_{j,3}, R_{j,3}, c_j, s_{j,\alpha}, s_{j,\beta}, s_{j,x}, s_{j,\gamma_1}, s_{j,\gamma_2})$, for $j \in \mathcal{J}$.

The adversary intercepts the auditing proof $\{\mu^*, \Theta, \Phi, \{\mathrm{id}_j\}_{j\in\mathcal{J}}\}$ and modifies each μ_l^* to $\mu_l = \mu_l^* - \sum_{j\in\mathcal{J}} y_j d_{j,l}$, for $l \in [1, k]$. By performing such a modification, the adversary derives a correct proof with respect to the original data $m_{j,l}$ for $j \in \mathcal{J}$ and $l \in [1, k]$. As a consequence, the proof can pass the auditing verification, which makes the auditor believe that the outsourced data are well-maintained by the server, while in fact the data have been corrupted.

3.3. A solution to the security issue

We provide a solution to address the security issue in Oruta and Knox by applying a secure digital signature scheme. Specifically, in Oruta, the *KeyGen* algorithm outputs an additional key pair (sk_S, pk_S) for the cloud server. In the auditing process, before responding the auditing proof $\{\lambda, \mu, \phi, \{\mathrm{id}_j\}_{j\in\mathcal{J}}\}$ to the auditor, the server uses its secret key sk_S to compute a signature Σ for the proof and sends $\{\lambda, \mu, \phi, \{\mathrm{id}_j\}_{j\in\mathcal{J}}, \Sigma\}$ as the response to the challenge. Upon receiving the response, the auditor first verifies whether the signature Σ is valid or not. If it is valid, the auditor performs the *ProofVerify* protocol; Otherwise, the auditor discards the response. The same solution can be applied to Knox. In the improved versions of Oruta and Knox, it is hard for an adversary to modify the auditing proof any more in the auditing process because of the employment of the digital signatures. Therefore, if the shared data have been modified, the auditor must be able to detect it. Moreover, since only a digital signature is introduced in the auditing process, all the merits of Oruta and Knox are still preserved.

4. Security discussions on a distributed storage auditing mechanism

In this section, we review the distributed storage integrity auditing mechanism in [9], and discuss its security in the context of active adversaries. Some notions are as follows.

- **F**: the data file to be stored. **F** can be denoted as a matrix of m equal-sized data vectors, each consisting of l blocks.
- **A**: the dispersal matrix used for Reed–Solomon coding.
- **G**: the encoded file matrix, which includes a set of $n = m + k$ vectors, each consisting of l blocks.
- $f_{\mathrm{key}}(\cdot)$: the pseudorandom function (PRF), which is defined as $f : \{0, 1\}^* \times \mathrm{key} \rightarrow GF(2^p)$.
- $\phi_{\mathrm{key}}(\cdot)$: the pseudorandom permutation (PRP), which is defined as $\phi : \{0, 1\}^{\log_2(l)} \times \mathrm{key} \rightarrow \{0, 1\}^{\log_2(l)}$.
- *ver*: a version number bound with the index for individual blocks, which records the times the block has been modified.
- s_{ij}^{ver}: the seed for PRF, which depends on the file name, block index i, the server position j as well as the optional block version number *ver*.

4.1. Review of the scheme

The main scheme in [9] is composed of the following three algorithms.

File distribution preparation: Let $\mathbf{F} = (F_1, F_2, \ldots, F_m)$ and $F_i = (f_{1i}, f_{2i}, \ldots, f_{li})^T$, $(i \in 1, \ldots, m)$. T denotes that each F_i is represented as a column vector, and l denotes data vector size in blocks.

The information dispersal matrix $\mathbf{A}$, derived from an $m \times (m+k)$ Vandermonde matrix:

$$\begin{bmatrix} 1 & 1 & \cdots & 1 & 1 & \cdots & 1 \\ \beta_1 & \beta_2 & \cdots & \beta_m & \beta_{m+1} & \cdots & \beta_n \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \beta_1^{m-1} & \beta_2^{m-1} & \cdots & \beta_m^{m-1} & \beta_{m+1}^{m-1} & \cdots & \beta_n^{m-1} \end{bmatrix}$$

where $\beta_j (j \in 1, \ldots, n)$ are the distinct elements randomly picked from $GF(2^w)$.

After a sequence of elementary row transformations, the desired matrix $\mathbf{A}$ can be written as follows:

$$\mathbf{A} = (\mathbf{I}|\mathbf{P}) = \begin{bmatrix} 1 & 0 & \cdots & 0 & p_{11} & \cdots & p_{1k} \\ 0 & 1 & \cdots & 0 & p_{21} & \cdots & p_{2k} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & p_{m1} & \cdots & p_{mk} \end{bmatrix}.$$

By multiplying $\mathbf{F}$ by $\mathbf{A}$, the user obtains the encoded file:

$$\begin{aligned} \mathbf{G} &= \mathbf{F} \cdot \mathbf{A} \\ &= (G^{(1)}, G^{(2)}, \ldots, G^{(m)}, G^{(m+1)}, \ldots, G^{(n)}) \\ &= (F_1, F_2, \ldots, F_m, G^{(m+1)}, \ldots, G^{(n)}) \end{aligned}$$

where $G^{(j)} = (g_1^{(j)}, g_2^{(j)}, \ldots, g_l^{(j)})^T (j \in 1, \ldots, n)$.

Token precomputation: suppose the user wants to challenge the server t times, he has to precompute t verification tokens for each $G^{(j)} (j \in 1, \ldots, n)$, using a PRF $f_{\text{key}}(\cdot)$, a PRP $\phi_{\text{key}}(\cdot)$, a challenge key k_{chal} and a master permutation key K_{PRP}.

For server j, the user generates the ith token as follows:

(1) Derive a random challenge value α_i of $GF(2^p)$ by $\alpha_i = f_{k_{\text{chal}}}(i)$ and a permutation key $k_{\text{prp}}^{(i)}$ based on K_{PRP}.
(2) Compute the set of r randomly-chosen indices $\{I_q \in [1, \ldots, l] | 1 \le q \le r\}$, where $I_q = \phi_{k_{\text{prp}}^{(i)}}(q)$.
(3) Calculate the token as $v_i^{(j)} = \sum_{q=1}^{r} \alpha_i^q * G^{(j)}[I_q]$, where $G^{(j)}[I_q] = g_{I_q}^{(j)}$.

Correctness verification: the ith challenge-response for checking over the n servers acts as follows:

(1) the user reveals the α_i as well as the ith permutation key $k_{\text{prp}}^{(i)}$ to each server.
(2) the server storing vector $G^{(j)} (j \in 1, \ldots, n)$ aggregates those r rows specified by index $k_{\text{prp}}^{(i)}$ into a linear combination

$$R_i^{(j)} = \sum_{q=1}^{r} \alpha_i^q * G^{(j)}[\phi_{k_{\text{prp}}^{(i)}}(q)]$$

and sends back $R_i^{(j)} (j \in 1, \ldots, n)$.
(3) upon receiving $R_i^{(j)}$ from all the servers, the user takes away blind values in $R_i^{(j)} (j \in m+1, \ldots, n)$ by $R_i^{(j)} \leftarrow R_i^{(j)} - \sum_{q=1}^{r} f_{k_j}(s_{I_q,j}) \cdot \alpha_i^q$, where $I_q = \phi_{k_{\text{prp}}^{(i)}}(q)$.
(4) the user verifies whether the received values remain a valid codeword determined by the secret matrix $\mathbf{P}$:

$$(R_i^{(1)}, \ldots, R_i^{(m)}) \cdot \mathbf{P} = (R_i^{(m+1)}, \ldots, R_i^{(n)}).$$

If the above equation holds, the challenge is passed. Otherwise, it shows that among those specified rows, there exist file block corruptions.

4.2. A security discussion on the scheme

Similar to the analysis for Oruta and Knox, an active adversary $\mathcal{A}$ can arbitrarily modify the data blocks without knowing the actual data blocks, but at the same time fool the user to believe that the data are well maintained by the cloud server. The details are described as follows.

(1) $\mathcal{A}$ chooses an $l \times n$ matrix $\mathbf{Y}$, whose elements are $y_q^{(i)} \in GF(2^p)$, $(1 \le q \le l, 1 \le j \le n)$.
(2) $\mathcal{A}$ modifies the data blocks $G^{(j)}[\phi_{k_{\text{prp}}^{(i)}}(q)]$ to $G^{(j)}[\phi_{k_{\text{prp}}^{(i)}}(q)] + y_q^{(i)}$ for $1 \le q \le r$.
(3) In the audit phase, the user and the servers execute the protocol honestly, that is, the user reveals the α_i as well as the ith permutation key $k_{\text{prp}}^{(i)}$ to each server and the server computes the response $R_i^{(j)'} (j \in 1, \ldots, n)$ and sends it back to the user, where

$$\begin{aligned} R_i^{(j)'} &= \sum_{q=1}^{r} \alpha_i^q * (G^{(j)}[\phi_{k_{\text{prp}}^{(i)}}(q)] + y_q^{(i)}) \\ &= \sum_{q=1}^{r} \alpha_i^q * (G^{(j)}[\phi_{k_{\text{prp}}^{(i)}}(q)]) + \sum_{q=1}^{r} (\alpha_i^q * y_q^{(i)}) \\ &= R_i^{(j)} + \sum_{q=1}^{r} (\alpha_i^q * y_q^{(i)}). \end{aligned}$$

(4) $\mathcal{A}$ intercepts the response $R_i^{(j)'}$ from the cloud server to the auditor, and modifies $R_i^{(j)'}$ to $R_i^{(j)} = R_i^{(j)'} - \sum_{q=1}^{r} (\alpha_i^q * y_q^{(i)})$ and forwards $R_i^{(j)}$ to the user.

It is easy to check that the verification will be successful. Fortunately, authentication is presumed in [9]. The point-to-point communication channels between each cloud server and the user is assumed to be authenticated and reliable. We argue that this is highly essential. Otherwise, the mechanism may be insecure against an active attack as described above. During the implementation in reality, the server can employ a secure digital signature to achieve the goal, as suggested in the previous section.

5. Conclusion

In this paper, we revisited three auditing mechanisms for shared data in the cloud, including two identity privacy-preserving auditing mechanisms and a distributed storage integrity auditing mechanism. We demonstrate that if the cloud server does not authenticate its response, an active adversary can launch an attack to violate the storage correctness. Specifically, the adversary can arbitrarily alter the cloud data without being detected by the auditor in the verification phase. It seems that this kind of attack was not considered in the previous proposals, and fortunately, the authors of [9] mentioned that reliable channels between cloud server and users are required but with no concrete deployment. We suggested using a secure digital signature scheme to fix the problem without sacrificing any desirable feature of the original mechanisms.

Acknowledgements

The authors would like to thank the editors and anonymous reviewers for their valuable comments of the manuscript. The first author is supported by the University of Wollongong Vice Chancellor Fellowship and the fifth author is supported by Australian Research Council Future Fellowship FT0991397. This work was supported by the National Natural Science Foundation of China under Grants 61003232, 61103207, U1233108, 61272436, the National Research Foundation for the Doctoral Program of Higher Education of China under Grants 20100185120012, the National Natural Science Foundation of China for International Young Scientists under Grant 61250110543, and the Fundamental Research Funds for the Central Universities under Grant ZYGX2010J066.

References

[1] M.T. Khorshed, A.B.M. Ali, S.A. Wasimi, A survey on gaps, threat remediation challenges and some thoughts for proactive attack detection in cloud computing, Future Gener. Comput. Syst. 28 (6) (2012) 833–851.
[2] Kui Ren, Cong Wang, Qian Wang, Security challenges for the public cloud, IEEE Internet Comput. 16 (1) (2012) 69–73.
[3] G. Ateniese, R.C. Burns, R. Curtmola, J. Herring, L. Kissner, Z.N.J. Peterson, D.X. Song, Provable data possession at untrusted stores, in: ACM Conference on Computer and Communications Security 2007, pp. 598–609.
[4] A. Juels, J. Burton, S. Kaliski, PORs: Proofs of retrievability for large files, in: Proc. of CCS 07, pp. 584–597.
[5] G. Ateniese, R.D. Pietro, L.V. Mancini, G. Tsudik, Scalable and efficient provable data possession, in: Proc. of SecureComm 2008, pp. 1–10.
[6] C.C. Erway, A. Kupcu, C. Papamanthou, R. Tamassia, Dynamic provable data possession, Proc. of CCS 2009, pp. 213–222.
[7] H. Shacham, B. Waters, Compact proofs of retrievability, in: Proc. of Asiacrypt 2008, pp. 90–107.
[8] D. Boneh, B. Lynn, H. Shacham, Short signatures from the Weil pairing, J. Cryptology 17 (4) (2004) 297–319.
[9] C. Wang, Q. Wang, K. Ren, N. Cao, W. Lou, Toward secure and dependable storage services in cloud computing, IEEE Trans. Serv. Comput. 5 (2) (2012) 220–232.
[10] Q. Wang, C. Wang, K. Ren, W. Lou, J. Li, Enabling public auditability and data dynamics for storage security in cloud computing, IEEE Trans. Parallel Distrib. Syst. 22 (5) (2011) 847–859.
[11] K. Yang, X. Jia, Data storage auditing service in cloud computing: challenges, methods and opportunities, World Wide Web 15 (4) (2012) 409–428.
[12] C. Wang, K. Ren, W. Lou, J. Li, Toward publicly auditable secure cloud data storage services, IEEE Netw. 24 (4) (2010) 19–24.
[13] Y. Zhu, H. Hu, G. Ahn, M. Yu, Cooperative provable data possession for integrity verification in multi-cloud storage, IEEE Trans. Parallel Distrib. Syst. 23 (12) (2012) 2231–2244.
[14] Y. Zhu, H. Wang, Z. Hu, G.-J. Ahn, H. Hu, S.S. Yau, Dynamic audit services for integrity verification of outsourced storages in clouds, in: SAC 2011, pp. 1550–1557.
[15] C. Wang, Q. Wang, K. Ren, W. Lou, Privacy-preserving public auditing for data storage security in cloud computing, in: Proc. of INFOCOM 2010, pp. 525–533.
[16] B. Wang, B. Li, H. Li, Oruta: privacy-preserving public auditing for shared data in the cloud, in: IEEE International Conference on Cloud Computing, 2012, pp. 293–302.
[17] B. Wang, B. Li, H. Li, Knox: privacy-preserving auditing for shared data with large groups in the cloud, in: Proc. of ACNS 2012, pp. 507–525.
[18] D. Boneh, C. Gentry, B. Lynn, H. Shacham, Aggregate and verifiably encrypted signatures from bilinear maps, in: Proc. of Eurocrypto 2003, pp. 416–432.
[19] D. Boneh, B. Lynn, H. Shacham, Short group signatures, in: Proc. of Crypto 2004, pp. 41–55.
[20] D. Boneh, M.K. Franklin, Identity-based encryption from the weil pairing, in: Proc. of Crypto 2001, pp. 213–229.

Yong Yu received his Ph.D. degree in cryptography from Xidian University in 2008. He is currently an associate professor of University of Electronic Science and Technology of China and a research fellow of University of Wollongong as well. His research focuses on cryptography and its applications, especially public encryption, digital signature and secure cloud storage.

Lei Niu is a master student of the University of Electronic Science and Technology of China, and his research interests are digital signature and secure cloud storage.

Guomin Yang received his Ph.D. degree from the Computer Science Department at City University of Hong Kong in 2009. He is currently a lecturer of the University of Wollongong. His research interests are cryptography and cloud computing security.

Yi Mu received his Ph.D. from the Australian National University in 1994. He currently is a professor, Head of School of Computer Science and Software Engineering and the codirector of Centre for Computer and Information Security Research at the University of Wollongong, Australia. His current research interests include network security, computer security, and cryptography. Yi Mu is the editor-in-chief of International Journal of Applied Cryptography and serves as associate editor for nine other international journals. He is a senior member of the IEEE and a member of the IACR.

Willy Susilo received a Ph.D. in Computer Science from the University of Wollongong, Australia. He is a Professor at the School of Computer Science and Software Engineering and the codirector of the Centre for Computer and Information Security Research (CCISR) at the University of Wollongong. He is currently holding the prestigious ARC Future Fellow awarded by the Australian Research Council (ARC). His main research interests include cryptography and information security. His main contribution is in the area of digital signature schemes. He has served as a program committee member in dozens of international conferences. He has published numerous publications in the area of digital signature schemes and encryption schemes.