Knox: Privacy-Preserving Auditing for Shared Data with Large Groups in the Cloud

Boyang Wang[1,2], Baochun Li[2] and Hui Li[1]

[1] State Key Laboratory of Integrated Services Networks, Xidian University, China
{bywang, lihui}@mail.xidian.edu.cn
[2] Department of Electrical and Computer Engineering, University of Toronto, Canada
bli@eecg.toronto.edu

Abstract. With cloud computing and storage services, data is not only stored in the cloud, but routinely shared among a large number of users in a group. It remains elusive, however, to design an efficient mechanism to audit the integrity of such shared data, while still preserving identity privacy. In this paper, we propose Knox, a privacy-preserving auditing mechanism for data stored in the cloud and shared among a large number of users in a group. In particular, we utilize group signatures to construct homomorphic authenticators, so that a third party auditor (TPA) is able to verify the integrity of shared data for users without retrieving the entire data. Meanwhile, the identity of the signer on each block in shared data is kept private from the TPA. With Knox, the amount of information used for verification, as well as the time it takes to audit with it, are not affected by the number of users in the group. In addition, Knox exploits homomorphic MACs to reduce the space used to store such verification information. Our experimental results show that Knox is able to efficiently audit the correctness of data, shared among a large number of users.

Keywords: Privacy-Preserving, Auditing, Shared Data, Cloud Computing

1 Introduction

With cloud computing and storage, users are able to access and to share resources offered by cloud service providers at a lower marginal cost. With Dropbox, for example, data is stored in the cloud (operated by Amazon), and shared among a group of users in a collaborative manner. It is natural for users to wonder whether their data remain intact over a prolonged period of time: due to hardware failures and human errors in an untrusted cloud environment [2], the integrity of data stored in the cloud can become compromised. To protect the integrity of data in the cloud and to offer "peace of mind" to users, it is best to introduce a third party auditor (TPA) to perform auditing tasks on behalf of users. Such a third party auditor enjoys amply computation/communication resources that users may not possess.

Provable data possession (PDP) [3], first proposed by Ateniese *et al.*, allows a verifier to perform public auditing on the integrity of data stored in an untrusted server without retrieving the entire data. Subsequent work focused on how dynamic data [5, 11, 20, 24] and data privacy [19] can be supported during the public auditing process. However, most of previous work only focus on auditing the integrity of *personal data*. Recently, Wang *et al.* [16] first design a privacy-preserving public auditing mechanism (named Oruta) for *shared data* in an untrusted cloud, so that the identity of the signer on each block in shared data is not disclosed to the third party auditor (TPA) during an auditing task. Without knowing the identities of signers, the TPA cannot learn which user in the group or which block in shared data is a higher valuable target than others [16].

Unfortunately, Oruta [16] fails to scale well to a large number of users sharing data in a group. In Oruta, information used for verification are computed with ring signatures [8]; as a result, the size of verification information, as well as the time it takes to audit with it, are linearly increasing with the number of users in a group. To make matters worse, when adding new users to a group, all the existing verification information will need to be re-computed if ring signatures are used, introducing a significant computation burden to all users. In addition, the identities of signers are unconditional [8] protected by ring signatures, which prevent the group manager to trace the identity when someone in the group is misbehaved.

In this paper, we propose Knox, a new privacy-preserving mechanism to audit data stored in an untrusted cloud and shared among a large number of users in a group. In Knox, we take advantage of group signatures [6, 12] to construct homomorphic authenticators [3, 15], so that the third party auditor is able to verify the integrity of shared data without retrieving the entire data, but cannot reveal the identities of signers on all blocks in shared data. Meanwhile, the size of verification information, as well as the time it takes to audit with it, are not affected when the number of users sharing the data increases. The original user, who creates and shares the data in the cloud, is able to add new users into a group without re-computing any verification information. In addition, the original user (acts as the group manager) can trace group signatures on shared data, and reveal the identities of signers when it is necessary. We also utilize homomorphic MACs [1] to effectively reduce the amount of storage space needed to store verification information. As a necessary trade-off, we allow the third party auditor to share a secret key pair with users, which we refer to as *authorized auditing*. Although we allow an authorized TPA to possess the secret key pair, the TPA cannot compute valid group signatures as group users because this secret key pair is only a part of a group user's private key. To our best knowledge, we present the first mechanism designed with scalability in mind when it comes to support auditing data shared among a large number of users in a privacy-preserving fashion. A high-level comparison between Knox and previous work [16] is shown in Table 1.

The remainder of this paper is organized as follows. In Section 2, we briefly discuss related work. Then, we present the system model, threat model and

Table 1. Comparison between Previous Work [16] and Knox

	Previous work [16]	Knox
Public Auditing	Yes	No
Identity Privacy	Yes	Yes
Support for Large Groups	No	Yes
Traceability	No	Yes

design goals in Section 3. In Section 4, we introduce complexity assumptions and cryptographic primitives used in Knox. Detailed design and security analysis of Knox are presented in Section 5 and 6. Finally, we evaluate the performance of Knox in Section 7, and conclude this paper in Section 8.

2 Related Work

Ateniese *et al.* [3] first proposed provable data possession (PDP), which allows a client to verify the integrity of her data stored at an untrusted server without retrieving the entire file. However, this mechanism is only suitable for static data. To improve the efficiency of verification, Ateniese *et al.* [5] constructed scalable and efficient PDP using symmetric keys. Unfortunately, it cannot support public verifiability, and only offers each user a limited number of verification requests.

Juels and Kaliski [14] defined another similar model called proofs of retrievability (POR), which is also able to check the correctness of data on an untrusted server. The original file is added with a set of randomly-valued check blocks called sentinels. The verifier challenges the untrusted server by specifying the positions of a collection of sentinels, and by asking the untrusted server to return the associated sentinel values. Shacham and Waters [15] designed two improved POR mechanisms, which are built on BLS signatures and pseudo-random functions.

Wang *et al.* [20] leveraged the Merkle Hash Tree to construct a public auditing mechanism with fully dynamic data. Hao *et al.* [13] also designed a dynamic public auditing mechanism based on RSA. Erway *et al.* [11] presented a dynamic PDP based on the rank-based authenticated dictionary. Zhu *et al.* [24] exploited index hash tables to support fully dynamic data. To ensure the correctness of users' data stored on multiple servers, Wang *et al.* [18] utilized homomorphic tokens and erasure codes in the auditing process. An excellent survey of previous work about data auditing can be found in [21].

Wang *et al.* [19] considered data privacy with public auditing in the cloud. In their mechanism, the TPA is able to check the integrity of cloud data but cannot obtain any private data. Zhu *et al.* [23] also designed a mechanism to preserve data privacy from the TPA. Recent work [16], Oruta, represents the first privacy-preserving public auditing mechanism for shared data in the cloud. In this mechanism, the TPA can verify the integrity of shared data, but is not able to reveal the identity of the signer on each block. Unfortunately, it is not readily scalable to auditing the integrity of data shared among a large number of users in the group.

3 Problem Statement

3.1 System Model

In this paper, we consider data storage and sharing services in the cloud with three entities: the cloud, the third party auditor (TPA), and users who participate as a group (as shown in Fig. 1). Users in a group include one original user and a number of group users. The original user is the original owner of data, and shares data in the cloud with other users. Based on access control policies [22], other users in the group are able to access, download and modify shared data. The cloud provides data storage and sharing services for users, and has ample storage space. The third party auditor is able to verify the integrity of shared data based on requests from users, without downloading the entire data.

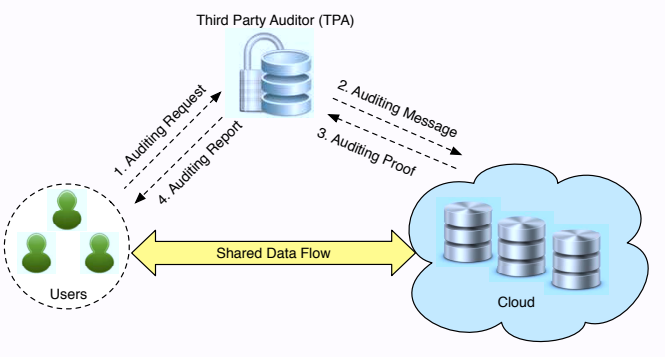

Fig. 1. The system model includes the cloud, the third party auditor and users.

When a user (either the original user or a group user) wishes to check the integrity of shared data, she first sends an auditing request to the TPA. After receiving the auditing request, the TPA generates an auditing message to the cloud, and retrieves an auditing proof of shared data from the cloud. Then the TPA verifies the correctness of the auditing proof. Finally, the TPA sends an auditing report to the user based on the result of the verification.

3.2 Threat Model

Integrity Threats In general, two kinds of threats related to the integrity of shared data are possible. First, an external adversary may try to pollute shared data in the cloud, and prevent users from using shared data correctly. Second, the cloud service provider may inadvertently corrupt or even remove shared data in the cloud due to hardware failures and human errors. To make matters worse, in order to avoid jeopardizing its reputation, the cloud service provider may be reluctant to inform users about such corruption of data.

Privacy Threats During an auditing task, a semi-trusted TPA, who is only responsible for verifying the integrity of shared data, may try to reveal the identity of the signer on each block in shared data based on verification information (i.e. signatures). The identity of the signer on each block is private and sensitive information, which users do not wish to be revealed to any third party.

3.3 Design Goals

To make it efficient and secure for the TPA to verify shared data with a large number of users in a group, Knox should be designed to achieve the following properties: (1) **Correctness**: The TPA is able to correctly audit the integrity of shared data. (2) **Efficiency**: The TPA is able to verify the integrity of shared data without retrieving the entire data from the cloud. (3) **Identity privacy**: During an auditing task, the TPA cannot distinguish the identity of the signer on each block. (4) **Support for large groups**: The TPA is able to efficiently audit data that are shared among a large number of users. In particular, the size of verification information, as well as the time it takes to audit with it, are not affected by the number of users in the group; the original user can add new users to the group without re-computing existing verification information. (5) **Traceability**: The original user is able to trace a signature on a block and reveal the identity of the signer.

4 Preliminaries

4.1 Bilinear Maps

Let G_1, G_2 and G_T be three multiplicative cyclic groups of prime order p, g_1 be a generator of G_1, and g_2 be a generator of G_2. A bilinear map e is a map e: $G_1 \times G_2 \rightarrow G_T$ with the following properties: 1) **Computability**: there exists an efficiently computable algorithm for computing map e. 2) **Bilinearity**: for all $u \in G_1$, $v \in G_2$ and $a, b \in Z_p$, $e(u^a, v^b) = e(u, v)^{ab}$. 3) **Non-degeneracy**: $e(g_1, g_2) \neq 1$. For ease of exposition, we assume $G_1 = G_2$ in the rest of this paper.

4.2 Complexity Assumptions

Definition 1. ***Computational Diffie-Hellman (CDH) Problem.*** *For* $(a, b) \in Z_p^2$*, given* $(g_1, g_1^a, g_1^b) \in G_1^3$ *as input, output* $g_1^{ab} \in G_1$*.*

The CDH assumption holds in G_1 if no t-time algorithm has advantage at least ϵ in solving the CDH problem in G_1, which means it is *computational infeasible* to solve the CDH problem in G_1.

Definition 2. ***q-Strong Diffie-Hellman (q-SDH) Problem.*** *For* $\gamma \in Z_p$*, given a* $(q+2)$*-tuple* $(g_1, g_2, g_2^{\gamma}, g_2^{\gamma^2}, ..., g_2^{\gamma^q}) \in G_1 \times G_2^{q+1}$ *as input, output a pair* $(g_1^{1/(\gamma+x)}, x) \in G_1 \times Z_p$*.*

The q-SDH assumption holds in (G_1, G_2) if no t-time algorithm has advantage at least ϵ in solving the q-SDH problem in (G_1, G_2).

Definition 3. ***Decision Linear (DL) Problem.*** *For $(a, b, c) \in Z_p^3$, given $(u, v, h, u^a, v^b, h^c) \in G_1^6$ as input, output yes if $a + b = c$ and no otherwise.*

The DL assumption holds in G_1 if no t-time algorithm has advantage at least ϵ in solving the DL problem in G_1.

4.3 Group Signatures

Group signatures, first introduced by Chaum and van Heyst [10], aim to provide anonymity of signers, who are from a same group. A verifier is convinced that messages are correct and signed by one of the group members, but cannot reveal the identity of the signer. Meanwhile, only the group manager is able to trace these group signatures and reveal the identity of the signer. Boneh *et al.* [6] (denoted as BBS) proposed a short group signature scheme based on the q-SDH assumption. In their scheme, the length of each group signature is independent from the number of group members.

4.4 Homomorphic MACs

Homomorphic MACs, introduced by Agrawal and Boneh [1], provide data integrity for network coding. Using homomorphic MACs, an intermediate node can construct a valid MAC of an output block based on the MACs of input blocks without knowing the secret keys. More specifically, given a block $\boldsymbol{m}_j = (m_{j,1}, ..., m_{j,n}) \in Z_p^n$, the homomorphic MAC of this block can be computed as $t_j = \sum_{i=1}^{n} \delta_i m_{j,i} + b_j \in Z_p$, where $\boldsymbol{\delta} = (\delta_1, ..., \delta_n)$ is generated by a pseudo-random generator and a secret key k_{prg}, and b_j is calculated by a pseudo-random function and a secret key k_{prf}. Given t_1 and t_2, an intermediate node can compute a valid MAC of a new block $\boldsymbol{m}' = \boldsymbol{m}_1 + \boldsymbol{m}_2$ by calculating $t' = t_1 + t_2$ without knowing the secret key pair (k_{prg}, k_{prf}).

4.5 Homomorphic Authenticators

Homomorphic authenticators (also denoted as homomorphic verifiable tags) are basic tools to construct data auditing mechanisms [3,13,15,16,19,23,24]. Besides unforgeability (only a user with a private key can generate valid signatures), a homomorphic authenticable signature scheme, which denotes a homomorphic authenticator based on signatures, should also satisfy the following properties:

Let $(\mathbf{pk}, \mathbf{sk})$ denote the signer's public/private key pair, σ_1 denote the signature on message $m_1 \in Z_p$, σ_2 denote the signature on message $m_2 \in Z_p$.

- **Blockless verification:** Given σ_1 and σ_2, two random values α_1, $\alpha_2 \in Z_p$ and a message $m' = \alpha_1 m_1 + \alpha_2 m_2 \in Z_p$, a verifier is able to check the correctness of message m' without knowing message m_1 and m_2.

– **Non-malleability:** Given σ_1 and σ_2, two random values α_1, $\alpha_2 \in Z_p$ and a message $m' = \alpha_1 m_1 + \alpha_2 m_2 \in Z_p$, a user who does not have private key **sk**, is not able to generate a valid signature σ' for message m' by combining signature α_1 and α_2.

The first property allows a verifier to check the correctness of data in the cloud with a linear combination of all the blocks, while the entire data does not need to be downloaded to the verifier. The second property prevents an attacker from generating signatures for invalid messages by combining existing signatures. Other cryptographic techniques related to homomorphic authenticable signatures includes aggregate signatures [8], homomorphic signatures [7] and batch-verification signatures [12]. If a signature scheme is blockless verifiable and malleable, it is a homomorphic signature scheme. In the construction of data auditing mechanisms, we should use homomorphic authenticable signatures, not homomorphic signatures. Otherwise, based on malleability of homomorphic signatures, an adversary can successfully corrupt data in the cloud by linearly combining existing blocks and corresponding signatures.

5 Homomorphic Authenticable Group Signatures

5.1 Overview

As introduced at the beginning of this paper, we expect to utilize group signatures for computing verification information, so that the identity of the signer on each block can be kept private from the TPA. However, traditional group signatures [4, 6, 10] cannot be directly used in Knox, since they are not blockless verifiable. Without blockless verification, a verifier has to download the entire data to check the integrity of shared data, which consumes excessive bandwidth and takes long verification times. Therefore, we first build a homomorphic authenticable group signature (HAGS) scheme in this section. Then we will present the full construction of our privacy-preserving auditing mechanism for shared data among a large number of users based on HAGS in the next section.

In HAGS, we extend BBS group signatures [6] to achieve blockless verification. Meanwhile, to keep unforgeability (nobody outside the group can produce valid signatures) of HAGS, we leverage BLS signatures [9] as a part of our group signatures. BLS signatures, which are based on bilinear maps, are used in previous work [15,19] to audit data integrity of personal users. In addition, we exploit batch verification methods of group signatures in [12] to improve the efficiency of HAGS for verifying multiple group signatures. Note that if only using BLS signatures among a group of users, which means that all the users in the group generate signatures on messages only with a common private key, it is also possible to achieve identity privacy on messages. Unfortunately, traceability of the group manager on signatures generated by group members will be immediately lost.

5.2 Construction of HAGS

Following general constructions of group signatures in [4,6], HAGS contains five algorithms: **KeyGen**, **Join**, **Sign**, **Verify** and **Open**. In **KeyGen**, the group manager generates her private key and a group public key. In **Join**, the group manager is able to compute a private key for a new group user and add this user to the group user list. A group user signs messages using her private key and the group public key in **Sign**. In **Verify**, a verifier is able to check the correctness of a message using the group public key, but she cannot reveal the identity of the signer. The group manager can reveal the identity of the signer on a message in **Open**.

Scheme Details: Let G_1, G_2 and G_T be multiplicative cyclic groups of order p, g_1 and g_2 be generators of G_1 and G_2 respectively, $G_1 \times G_2 \rightarrow G_T$ be a bilinear map. There are two hash functions, $H_1 : \{0,1\}^* \rightarrow Z_p$ and $H_2 : \{0,1\}^* \rightarrow G_1$. The total number of group users is d.

KeyGen. The group manager randomly selects $h \in G_1 \backslash \{1_{G_1}\}$ and ξ_1, $\xi_2 \in Z_p^*$, and sets u, $v \in G_1$ such that $u^{\xi_1} = v^{\xi_2} = h$. Then, she randomly selects $\gamma, \pi, \eta \in Z_p^*$, and sets $w = g_2^{\gamma}, \rho = g_2^{\pi} \in G_2$.

The group public key is $\mathbf{gpk} = (g_1, g_2, h, u, v, w, \rho, \eta)$, the group manager's private key is $\mathbf{gmsk} = (\xi_1, \xi_2)$. The group manager keeps γ private. And π will be a part of a group user's private key, which is issued to group users later.

Join. For user i, $1 \leq i \leq d$, the group manager randomly selects $x_i \in Z_p^*$ with $x_i + \gamma \neq 0$, and sets $A_i = g_1^{1/(\gamma + x_i)} \in G_1$. The private key of user i is $\mathbf{gsk[i]} = (A_i, x_i, \pi)$. The group manager secretly issues $\mathbf{gsk[i]}$ to user i, and adds this user into the group user list.

Sign. Given a group public key $\mathbf{gpk} = (g_1, g_2, h, u, v, w, \rho, \eta)$, a private key $\mathbf{gsk[i]} = (A_i, x_i, \pi)$, a message $m \in Z_p$ and this message's identifier id, user i computes the signature σ as follows:

1. Randomly select $\alpha, \beta, r_\alpha, r_\beta, r_x, r_{\gamma_1}, r_{\gamma_2} \leftarrow Z_p$.
2. Compute T_1, T_2 and T_3 as $T_1 = u^{\alpha}, T_2 = v^{\beta}, T_3 = A_i \cdot h^{\alpha+\beta}$.
3. Compute $\gamma_1 = x_i \cdot \alpha$ and $\gamma_2 = x_i \cdot \beta$.
4. Compute R_1, R_2, R_3, R_4 and R_5 as

$$R_1 = u^{r_\alpha}, \quad R_2 = v^{r_\beta}, \quad R_4 = T_1^{r_x} \cdot u^{-r_{\gamma_1}}, \quad R_5 = T_2^{r_x} \cdot v^{-r_{\gamma_2}},$$

$$R_3 = e(T_3, g_2)^{r_x} \cdot e(h, w)^{-r_\alpha - r_\beta} \cdot e(h, g_2)^{-r_{\gamma_1} - r_{\gamma_2}}.$$

5. Compute a challenge $c \in Z_p$ as $c = \eta^m H_1(T_1, T_2, T_3, R_1, R_2, R_3, R_4, R_5)$. For ease of exposition, we use $H_1(T_1, ..., R_5)$ instead of $H_1(T_1, T_2, T_3, R_1, R_2, R_3, R_4, R_5)$ in the remainder of this paper.
6. Compute $s_\alpha, s_\beta, s_x, s_{\gamma_1}, s_{\gamma_2} \in Z_p$ as:

$$s_\alpha = r_\alpha + c \cdot \alpha, \quad s_\beta = r_\beta + c \cdot \beta, \quad s_x = r_x + c \cdot x_i,$$

$$s_{\gamma_1} = r_{\gamma_1} + c \cdot \gamma_1, \quad s_{\gamma_2} = r_{\gamma_2} + c \cdot \gamma_2.$$

7. Compute a tag θ as $\theta = [H_2(id) g_1^m]^{\pi} \in G_1$.

8. Output the signature of this message m as $\sigma = (T_1, T_2, T_3, \theta, R_3, c, s_\alpha, s_\beta, s_x, s_{\gamma_1}, s_{\gamma_2}) \in G_1^4 \times G_T \times Z_p^6$.

Verify. Given a group public key $\mathbf{gpk} = (g_1, g_2, h, u, v, w, \rho, \eta)$, a message m, an identifier id and a group signature $\sigma = (T_1, T_2, T_3, \theta, R_3, c, s_\alpha, s_\beta, s_x, s_{\gamma_1}, s_{\gamma_2})$, a verifier checks the integrity of this message as follows:

1. Re-compute values R_1, R_2, R_4 and R_5 from σ as:

$$\tilde{R_1} = u^{s_\alpha} \cdot T_1^{-c}, \quad \tilde{R_2} = v^{s_\beta} \cdot T_2^{-c}, \quad \tilde{R_4} = T_1^{s_x} \cdot u^{-s_{\gamma_1}}, \quad \tilde{R_5} = T_2^{s_x} \cdot v^{-s_{\gamma_2}}$$

2. Check the following equations as:

$$R_3 \stackrel{?}{=} e(T_3, g_2)^{s_x} e(h, w)^{-s_\alpha - s_\beta} e(h, g_2)^{-s_{\gamma_1} - s_{\gamma_2}} \cdot \big(e(T_3, w) \cdot e(g_1, g_2)^{-1}\big)^c, \quad (1)$$

$$c \stackrel{?}{=} \eta^m \cdot H_1(T_1, T_2, T_3, \tilde{R_1}, \tilde{R_2}, R_3, \tilde{R_4}, \tilde{R_5}), \quad (2)$$

$$e(\theta, g_2) \stackrel{?}{=} e(H_2(id) \cdot g_1^m, \rho). \quad (3)$$

If all the three equations hold, the verifier accepts message m. Otherwise, she rejects this message.

Open. Only the group manager can trace a group signature and reveal the identity of the signer. Given a group public key $\mathbf{gpk} = (g_1, g_2, h, u, v, w, \rho, \eta)$, the group manager's private key $\mathbf{gmsk} = (\xi_1, \xi_2)$, a message m and a signature σ, the group manager reveals the identity of the signer as follows:

1. Verify that the signature σ is a valid signature on message m.
2. Decrypt user i's A_i as $A_i = T_3/(T_1^{\xi_1} \cdot T_2^{\xi_2})$.
3. Given A_i, which is a part of user i's private key, the group manager is able to reveal the identity of the signer on message m.

5.3 Security Analysis of HAGS

Theorem 1. *Given a message m and its group signature σ, a verifier is able to correctly check the integrity of message m under HAGS.*

Proof. Equation (1) is correct because $e(T_3, g_2)^{r_x} \cdot e(h, w)^{-r_\alpha - r_\beta} \cdot e(h, g_2)^{-r_{\gamma_1} - r_{\gamma_2}}$ $= e(T_3, g_2)^{s_x} \cdot e(h, w)^{-s_\alpha - s_\beta} \cdot e(h, g_2)^{-s_{\gamma_1} - s_{\gamma_2}} \cdot \big(e(T_3, w) \cdot e(g_1, g_2)^{-1}\big)^c$. Because R_1, R_2, R_4, R_5 can be successfully recomputed [6], Equation (2) is correct. Equation (3) is correct since $e(\theta, g_2) = e([H_2(id) g_1^m]^\pi, g_2) = e(H_2(id) g_1^m, \rho)$. Further explanations about correctness can be found in [6, 12].

Theorem 2. *Suppose $\mathcal{F}$ is a (t', ϵ')-algorithm that can generate a forgery of a group signature under HAGS. Then there exists a (t, ϵ)-algorithm $\mathcal{A}$ that can solve the CDH problem with $t \leq t' + (q_H + q_S + 1)c_{G_1}$ and $\epsilon \geq \epsilon'/(\mathbf{e} + q_S \mathbf{e})$, where $\mathcal{F}$ issues at most q_H hash queries and at most q_S signing queries, $\mathbf{e} = \lim_{q_S \to \infty}(1 + 1/q_S)^{q_S}$, exponentiation and inversion on G_1 take time c_{G_1}.*

Proof. Details of this proof can be found in our technical report [17].

Theorem 3. *HAGS is a homomorphic authenticable group signature scheme.*

Proof. We first prove that HAGS has the property of blockless verification. Then we show HAGS is also non-malleable.

Given a group public key $\mathbf{gpk} = (g_1, g_2, h, u, v, w, \rho, \eta)$, two signatures $\sigma_1 = (T_{1,1}, T_{1,2}, T_{1,3}, \theta_1, R_{1,3}, c_1, s_{1,\alpha}, s_{1,\beta}, s_{1,x}, s_{1,\gamma_1}, s_{1,\gamma_2})$, $\sigma_2 = (T_{2,1}, T_{2,2}, T_{2,3}, \theta_2, R_{2,3}, c_2, s_{2,\alpha}, s_{2,\beta}, s_{2,x}, s_{2,\gamma_1}, s_{2,\gamma_2})$, and a message $m' = \sum_{j=1}^{2} y_j m_j \in Z_p$, where $y_j \in Z_p^*$, a verifier is able to check the correctness of message m' without knowing message m_1 and m_2. More specifically, she first recomputes $R_{j,1}$, $R_{j,2}$, $R_{j,4}$ and $R_{j,5}$ as in **Verify**. Then she checks:

$$\prod_{j=1}^{2} R_{j,3}^{y_j} \stackrel{?}{=} e(\prod_{j=1}^{2}(T_{j,3}^{s_{j,x}} \cdot h^{-s_{j,\gamma_1}-s_{j,\gamma_2}} \cdot g_1^{-c_j})^{y_j}, g_2) e(\prod_{j=1}^{2}(h^{-s_{j,\alpha}-s_{j,\beta}} \cdot T_{j,3}^{c_j})^{y_j}, w), \tag{4}$$

$$\prod_{j=1}^{2} c_j^{y_j} \stackrel{?}{=} \eta^{m'} \cdot \prod_{j=1}^{2} H_1(T_{j,1}, ..., \tilde{R}_{j,5})^{y_j}, \tag{5}$$

$$e(\prod_{j=1}^{2} \theta_j^{y_j}, g_2) \stackrel{?}{=} e(\prod_{j=1}^{2} H_2(id_j)^{y_j} \cdot g_1^{m'}, \rho). \tag{6}$$

Only if all the three equations hold, then the verifier accepts message m'.

Note that only Equation (5) and (6) are related to message m', while Equation (4) is independent from the content of message m'. The correctness of Equation (4) can be proved using batch verification methods of group signatures [12]. Based on Theorem 1, the correctness of Equation (5) and (6) can be proved as:

$$\begin{aligned}\prod_{j=1}^{2} c_j^{y_j} &= \prod_{j=1}^{2} \left(\eta^{m_j} \cdot H_1(T_{j,1}, ..., \tilde{R}_{j,5})\right)^{y_j} \\ &= \eta^{y_1 m_1 + y_2 m_2} \cdot \prod_{j=1}^{2} H_1(T_{j,1}, ..., \tilde{R}_{j,5})^{y_j} = \eta^{m'} \cdot \prod_{j=1}^{2} H_1(T_{j,1}, ..., \tilde{R}_{j,5})^{y_j}.\end{aligned}$$

$$\begin{aligned}e(\prod_{j=1}^{2} \theta_j^{y_j}, g_2) &= e(\prod_{j=1}^{2}(H_2(id_j) \cdot g_1^{m_j})^{\pi \cdot y_j}, g_2) \\ &= e(\prod_{j=1}^{2} H_2(id_j)^{y_j} \cdot g_1^{y_1 m_1 + y_2 m_2}, g_2^{\pi}) = e(\prod_{j=1}^{2} H_2(id_j)^{y_j} \cdot g_1^{m'}, \rho).\end{aligned}$$

Because all the three equations are correct, HAGS is blockless verifiable.

Meanwhile, an attacker, who does not have a private key, cannot generate a valid signature σ' for message m' by combining existing signatures. More specifically, this user cannot construct a tag θ' by linearly combining θ_1 and θ_2 with

y_1 and y_2. Because $\theta_1^{y_1}\theta_2^{y_2} = [H_2(id_1)^{y_j}H_2(id_2)^{y_2}g_1^{m'}]^{\pi}$, $\theta' = [H_2(id')g_1^{m'}]^{\pi}$ and $H_2(id') \neq H_2(id_1)^{y_j}H_2(id_2)^{y_2}$, then we have $\theta_1^{y_1} \cdot \theta_2^{y_2} \neq \theta'$. Therefore, HAGS is non-malleable.

Theorem 4. *Given a message m and its group signature σ, only the group manager can reveal the identity of the signer on this message. For a verifier, it is computational infeasible to reveal the identity of the signer on message m.*

Proof. For the group manager, she can always successfully recover the identity of the signer on message m using her manager private key $\mathbf{gmsk} = (\xi_1, \xi_2)$. Because $T_3/(T_1^{\xi_1} \cdot T_2^{\xi_2}) = A_i \cdot h^{\alpha+\beta}/(u^{\alpha \cdot \xi_1} \cdot v^{\beta \cdot \xi_2}) = A_i \cdot h^{\alpha+\beta}/h^{\alpha+\beta} = A_i$. For a verifier, if she can successfully choose a value c with $c = \alpha+\beta$, then she is able to decrypt A_i and reveal the identity of the signer by computing $T_3/h^c = A_i \cdot h^{a+b}/h^c$. However, given $u, v, h, T_1 = u^{\alpha}, T_2 = v^{\beta}, h^c \in G_1$, deciding whether $c = \alpha + \beta$ is as hard as solving Decision Linear problem in G_1. Further proofs about anonymity and traceability of group signatures can be found in [6].

6 Privacy-Preserving Auditing for Shared Data

6.1 Overview

We now present Knox, a privacy-preserving auditing mechanism for shared data among a large number of users. Using HAGS, we can preserve the identity of the signer on each block from the TPA. Meanwhile, the original user, who is the group manager and shares data with other group users, can reveal an identity of a signer when it is necessary. Moreover, the length of each group signature is independent from the number of group users, which is a desirable property for large groups to share their data in the cloud. If users wish to protect the privacy of shared data during an auditing task, users can encrypt data using encryption techniques, such as the combination of symmetric encryption and attribute-based encryption [22], before outsourcing data to the cloud server. The main objective of designing Knox is to provide identity privacy for users.

To reduce the storage space of group signatures on shared data, we utilize homomorphic MACs [1] to compress each block into a small value, and then sign this small value instead of signing the entire block. As a necessary trade-off, Knox does not support public auditing, since the TPA in our mechanism needs to share a secret key pair with all group users, referred to as authorized auditing. This secret key pair is used to compute homomorphic MACs. Although we allow an authorized TPA to possess this secret key pair, the TPA cannot compute valid group signatures as group users because this secret key pair is only a part of a group user's private key.

Because the computation of a signature includes an identifier of a block (as we described in HAGS), conventional methods, which only use the index of a block as its identifier, are not suitable for dynamic data. The reason is that when a user modifies shared data by performing an insert or delete operation on a single block, the indices of blocks that after the modified block are all changed,

and the change of these indices requires users to re-compute the signatures of these blocks, even though the content of these blocks are not modified [16]. To avoid this type of re-computation and support dynamic data for users, we take advantage of index hash tables [16, 24] as identifiers of blocks. Further explanations about index hash tables can be found in [16, 24].

In addition, we continue to use sampling strategies as previous work [3] to detect any corrupted block in shared data with a high probability, by only choosing a subset of all blocks in each auditing task. For example, if 1% of all the blocks are corrupted, the TPA can detect this misbehavior with a probability greater than 99% by choosing only 460 random selected blocks, where the number of selected blocks is independently with the total number of blocks in shared data if the percentage of corrupted blocks is fixed [3]. To improve the detection probability, the TPA can increase the number of selected blocks in each auditing task [3, 23]. In some emerging applications, the auditor may need to achieve a 100% detection probability if only one corrupted block exists, then all the blocks in shared data should be selected during an auditing task. As a trade-off, the computation and communication cost are significantly increased.

6.2 Construction of Knox

Knox includes six algorithms: **KeyGen**, **Join**, **Sign**, **ProofGen**, **ProofVerify** and **Open**. In **KeyGen**, the original user of shared data generates a group public key and a group manager private key. In **Join**, the original user, who acts as the group manager, is able to issue private keys to users. A user (either the original user or a group user) is able to sign blocks using her private key and the group public key in **Sign**. In **ProofGen**, the cloud generates a proof of possession of shared data to the TPA. **ProofVerify** is operated by the TPA to verify the correctness of the proof. The original user can reveal the identity of the signer on each block in **Open**.

Scheme Details: Let G_1, G_2 and G_T be multiplicative cyclic groups of order p, g_1 and g_2 be generators of G_1 and G_2 respectively, $G_1 \times G_2 \rightarrow G_T$ be a bilinear map. There are two hash functions, $H_1 : \{0,1\}^* \rightarrow Z_p$ and $H_2 : \{0,1\}^* \rightarrow G_1$. The total number of group users is d. Data M, which is going to be shared by users, is divided into n blocks. Each block is further divided into k elements of Z_p. Therefore, shared data M can be presented as:

$$M = \begin{pmatrix} \boldsymbol{m}_1 \\ \vdots \\ \boldsymbol{m}_n \end{pmatrix} = \begin{pmatrix} m_{1,1} & \dots & m_{1,k} \\ \vdots & \ddots & \vdots \\ m_{n,1} & \dots & m_{n,k} \end{pmatrix} \in Z_p^{n \times k}.$$

There are also a pseudo-random generator $\mathtt{PRG}$: $\mathcal{K}_{prg} \rightarrow Z_p^k$ and a pseudo-random function $\mathtt{PRF}$: $\mathcal{K}_{prf} \times \mathcal{I} \rightarrow Z_p$, where $\mathcal{K}_{prg}$ and $\mathcal{K}_{prf}$ denote the set of secret keys for $\mathtt{PRG}$ and $\mathtt{PRF}$ respectively, and $\mathcal{I}$ is the set of all identifiers in the index hash table of data M.

KeyGen. The original user, who acts as the group manager, first selects system parameters as in HAGS. Meanwhile, she also randomly computes a secret

key pair $\mathbf{skp} = (sk_{prg}, sk_{prf})$, where $sk_{prg} \in \mathcal{K}_{prg}$ and $sk_{prf} \in \mathcal{K}_{prf}$. The group public key is $\mathbf{gpk} = (g_1, g_2, h, u, v, w, \rho, \eta)$, the group manager's private key is $\mathbf{gmsk} = (\xi_1, \xi_2)$. The original user keeps γ private. Both π and $\mathbf{skp}$ will be a part of a group user's private key, which is issued to group users later. The original user also privately shares $\mathbf{skp} = (sk_{prg}, sk_{prf})$ with an authorized TPA.

Join. For user i, $1 \leq i \leq d$, the original user randomly selects $x_i \in Z_p^*$ with $x_i + \gamma \neq 0$, and sets $A_i = g_1^{1/(\gamma + x_i)} \in G_1$. The private key of user i is $\mathbf{gsk[i]} = (A_i, x_i, \pi, \mathbf{skp})$. The original user secretly issues $\mathbf{gsk[i]}$ to user i, and adds this user into the group user list.

Sign. Given a group public key $\mathbf{gpk} = (g_1, g_2, h, u, v, w, \rho, \eta)$, a private key $\mathbf{gsk[i]} = (A_i, x_i, \pi, \mathbf{skp})$, a block $\boldsymbol{m}_j \in Z_p^k$ and this block's identifier $id_j \in \mathcal{I}$, user i computes the signature σ_j as follows:

1. Select $\alpha_j, \beta_j, r_{j,\alpha}, r_{j,\beta}, r_{j,x}, r_{j,\gamma_1}, r_{j,\gamma_2}$ as in HAGS.
2. Compute $T_{j,1}$, $T_{j,2}$, $T_{j,3}$, $\gamma_{j,1}$, $\gamma_{j,2}$, $R_{j,1}$, $R_{j,2}$, $R_{j,3}$, $R_{j,4}$ and $R_{j,5}$ as in HAGS.
3. Compute $\boldsymbol{\delta} = (\delta_1, ..., \delta_k) \leftarrow \mathtt{PRG}(sk_{prg}) \in Z_p^k$ and $b_j \leftarrow \mathtt{PRF}(sk_{prf}, id_j) \in Z_p$, then calculate the homomorphic MAC of block $\boldsymbol{m}_j = (m_{j,1}, ..., m_{j,k})$ as $t_j = \sum_{l=1}^{k} \delta_l \cdot m_{j,l} + b_j \in Z_p$.
4. Compute a challenge c_j for block $\boldsymbol{m}_j$ as $c_j = \eta^{t_j} \cdot H_1(T_{j,1}, ..., R_{j,5}) \in Z_p$.
5. Compute $s_{j,\alpha}$, $s_{j,\beta}$, $s_{j,x}$, s_{j,γ_1}, s_{j,γ_2} as in HAGS.
6. Compute a tag θ_j as $\theta_j = [H_2(id_j) g_1^{t_j}]^{\pi} \in G_1$.
7. Output a signature σ_j of this block $\boldsymbol{m}_j$ 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. To audit the integrity of shared data, a user first sends an auditing request to the TPA. After receiving an auditing request, the TPA generates an auditing message as follows:

1. Randomly pick a q-element subset $\mathcal{J}$ of set $[1, n]$ to locate the q selected blocks in this auditing task.
2. Generate a random $y_j \in Z_p$, for $j \in \mathcal{J}$.
3. Output an auditing message $\{(j, y_j)\}_{j \in \mathcal{J}}$, and send it to the cloud.

After receiving an auditing message, the cloud generates a proof of possession of selected blocks in shared data 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 $\{\boldsymbol{\mu}, \Theta, \Phi, \{id_j\}_{j \in \mathcal{J}}\}$, and send it to the TPA, where $\boldsymbol{\mu} = (\mu_1, ..., \mu_k)$.

ProofVerify. Given an auditing proof $\{\boldsymbol{\mu}, \Theta, \Phi, \{id_j\}_{j \in \mathcal{J}}\}$, an auditing message $\{(j, y_j)\}_{j \in \mathcal{J}}$, a group public key $\mathbf{gpk} = (g_1, g_2, h, u, v, w, \rho, \eta)$, a secret key pair $\mathbf{skp} = (sk_{prg}, sk_{prf})$, the TPA verifies this proof as follows:

1. Generate $\boldsymbol{\delta} = (\delta_1, ..., \delta_k) \leftarrow \mathtt{PRG}(sk_{prg}) \in Z_p^k$ and $b_j \leftarrow \mathtt{PRF}(sk_{prf}, id_j) \in Z_p$, 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

$$\prod_{j \in \mathcal{J}} R_{j,3}^{y_j} \stackrel{?}{=} e(\prod_{j \in \mathcal{J}} (T_{j,3}^{s_{j,x}} \cdot h^{-s_{j,\gamma_1} - s_{j,\gamma_2}} \cdot g_1^{-c_j})^{y_j}, g_2) \cdot e(\prod_{j \in \mathcal{J}} (h^{-s_{j,\alpha} - s_{j,\beta}} \cdot T_{j,3}^{c_j})^{y_j}, w), \tag{7}$$

$$\prod_{j \in \mathcal{J}} c_j^{y_j} \stackrel{?}{=} \eta^{\lambda} \cdot \prod_{j \in \mathcal{J}} H_1(T_{j,1}, ..., \tilde{R}_{j,5})^{y_j}, \tag{8}$$

$$e(\Theta, g_2) \stackrel{?}{=} e(\prod_{j \in \mathcal{J}} H_2(id_j)^{y_j} \cdot g_1^{\lambda}, \rho). \tag{9}$$

   If all three equations hold, the proof is valid. Otherwise, it is not.
5. If the proof is valid, the TPA sends a positive report to the user. Otherwise, she sends a negative report.

Open. Given a block $\boldsymbol{m}_j$ and a signature σ_j, the original user can reveal the identity of the signer on this block using her group manager private key $\mathbf{gmsk} = \{\xi_1, \xi_2\}$ as in HAGS.

Discussions. In Knox, the TPA is able to verify the integrity of shared data without retrieving the entire data. The original user can add new users to the group without re-computing any signature. Using the group manager's private key, the original user can reveal the identity of the signer on each block. While in previous work [16], the original user cannot disclose the identity of the signer because the identity is unconditional protected by ring signatures [8]. In addition, if the original user in previous work [16] wishes to add new users to the group, all signatures on shared data has to be recomputed, because the generation and verification of a ring signature require all the current group members' public keys.

User Revocation. Once a group user is misbehaved and her identity is revealed by the group manager, it is necessary to revoke this misbehaved user from the group. In our current mechanism, to revoke a group user from the group, the group manager needs to re-generate and re-distribute some parts of the private key for existing users, then all existing users need to re-sign their blocks in shared data with new private keys. The blocks previously signed by the revoked user should be re-signed by the group manager. Specifically, the group manager generates and distributes a new pair $(\pi', \mathbf{skp}')$ for existing users, then user i can compute group signatures with her new private key $\mathbf{gsk}'[\mathbf{i}] = (A_i, x_i, \pi', \mathbf{skp}')$; while the revoked user cannot compute valid group signatures anymore because she has no knowledge of $(\pi', \mathbf{skp}')$. The TPA will audit shared data with the new corresponding public key $\mathbf{gpk}' = (g_1, g_2, h, u, v, w, \rho', \eta)$, where $\rho' = g_2^{\pi'}$. In some special cases, the group manager herself may need to leave the group. Then the new group manager should compute new private keys for users and a

new public key for the new group, and all the users in the new group need to re-sign blocks in shared data with their new private keys.

6.3 Security Analysis of Knox

Theorem 5. *Given shared data M and its group signatures, a verifier is able to correctly check the integrity of shared data M.*

Proof. To prove the correctness of Knox is equivalent of proving Equation (7), (8) and (9) are all correct. Because Equation (1) is correct, it is clear that Equation (7) is also correct. Equation (8) can be expanded as follows:

$$\begin{aligned}
\prod_{j\in\mathcal{J}} c_j^{y_j} &= \prod_{j\in\mathcal{J}} \left(\eta^{t_j} \cdot H_1(T_{j,1}, ..., \tilde{R}_{j,5})\right)^{y_j} \\
&= \prod_{j\in\mathcal{J}} \eta^{t_j y_j} \cdot \prod_{j\in\mathcal{J}} H_1(T_{j,1}, ..., \tilde{R}_{j,5})^{y_j} \\
&= \eta^{\sum_{j\in\mathcal{J}} y_j (\sum_{l=1}^{k} \delta_l m_{j,l} + b_j)} \cdot \prod_{j\in\mathcal{J}} H_1(T_{j,1}, ..., \tilde{R}_{j,5})^{y_j} \\
&= \eta^{\sum_{l=1}^{k} \delta_l \mu_l + \sum_{j\in\mathcal{J}} y_j b_j} \cdot \prod_{j\in\mathcal{J}} H_1(T_{j,1}, ..., \tilde{R}_{j,5})^{y_j} \\
&= \eta^{\lambda} \cdot \prod_{j\in\mathcal{J}} H_1(T_{j,1}, ..., \tilde{R}_{j,5})^{y_j}.
\end{aligned}$$

Similar to the proof of Equation (8), the correctness of Equation (9) can be presented as

$$\begin{aligned}
e(\Theta, g_2) &= e(\prod_{j\in\mathcal{J}} \left(H_2(id_j) \cdot g_1^{t_j}\right)^{y_j}, g_2^{\pi}) \\
&= e(\prod_{j\in\mathcal{J}} H_2(id_j)^{y_j} \cdot \prod_{j\in\mathcal{J}} g_1^{t_j y_j}, \rho) \\
&= e(\prod_{j\in\mathcal{J}} H_2(id_j)^{y_j} \cdot g_1^{\lambda}, \rho).
\end{aligned}$$

All three equations are correct, therefore, a verifier in Knox is able to correctly check the integrity of shared data M.

Theorem 6. *Given shared data M and its group signatures, it is computational infeasible for an untrusted cloud or adversary to generate an auditing proof based on corrupted data M', where this auditing proof can pass the verification under Knox.*

Proof. Details of this proof can be found in our technical report [17].

Theorem 7. *Given shared data M and its group signatures, only the original user (the group manager) can reveal the identity of the signer on each block. For the TPA, it is computational infeasible to reveal the identity of the signer on each block during the auditing process.*

Proof. According to Theorem 4, for the TPA, who does not possess group manager's private key $\mathbf{gmsk} = (\xi_1, \xi_2)$, revealing the identity of the signer on each block during the auditing process is as hard as solving Decision Linear problem in G_1. The original user, who acts as the group manager, is able to trace the identity of the signer on each block using her group manager's private key.

7 Experimental Results

We now compare the performance of Knox with previous work, Oruta [16]. Due to space limitations, we only provide some experimental results in this section. Detailed analysis of computation and communication cost of Knox can be found in [17]. In our experiments, we utilize GMP and PBC libraries to implement cryptographic operations in Knox. All our experiments are tested on a 2.26 GHz Linux system over $1,000$ times. The security level is $|p| = 160$ bits. We also assume the total number of blocks in shared data is $n = 1,000,000$, each block contains $k = 100$ elements, the size of each block is 2KB and total size of shared data is 2GB. In the following experiments, we assume the number of selected blocks is $q = 300$, which allows the TPA to keep the detection probability greater than 95% if 1% of all the blocks are corrupted [3].

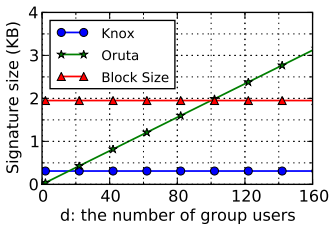

Fig. 2. Impact of group size d on signature size (KB).

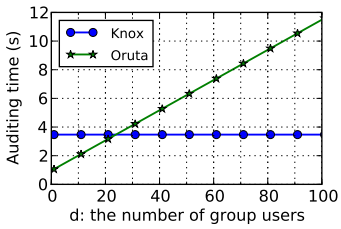

Fig. 3. Impact of group size d on auditing time (s).

As shown in Fig. 2, the signature size of Knox is independent from the number of users in a group. On the contrary, the signature size of Oruta is linearly increasing with the size of the group. Specifically, when $d = k$ in Oruta, the size of a signature is even the same as the size of a block.

In Fig. 3, we compare the auditing time of Knox and Oruta. In Knox, the auditing time is independent from the group size, while the auditing time in Oruta linearly increases with the size of the group. When the data in the cloud are shared by a large group, Knox requires less auditing time than Oruta. More specifically, when the group size is 100, Knox is able to finish an auditing task in less than 4 seconds while Oruta requires nearly 12 seconds to finish the same auditing task.

A detailed comparison of the auditing performance between Knox and Oruta is illustrated in Table 2, where $d = 100$ and $k = 100$. Although Knox requires less auditing time than Oruta, the communication cost of Knox is higher. However, it is still a small percentage of the entire size of shared data, which means the TPA can efficiently audit shared data without downloading the entire data. Our experimental results show that Knox has a better performance when auditing data shared among a large number of users.

Table 2. Comparison of Auditing Performance

	Oruta [16]	Knox
Data Storage Usage (GB)	2	
Signature Storage Usage (GB)	2	0.33
Communication Cost (KB)	18	106.4
Auditing Time (seconds)	11.49	3.44

8 Conclusion

In this paper, we propose Knox, a privacy-preserving auditing scheme for shared data with large groups in the cloud. We utilize group signatures to compute verification information on shared data, so that the TPA is able to audit the correctness of shared data, but cannot reveal the identity of the signer on each block. With the group manager's private key, the original user can efficiently add new users to the group and disclose the identities of signers on all blocks. The efficiency of Knox is not affected by the number of users in the group.

Acknowledgement. We are grateful to the anonymous reviewers for their helpful comments. This work is supported by the National Science and Technology Major Project (No. 2012ZX03002003), Fundamental Research Funds for the Central Universities (No. K50511010001), National 111 Program (No. B08038) and Doctoral Foundation of Ministry of Education of China (No. 20100203110002).

References

1. Agrawal, S., Boneh, D.: Homomorphic MACs: MAC-based Integrity for Network Coding. In: Proc. ACNS. pp. 292–305. Springer-Verlag (2009)
2. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R.H., Konwinski, A., Lee, G., Patterson, D.A., Rabkin, A., Stoica, I., Zaharia, M.: A View of Cloud Computing. Communications of the ACM 53(4), 50–58 (Apirl 2010)
3. Ateniese, G., Burns, R., Curtmola, R., Herring, J., Kissner, L., Peterson, Z., Song, D.: Provable Data Possession at Untrusted Stores. In: Proc. ACM CCS. pp. 598–610 (2007)

4. Ateniese, G., Camenisch, J., Joye, M., Tsudik, G.: A Practical and Provably Secure Coalition-Resistant Group Signature Scheme. In: Proc. CRYPTO. pp. 255–270. Springer-Verlag (2000)
5. Ateniese, G., Pietro, R.D., Mancini, L.V., Tsudik, G.: Scalable and Efficient Provable Data Possession. In: Proc. ICST SecureComm. pp. 1–10 (2008)
6. Boneh, D., Boyen, X., Shacham, H.: Short Group Signatures. In: Proc. CRYPTO. pp. 41–55. Springer-Verlag (2004)
7. Boneh, D., Freeman, D.M.: Homomorphic Signatures for Polynomial Functions. In: Proc. EUROCRYPT. pp. 149–168. Springer-Verlag (2011)
8. Boneh, D., Gentry, C., Lynn, B., Shacham, H.: Aggregate and Verifiably Encrypted Signatures from Bilinear Maps. In: Proc. EUROCRYPT. pp. 416–432. Springer-Verlag (2003)
9. Boneh, D., Lynn, B., Shacham, H.: Short Signatures from the Weil Pairing. In: Proc. ASIACRYPT. pp. 514–532. Springer-Verlag (2001)
10. Chaum, D., van Heyst, E.: Group Signatures. In: Proc. EUROCRYPT. pp. 257–265. Springer-Verlag (1991)
11. Erway, C., Kupcu, A., Papamanthou, C., Tamassia, R.: Dynamic Provable Data Possession. In: Proc. ACM CCS. pp. 213–222 (2009)
12. Ferrara, A.L., Green, M., Hohenberger, S., Pedersen, M.Ø.: Practical Short Signature Batch Verification. In: Proc. CT-RSA. pp. 309–324. Springer-Verlag (2009)
13. Hao, Z., Zhong, S., Yu, N.: A Privacy-Preserving Remote Data Integrity Checking Protocol with Data Dynamics and Public Verifiability. IEEE Transactions on Knowledge and Data Engineering 23(9), 1432–1437 (September 2011)
14. Juels, A., Kaliski, B.S.: PORs: Proofs pf Retrievability for Large Files. In: Proc. ACM CCS. pp. 584–597 (2007)
15. Shacham, H., Waters, B.: Compact Proofs of Retrievability. In: Proc. ASIACRYPT. pp. 90–107. Springer-Verlag (2008)
16. Wang, B., Li, B., Li, H.: Oruta: Privacy-Preserving Public Auditing for Shared Data in the Cloud. Tech. rep., University of Toronto (2011), `http://iqua.ece.toronto.edu/~bli/techreports/oruta.pdf`
17. Wang, B., Li, B., Li, H.: Knox: Privacy-Preserving Auditing for Shared Data with Large Groups in the Cloud. Tech. rep., University of Toronto (2012), `http://iqua.ece.toronto.edu/~bli/techreports/knox.pdf`
18. Wang, C., Wang, Q., Ren, K., Lou, W.: Ensuring Data Storage Security in Cloud Computing. In: Proc. IEEE IWQoS. pp. 1–9 (2009)
19. Wang, C., Wang, Q., Ren, K., Lou, W.: Privacy-Preserving Public Auditing for Data Storage Security in Cloud Computing. In: Proc. IEEE INFOCOM. pp. 525–533 (2010)
20. Wang, Q., Wang, C., Li, J., Ren, K., Lou, W.: Enabling Public Verifiability and Data Dynamic for Storage Security in Cloud Computing. In: Proc. European Symposium on Research in Computer Security. pp. 355–370. Springer-Verlag (2009)
21. Yang, K., Jia, X.: Data storage auditing service in cloud computing: challenges, methods and opportunities. World Wide Web 15(4), 409–428 (2012)
22. Yu, S., Wang, C., Ren, K., Lou, W.: Achieving Secure, Scalable, and Fine-grained Data Access Control in Cloud Computing. In: Proc. IEEE INFOCOM. pp. 534–542 (2010)
23. Zhu, Y., Hu, H., Ahn, G.J., Yau, S.S.: Efficient Audit Service Outsourcing for Data Integrity in Clouds. Journal of System and Software 85(5), 1083–1095 (May 2012)
24. Zhu, Y., Wang, H., Hu, Z., Ahn, G.J., Hu, H., Yau, S.S.: Dynamic Audit Services for Integrity Verification of Outsourced Storage in Clouds. In: Proc. ACM Symposium On Applied Computing. pp. 1550–1557 (2011)