11-2016

Ownership-hidden group-oriented proofs of storage from pre-homomorphic signatures

Yujue WANG
Singapore Management University, yjwang@smu.edu.sg

Qianhong WU

Bo QIN

Xiaofeng CHEN

Xinyi HUANG

See next page for additional authors

DOI: https://doi.org/10.1007/s12083-016-0530-8

Citation

WANG, Yujue; WU, Qianhong; QIN, Bo; CHEN, Xiaofeng; HUANG, Xinyi; and LOU, Jungang. Ownership-hidden group-oriented proofs of storage from pre-homomorphic signatures. (2016). *Peer-to-Peer Networking and Applications*. 1-17. Research Collection School Of Information Systems.
Available at: https://ink.library.smu.edu.sg/sis_research/3627

Published in Peer-to-Peer Networking and Applications, 7 November 2016, Advance Online
http://doi.org/10.1007/s12083-016-0530-8

Ownership-hidden group-oriented proofs of storage from pre-homomorphic signatures

Yujue Wang[1,2] · Qianhong Wu[3,4,5] · Bo Qin[6,7] · Xiaofeng Chen[7] · Xinyi Huang[8] · Jungang Lou[9]

Abstract In this paper, we study the problem of secure cloud storage in a multi-user setting such that the ownership of outsourced files can be hidden against the cloud server. There is a group manager for initiating the system, who is also responsible for issuing private keys for the involved group members. All authorized members are able to outsource files to the group's storage account at some cloud server. Although the ownership of outsourced file is preserved against the cloud server, the group manager could trace the true identity of any suspicious file for liability investigation. To address this issue, we introduce and formalize a notion of *ownership-hidden group-oriented proofs of storage* (OPoS). We present a generic OPoS construction from pre-homomorphic signatures, and propose an OPoS instantiation by employing the Boneh–Boyen short signature. We show that the OPoS instantiation can be optimized using a polynomial commitment technique, so that the integrity auditing protocol would only take constant-size communication overheads by the cloud server. Theoretical and experimental analyses show that our OPoS instantiations are efficient and practical for enterprise-oriented cloud storage applications. Also, we show that the OPoS instantiations can be enhanced to safeguard against a dynamic set of corrupted members, as well as support batch integrity auditing mechanism.

Keywords Cloud storage · Data outsourcing · Proofs of storage · Provable data possession · Proofs of retrievability · Public auditability

Qianhong Wu
qianhong.wu@buaa.edu.cn

[1] School of Information Systems, Singapore Management University, 178902, Singapore, Singapore

[2] Network and Data Security Key Laboratory of Sichuan Province, University of Electronic Science and Technology of China, Chengdu, 610054, China

[3] School of Electronic and Information Engineering, Beihang University, Beijing, China

[4] State Key Laboratory of Cryptology, P. O. Box 5159, Beijing, 100878, China

[5] State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, 100093, China

[6] School of Information, Renmin University of China, Beijing, China

[7] State Key Laboratory of Integrated Services Networks, Xidian University, Xi'an, China

[8] School of Mathematics and Computer Science, Fujian Normal University, Fuzhou, China

[9] School of Information Engineering, Huzhou University, Huzhou, 313000, China

1 Introduction

Many advanced information technology (e.g., handheld devices) and novel applications (e.g., social networks [1]) will create huge amounts of data. Common users who hold weak devices may not be affordable to maintain these data. A possible approach is to outsource user data to a remote server. In fact, cloud computing can provide such data outsourcing services to users, which is so attractive that the maintenance burden for users' local storage can be greatly reduced [2]. However, after data being outsourced, the remote cloud server has total control for the outsourced files, which will raise security concerns for the file owners and users in this data outsourcing scenario, including data integrity [3, 4], data privacy [5, 6], access control [2, 5], data update [7, 8], data deduplication [9–11], user authentication [12, 13], and key management [14, 15], etc.

To safeguard data integrity in clouds, many cryptographic primitives have been introduced such as Proofs of Storage [4], Provable Data Possession (PDP) [3] and Proofs of Retrievability (PoR) [16]. Also, in the literature, many PoS/PDP/PoR schemes have been proposed [16–20]. However, there exists no publicly auditable scheme that is applicable to an enterprise-oriented application scenario, where the ownership of outsourced files can be hidden against the cloud server. To exemplify the problem, consider the following application: A company purchases remote storage service from some cloud service provider to store company files. The IT department of the company is responsible for authorizing company employees so that they can outsource files to the company's account. From the view of an outsider (e.g., the cloud server), these outsourced files are the properties of the company, which requires that their true ownership cannot be leaked to the cloud server. On the other hand, to enable the company to trace dishonest or even malicious outsourcing behavior if necessary, there also needs a mechanism to include some specific information (e.g., identity) of the file owner to the outsourced file.

1.1 Contribution

In this paper, we introduce the notion of *ownership-hidden group-oriented proofs of storage* (OPoS) and formalize its framework. An OPoS system has the following functionalities:

– Each authorized member would get a private key that associated with his/her unique identity from a trusted group manager, and would not share any private parameter with other members. The private key can be locally validated using the public key of the group manager.
– With the private key, each group member can locally process file to produce aggregatable meta-data which will be used in auditing integrity.
– The OPoS system should be publicly auditable, that is, anyone who knows the ownership of some file can audit that file by interacting with the cloud server. The integrity auditing can be performed for unbounded times and can get overwhelming auditing probability without touching the whole file.
– The ownership of outsourced file is hidden from the cloud storage server, so that the server can only know that the files are outsourced to the company's account. While the ownership can be revealed by the group manager, as required for executing liability investigation when needed.

We formalize the security model for publicly auditable OPoS system and present a generic OPoS construction from pre-homomorphic signatures. The system should safeguard against conspiracy attacks for forging private keys and meta-data launched by a static set of colluding group members, and against the cloud storage server for forging an integrity proof in auditing outsourced files and for extracting the files' ownership. We then present a generic OPoS construction by employing pre-homomorphic signatures with homomorphic composability and homomorphic verifiability as building block, and prove its security in the formalized security model.

We present two efficient (basic and optimized) OPoS instantiations, where the building block of pre-homomorphic signature scheme is replaced by the Boneh–Boyen short signature scheme [21]. In detail, Boneh–Boyen scheme is run by the group manager for issuing private keys for group members, that is, the private key is exactly a signature of the member's identity in the group. The basic OPoS instantiation takes linearly computation and communication complexities. By using a polynomial commitment technique [22], the basic instantiation built on symmetric bilinear groups is optimized for reducing communication overheads and computation costs in auditing outsourced files. In the optimized instantiation, a commitment to the aggregated file block is sent to the auditor, whereas the aggregated file block is directly sent back in the basic instantiation.

We also analyze the performance of our OPoS instantiations. The theoretical analysis shows that the optimized instantiation greatly saves the communication costs in integrity auditing compared to the basic one, at the same time it would also bring some additional but affordable computation burden to the cloud server. In fact, in the optimized instantiation, the proof to a integrity challenge has constant size, which is independent of the sector number and the number of the challenged file blocks. The experimental results show that the file splitting manner (i.e., the sector number in a file block) determines file processing performance, for example, for processing a 10M file, the more sectors in a block, the higher efficiency can be achieved. These analyses confirm that our OPoS instantiations are practical to support company-oriented cloud storage applications.

We provide further discussions on our OPoS instantiations. First, we show that the security of the above mentioned OPoS instantiations can be strengthened. Those two OPoS instantiations are secure against a static set of colluding group members, which is due to the underlying pre-homomorphic signature is weakly secure. By employing a full short signature of Boneh and Boyen as pre-homomorphic signature, the security can be strengthened as resistant against a dynamic set of corrupted group members, without sacrificing file processing efficiency. Second, we show that our OPoS instantiations support batch integrity auditing mechanism, which is much more efficient than auditing multiple files separately in terms of both computation costs and communication overheads for the auditor.

Compared to the preliminary version [23], this paper has the following changes: First, some details of the security model, construction and security proof are modified. Second, the experiments are conducted and analyzed for evaluating the performance of the proposed OPoS instantiations. Third, we show that the security of the presented OPoS instantiations can be strengthened. Also, we discuss that the proposed OPoS instantiations support batch integrity auditing mechanism.

1.2 Related work

Many studies have been conducted on the problem of integrity auditing for outsourced files. In the literature, many schemes are proposed in a single-user setting. Ateniese et al. [3] and Juels and Kaliski [16] independently introduced PDP and PoR, respectively, in cloud storage setting. PoR is stronger than PDP that PoR supports data retrieveability, which can take integrity auditing in PDP as a special case. Shacham and Waters [19] investigated both privately and publicly auditable PoR schemes with strong security proofs. In Xu and Chang's privately auditable PoR [24], the polynomial commitment technique [22] is used so that the communication overheads in integrity auditing are greatly saved compared to the schemes of [19]. Also, Yuan and Yu [20] presented a publicly auditable PoR using the same commitment technique. Wang et al. [25] offloaded PDP schemes by outsourcing expensive computations (i.e., exponentiations) to a computation server.

There are also some proposals that support data outsourcing and integrity auditing in clouds in multi-user setting. Wang, Li, and Li [26] researched data sharing problem in clouds for a group of members. Their scheme can hidden the group member's identity when auditing the integrity, yet it cannot support public auditability or identity-based deployment. The same problem was revisited using ring signatures in [27], where each group member should locally prepare his/her private key rather than generate by a group manager. In Wang et al.'s scheme [28], each group member does not hold private key and should interact with a security-mediator for processing a file. The scheme presented by Wang et al. [29] requires the auditor to hold a secret parameter of the file owner for conducting integrity auditing. Their scheme [29] has larger secret key size for each group member than our OPoS instantiations, that is, each key consists of two elements in $\mathbb{G}$ and $\mathbb{Z}_p$ in their scheme, while there is only one element in $\mathbb{G}$ in our instantiations. Yu et al.'s proposal [30] can hidden the ownership against the third party auditor in integrity auditing, but the group members should locally prepare private keys and jointly negotiate a pair of group public/private keys.

2 Preliminaries

2.1 Mathematical background

Suppose $\mathbb{G}_1 = \langle g_1 \rangle$ and $\mathbb{G}_2 = \langle g_2 \rangle$ are (multiplicative) cyclic groups with prime order p and efficient group actions. The groups $(\mathbb{G}_1, \mathbb{G}_2)$ are bilinear if there exists a cyclic group $\mathbb{G}_T$ with the same order and an efficient bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ such that: (1) bilinearity: $\forall h_1 \in \mathbb{G}_1, \forall h_2 \in \mathbb{G}_2$, and $\forall \alpha, \beta \in \mathbb{Z}_p^*$, $\hat{e}(h_1^\alpha, h_2^\beta) = \hat{e}(h_1, h_2)^{\alpha\beta}$; (2) non-degeneracy: $\hat{e}(g_1, g_2) \neq 1$.

Our OPoS scheme and instantiations rely on the following computational assumptions.

Discrete logarithm (DL) assumption Let $\mathbb{G} = \langle g \rangle$ be a cyclic group with prime order p. Given a random element $h \in_R \mathbb{G}$, any probabilistic polynomial time (PPT) algorithm $\mathcal{A}$ would have negligible probability to compute $x \in \mathbb{Z}_p^*$ such that $h = g^x$.

s-Strong Diffie–Hellman (s-SDH) assumption [21] Let $\mathbb{G} = \langle g \rangle$ be a cyclic groups with prime order p. Given a $(s+1)$-tuple $(g, g^\alpha, \cdots, g^{\alpha^s}) \in \mathbb{G}^{s+1}$ for $\alpha \in_R \mathbb{Z}_p^*$, any PPT algorithm $\mathcal{A}$ would have negligible probability to compute a pair $(z, g^{\frac{1}{\alpha+z}})$, where $z \in \mathbb{Z}_p^* \setminus \{-\alpha\}$.

2.2 Pre-homomorphic signature

In [23], we identified two useful properties such as *homomorphic composability* and *homomorphic verifiability* for some signatures, and presented a generic OPoS construction using the signature scheme with these properties as building block. We notice that these properties have been used in [31] to design generic homomorphic signature scheme, where a scheme with these properties is termed as pre-homomorphic signature scheme.

Let $\mathcal{S} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vrfy})$ be a pre-homomorphic signature scheme defined on some cyclic group $\mathbb{G} = \langle g \rangle$, and $\mathcal{M}$ and $\mathcal{R}$ be the message space and randomness space sampled by algorithm Sign of $\mathcal{S}$, respectively. Using $\mathcal{S}$, a signature for some message $m \in \mathcal{M}$ must have a component of the form $g^{\varphi(m,r)}$, where $\varphi(\cdot)$ is a function that may rely on the private key and $r \in_R \mathcal{R}$.

Definition 1 (**Homomorphic composability**) A signature scheme $\mathcal{S}$ is φ-homomorphic composable if

(1) there exists an efficient function $\varphi_{\mathsf{sk}} : \mathcal{M} \times \mathcal{R} \rightarrow \mathbb{Z}$; and

(2) the signing algorithm is defined as $\mathcal{S}.\mathsf{Sign}_{\mathsf{sk}} : \mathcal{M} \times \mathcal{R} \rightarrow \mathbb{G} \times \{0,1\}^*$. That is, for each message $m \in_R \mathcal{M}$ and every key pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{S}.\mathsf{KGen}(1^\lambda)$, the signature has the form $\sigma = (\sigma_1, \sigma_2)$ such that $\sigma_1 = g^{\varphi(m,r)}$ and σ_2 may be empty.

Definition 2 (**Homomorphic verifiability**) A signature scheme $\mathcal{S}$ is φ-homomorphic verifiable if

(1) there is an efficient test algorithm $\Xi(\mathsf{pk}; m, \sigma; \ddot{x}, \ddot{y})$ which takes a public key pk, a message/signature pair $(m, \sigma = (\sigma_1, \sigma_2))$, and a pair of elements $(\ddot{x}, \ddot{y}) \in \mathbb{G}'^2$, where $\mathbb{G}'$ is also a cyclic group with the same order of $\mathbb{G}$; and

(2) the algorithm Ξ outputs "1" if the given pair (m, σ) is valid under pk, that is, $\mathcal{S}.\mathsf{Vrfy}(\mathsf{pk}, m, \sigma) = 1$, and $\log_g \sigma_1 = \log_{\ddot{x}} \ddot{y}$. Otherwise, outputs "0".

As noted in [31], there exist many pre-homomorphic signature schemes that enjoy the properties of homomorphic composability and homomorphic verifiability. We then provide an exemplary signature scheme that will be used to instantiate our OPoS scheme.

Boneh–Boyen scheme [21]. Let $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ be an asymmetric bilinear map, where $\mathbb{G}_1 = \langle g_1 \rangle$, $\mathbb{G}_2 = \langle g_2 \rangle$ and $\mathbb{G}_T$ are (multiplicative) cyclic groups with prime order p.

– $\mathsf{KGen}(1^\lambda)$: Pick a random value $\gamma \in_R \mathbb{Z}_p^*$ as the secret key sk, and compute $\varpi = g_2^\gamma$. The public key is $\mathsf{pk} = (\hat{e}, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, p, \varpi)$.
– $\mathsf{Sign}(\mathsf{pk}, \mathsf{sk}, m)$: Given a message $m \in_R \mathbb{Z}_p$, compute the signature $\sigma = g_1^{1/(\gamma+m)}$. If $m + \gamma = 0$, then set the signature as the identity element in $\mathbb{G}_1$.
– $\mathsf{Vrfy}(\mathsf{pk}, m, \sigma)$: If $\hat{e}(\sigma, \varpi \cdot g_2^m) \overset{?}{=} \hat{e}(g_1, g_2)$ holds, then σ is valid for m and thus output "1"; otherwise, output "0".

If let $\varphi_{\mathsf{sk}}(m, \cdot) = \frac{1}{\gamma+m} \mod p$ and the second signature component $\sigma_2 = \varnothing$, then the signature scheme clearly satisfies homomorphic composability property. By defining the testing algorithm Ξ to output "1" if and only if both $\hat{e}(\sigma, \varpi \cdot g_2^m) = \hat{e}(g_1, g_2)$ and $\hat{e}(\ddot{y}, \varpi \cdot g_2^m) = \hat{e}(\ddot{x}, g_2)$ hold, it also satisfies homomorphic verifiability.

3 System model and definitions

In this section, we define OPoS system model and the corresponding security requirements.

3.1 System model

As shown in Fig. 1, an OPoS system consists of four types of entities such as a cloud storage server, a group manager, many group members and an auditor. The cloud storage server is an untrusted party that offers remote storage services to users. It also has powerful computation capability in responding (integrity) requests from cloud users. The group manager initiates the system and is trusted by its members for issuing secret keys for them. Each group member may hold some files and would like to outsource them to the cloud storage server on behalf of the group. The auditor is also a cloud user, but may not be a group member. The auditor has access to all the public information of the group and the outsourced files, so that the auditor is able to audit these outsourced files on behalf of group manager and file owners. The auditor will not collude with the cloud storage server. We note this assumption is necessary since if the auditor colludes with the server, then any scheme cannot be secure as the auditor can cheat the users with wrong verification results.

3.2 System definition

A publicly verifiable OPoS scheme comprises six polynomial time efficient procedures, such as Setup, KeyExt, PrFile, Chall, PrfGen and Verify.

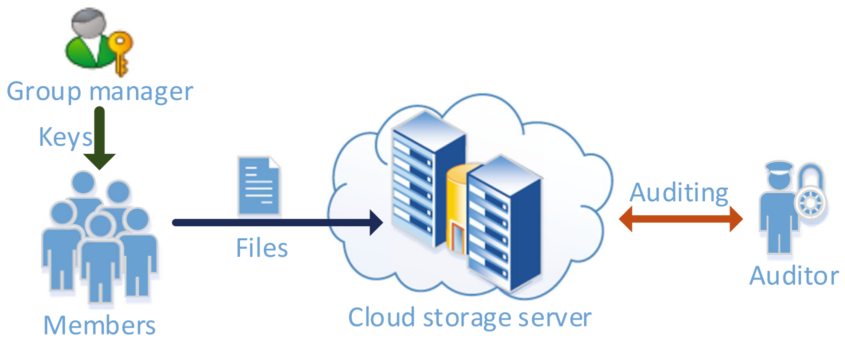

Fig. 1 OPoS system model

- $(\mathsf{gpk}, \mathsf{gsk}) \leftarrow \mathsf{Setup}(1^\lambda)$: On input a security parameter λ, the group manager runs the system setup procedure to generate a pair of group public key and group private key $(\mathsf{gpk}, \mathsf{gsk})$.
- $\mathsf{sk}_\ell \leftarrow \mathsf{KeyExt}(\mathsf{gpk}, \mathsf{gsk}, \mathsf{id}_\ell)$: On input a group public key gpk, a group private key gsk and a member identity id_ℓ, the group manager runs the key extraction procedure to produce a private key sk_ℓ for id_ℓ. This private key is verifiable by id_ℓ using the group public key gpk.
- $(\tau, F^*) \leftarrow \mathsf{PrFile}(\mathsf{gpk}, \mathsf{sk}_\ell, F)$: On input a group public key gpk, a member's private key sk_ℓ and a file $F \in \{0, 1\}^*$, the file owner id_ℓ (a group member) runs the file processing procedure to produce a file tag τ and a processed file F^* that comprises F and a number of meta-data $\vec{\sigma}$.
- $C \leftarrow \mathsf{Chall}(\mathsf{gpk}, \tau)$: On input a group public key gpk and a file tag τ, the auditor runs the challenge generation procedure to generate a challenge C.
- $R \leftarrow \mathsf{PrfGen}(\mathsf{gpk}\ , F^*, C)$: On input a group public key gpk, a processed file F^* and a challenge C, the cloud storage server runs the proof generation procedure to produce a proof R.
- $0/1 \leftarrow \mathsf{Verify}(\mathsf{gpk}, \tau, C, R)$: On input a group public key gpk, a file tag τ and a pair of challenge and proof (C, R), the auditor runs the verification procedure to output "1" if R is a valid proof for C, or "0" otherwise.

An OPoS scheme must be able to successfully audit the integrity of all files that outsourced by any group member.

Definition 3 (**Correctness**) An OPoS scheme $(\mathsf{Setup}$, KeyExt, PrFile, Chall, PrfGen, $\mathsf{Verify})$ is *correct* if for any group key pair $(\mathsf{gpk}, \mathsf{gsk}) \leftarrow \mathsf{Setup}(1^\lambda)$, any group member id_ℓ with private key $\mathsf{sk}_\ell \leftarrow \mathsf{KeyExt}(\mathsf{gpk}, \mathsf{gsk}, \mathsf{id}_\ell)$, and any file $F \in \{0, 1\}^*$, let $(\tau, F^*) \leftarrow \mathsf{PrFile}(\mathsf{gpk}, \mathsf{sk}_\ell, F)$, the verification equation $\mathsf{Verify}(\mathsf{gpk}, \tau, C, \mathsf{PrfGen}(\mathsf{gpk}, F^*, C)) = 1$ holds for any challenge $C \leftarrow \mathsf{Chall}(\mathsf{gpk}, \tau)$.

3.3 Security definitions

An OPoS system would confront three types of conspiracy attacks launched by group members and the cloud storage server.

- Private key forgery: Several group members may collude to forge a private key for another group member.
- Meta-data forgery: The group members or the cloud storage server may forge meta-data for some file of another group member.
- Proof forgery: The cloud storage server and group members may collude to forge a proof when auditing the integrity of some file outsourced by another group member.

Note that a forged meta-data implies a forged proof in integrity auditing for the same file. Thus, the second type of attacks about meta-data forgery can be captured by the third case. Hence, we only need to consider the private key forgery and proof forgery in defining the security model. They are captured by the following security game that is carried out by a PPT adversary $\mathcal{A}$ and a challenger $\mathcal{C}$. Initially, the adversary controls a static set S_c of corrupted group members.

Setup: The adversary gives the corrupted member set S_c to $\mathcal{C}$. Challenger runs $\mathsf{Setup}(1^\lambda)$ to generate a pair of group public/private keys $(\mathsf{gpk}, \mathsf{gsk})$, and generates private keys $\{\mathsf{sk}_i : \mathsf{id}_i \in S_c\}$ for all corrupted members. Challenger $\mathcal{C}$ gives gpk and $\{\mathsf{sk}_i : \mathsf{id}_i \in S_c\}$ to $\mathcal{A}$.

Queries: The following queries can be requested adaptively. The challenger records all the intermediate information.

- *File processing:* For each query with a file F and a member identity $\mathsf{id}_\ell \notin S_c$, the challenger runs procedure $\mathsf{KeyExt}(\mathsf{gpk}, \mathsf{gsk}, \mathsf{id}_\ell)$ to get sk_ℓ and computes $(\tau, F^*) \leftarrow \mathsf{PrFile}(\mathsf{gpk}, \mathsf{sk}_\ell, F)$. Then, the challenger sends (τ, F^*) to $\mathcal{A}$. For each query (F, id_ℓ), there is a unique file identifer in the file tag τ, which is randomly chosen by $\mathcal{C}$ for guaranteeing its uniqueness.
- *Integrity auditing:* For any processed file in above processing file queries, the challenger (acting as the auditor) can audit its integrity by challenging the adversary (acting as the prover). That is, they jointly carry out the integrity auditing protocol. In detail, for any file F in the query list, the challenger can challenge $\mathcal{A}$ with $C \leftarrow \mathsf{Chall}(\mathsf{gpk}, \tau)$, and $\mathcal{A}$ sends back a proof R. Then, the challenger verifies R by invoking $\mathsf{Verify}(\mathsf{gpk}, \tau, C, R)$ and gives the results to $\mathcal{A}$.

End-Game: Finally, the adversary outputs a private key sk'_ℓ for some member id'_ℓ, or a pair of challenge/proof (C', R') for some file F' with file tag τ'.

Definition 4 (**Soundness**) An OPoS scheme is *sound* if for any PPT adversary $\mathcal{A}$ who plays the above mentioned security game by interacting with $\mathcal{C}$, the outputs are neither of the following cases:

- **Case 1.** The private key sk'_ℓ is valid under the group public key gpk for an uncorrupted member $\mathsf{id}'_\ell \notin S_c$.
- **Case 2.** The pair of challenge/proof (C', R') is valid but R' does not equal to that generated from the maintained information by $\mathcal{C}$, where the associated identity $\mathsf{id}' \notin S_c$.

An OPoS scheme also requires that, in the entire life span of an outsourced file, its owner identity should be hidden from the cloud storage server. Essentially, this *ownership-hidden* property requires the files should be uploaded in the name of the group. As in the real application scenario, the files are uploaded by the employee under the company's account. Also, in auditing an outsourced file, the cloud storage server should be able to respond the integrity challenge without using its owner's identity. More technically,

Definition 5 (**Ownership privacy**)An OPoS scheme is *ownership-hidden* against the cloud storage server if for any file $F \in \{0,1\}^*$ and any two distinct members $\mathsf{id}_{\ell,1}$ and $\mathsf{id}_{\ell,2}$ in the same group, the following two distributions are identical from the view of the cloud storage server:

$$\left\{\sigma_1 : \begin{array}{l} (\mathsf{gpk}, \mathsf{gsk}) \leftarrow \mathsf{Setup}(1^\lambda), \\ \mathsf{sk}_{\ell,1} \leftarrow \mathsf{KeyExt}(\mathsf{gpk}, \mathsf{gsk}, \mathsf{id}_{\ell,1}), \\ (\tau_1, F_1^*) \leftarrow \mathsf{PrFile}(\mathsf{gpk}, \mathsf{sk}_{\ell,1}, F) \end{array}\right\}$$

and

$$\left\{\sigma_2 : \begin{array}{l} (\mathsf{gpk}, \mathsf{gsk}) \leftarrow \mathsf{Setup}(1^\lambda), \\ \mathsf{sk}_{\ell,2} \leftarrow \mathsf{KeyExt}(\mathsf{gpk}, \mathsf{gsk}, \mathsf{id}_{\ell,2}), \\ (\tau_2, F_2^*) \leftarrow \mathsf{PrFile}(\mathsf{gpk}, \mathsf{sk}_{\ell,2}, F) \end{array}\right\}$$

4 OPoS construction

4.1 Overview

In our OPoS construction, the private keys for all members are generated and distributed by a (trusted) group manager. These private keys are associated with their identities, that is, the group manager signs the identity of each member and outputs the signature as the private key for that member. An advantage of this approach is that it is naturally collusion-resistant in the sense that even colluding users cannot forge a valid private key if the underlying signature is existentially unforgeable. However, this approach also brings a challenge in constructing OPoS schemes, not only because the files are processed by group members with their private keys and each meta-data should incorporate at least a file identifier and a file block with many sectors, but also the produced meta-data should be aggregatable and publicly auditable. In fact, things would be even worse in designing generic OPoS construction equipped with user privacy.

Our OPoS construction takes pre-homomorphic signatures as building block, so that a group member id_ℓ only holds his/her private key in the form of $\mathsf{sk}_\ell = (\mathsf{sk}_{\ell,1}, \mathsf{sk}_{\ell,2})$ such that $\mathsf{sk}_{\ell,1} = g^{\varphi_{\mathsf{gsk}}(\mathsf{id}_\ell, r)}$. Thus, the member cannot directly use gsk or $\varphi_{\mathsf{gsk}}(\mathsf{id}_\ell, r)$ for producing meta-data (a signature for each file block). We circumvent this issue by employing trapdoor technique. That is, the group member id_ℓ randomly picks (secret) values $\alpha_i \in_R \mathbb{Z}$ and computes the corresponding powers $u_i = g^{\alpha_i} \in \mathbb{G}$ as public parameters. When signing on a list of messages $\{m_i\}$, the member id_ℓ first evaluates a linear function $f_e = f(\{m_i, \alpha_i\}) = \sum \alpha_i m_i$ over $\mathbb{Z}$, and then computes

$$\sigma' = \mathsf{sk}_{\ell,1}^{f_e} = (g^{\varphi_{\mathsf{gsk}}(\mathsf{id}_\ell, r)})^{f_e} = (g^{f_e})^{\varphi_{\mathsf{gsk}}(\mathsf{id}_\ell, r)}$$

In this way, the component g^{f_e} can be recovered using public parameters $\{u_i\}$ and messages $\{m_i\}$. Hence, under the group public key gpk, the test algorithm Ξ can go through as $\Xi(\mathsf{gpk}; \mathsf{id}_\ell, \mathsf{sk}_{\ell,1}; g^{f_e}, \sigma') = 1$.

4.2 Construction

Let $\mathcal{S}_{ph} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vrfy})$ be a secure pre-homomorphic signature scheme which has the properties described in Section 2.2. Also let $\mathcal{S}_t = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vrfy})$ be a secure standard signature scheme. Suppose the group comprises n members and $H_0 : \{0,1\}^* \to \mathbb{Z}$ is a collision-resistant hash function.

$\mathsf{Setup}(1^\lambda)$: The group manager invokes $\mathcal{S}_{ph}.\mathsf{KGen}(1^\lambda)$ to obtain a pair of public/private keys, and sets them as the group public key gpk and group private key gsk, respectively.

$\mathsf{KeyExt}(\mathsf{gpk}, \mathsf{gsk}, \mathsf{id}_\ell)$: For each member id_ℓ $(1 \le \ell \le n)$ in the group, the group manager computes a signature of his/her identity as follows:

$$\mathsf{sk}_\ell = (\mathsf{sk}_{\ell,1}, \mathsf{sk}_{\ell,2}) \leftarrow \mathcal{S}_{ph}.\mathsf{Sign}(\mathsf{gsk}, \mathsf{id}_\ell) \in \mathbb{G}$$

Such a signature serves as his/her private key. When receiving sk_ℓ, the member id_ℓ is able to validate it by invoking $\mathcal{S}_{ph}.\mathsf{Vrfy}$ with the group public key gpk.

$\mathsf{PrFile}(\mathsf{gpk}, \mathsf{sk}_\ell, F)$: Given a file F, the member id_ℓ splits it into r blocks such that each block has s sectors (as elements in $\mathbb{Z}$) as follows

$$F = \{\mathbf{f}_i = (f_{i,1}, \cdots, f_{i,s}) : 1 \le i \le r\} \tag{1}$$

Chooses a unique file identifier $fid \in_R \mathbb{Z}$. The member also picks $(s+1)$ random values $\alpha_0, \alpha_1, \cdots, \alpha_s \in_R \mathbb{Z}$ and computes $u_j = g^{\alpha_j} \in \mathbb{G}$ for each $0 \le j \le s$. Let τ_0 be the concatenation string of $(\mathsf{gpk}, \mathsf{id}_\ell, u_0, u_1, \cdots, u_s, fid, r)$. Generates the file tag by the following steps:

- Computes $(\mathsf{tpk}, \mathsf{tsk}) \leftarrow \mathcal{S}_t.\mathsf{KGen}(1^\lambda)$ to obtain a pair of public/private keys.
- Computes $\vartheta \leftarrow \mathcal{S}_t.\mathsf{Sign}(\mathsf{tsk}, \tau_0)$ to obtain a signature of string τ_0.
- Sends the file tag $\tau = (\tau_0, \mathsf{tpk}, \vartheta)$ to the group manager.

Then, for each file block $\vec{f_i}$ $(1 \le i \le r)$, computes

$$\theta_i = \alpha_0 H_0(fid\|i) + \sum_{j=1}^{s} \alpha_j f_{i,j} \in \mathbb{Z}$$

and generates meta-data as $\sigma_i \leftarrow \mathsf{sk}_{\ell,1}^{\theta_i} \in \mathbb{G}$. Sends the processed file $F^* = \{(\mathbf{f}_i, \sigma_i) : 1 \leq i \leq r\}$ to the cloud storage server. Deletes the random values $\alpha_0, \alpha_1, \cdots, \alpha_s$, private key tsk and the file locally.

Chall(gpk, τ): The auditor invokes $\mathcal{S}_t$.Vrfy(tpk, τ_0, ϑ) to validate the file tag τ. If it is invalid, outputs "0" and terminates. Otherwise, picks a random subset $Q \subseteq [1, r]$ and chooses a random value $\beta_i \in_R \mathbb{Z}$ for each $i \in Q$. Sends the challenge $C = (fid, Q, \{\beta_i : i \in Q\})$ to the cloud storage server.

PrfGen(gpk, F^*, C): The cloud storage server computes the aggregated file block $\mu = (\mu_1, \cdots, \mu_s)$ and meta-data σ as follows:

$$\mu_j = \sum_{i\in Q} \beta_i f_{i,j} \in \mathbb{Z} \text{ for each } j \in [1, s]$$

and

$$\sigma = \prod_{i\in Q} \sigma_i^{\beta_i} \in \mathbb{G} \tag{2}$$

Sends the proof $R = (\vec{\mu}, \sigma)$ to the auditor.

Verify(gpk, τ, C, R): If R cannot be parsed, outputs "0" and terminates. Otherwise, checks

$$\Xi(\mathsf{gpk}; \mathsf{id}_\ell, \mathsf{sk}_{\ell,1}; u_0^{\sum_{i\in Q}\beta_i H_0(fid\|i)} \cdot \prod_{j=1}^{s} u_j^{\mu_j}, \sigma) \overset{?}{=} 1 \tag{3}$$

If so, outputs "1"; otherwise, outputs "0".

4.3 Security analysis

Theorem 1 *The proposed OPoS scheme is correct.*

Proof We only show (3) holds as the other parts are straightforward. As $\mathsf{sk}_{\ell,1} = g^{\varphi_{\mathsf{gsk}}(\mathsf{id}_\ell, r)}$, we have

$$\begin{aligned}\sigma &= \prod_{i\in Q}\sigma_i^{\beta_i} = \prod_{i\in Q}(g^{\varphi_{\mathsf{gsk}}(\mathsf{id}_\ell,r)})^{\theta_i\beta_i}\\ &= (g^{\sum_{i\in Q}\beta_i(\alpha_0 H_0(fid\|i)+\sum_{j=1}^{s}\alpha_j f_{i,j})})^{\varphi_{\mathsf{gsk}}(\mathsf{id}_\ell,r)}\\ &= (\prod_{i\in Q} g^{\beta_i\alpha_0 H_0(fid\|i)} \cdot \prod_{j=1}^{s} u_j^{\sum_{i\in Q}\beta_i f_{i,j}})^{\varphi_{\mathsf{gsk}}(\mathsf{id}_\ell,r)}\\ &= (u_0^{\sum_{i\in Q}\beta_i H_0(fid\|i)} \cdot \prod_{j=1}^{s} u_j^{\mu_j})^{\varphi_{\mathsf{gsk}}(\mathsf{id}_\ell,r)}\end{aligned}$$

Therefore, σ has the same component $\varphi_{\mathsf{gsk}}(\mathsf{id}_\ell, r)$ as the private key $\mathsf{sk}_{\ell,1}$. According to the definition of algorithm Ξ, the correctness follows. □

Theorem 2 *Suppose that the signature scheme $\mathcal{S}_t$ for file tags and the pre-homomorphic signature scheme $\mathcal{S}_{ph}$ are existentially unforgeable. The proposed OPoS construction is sound under the DL assumption.*

Proof In our construction, the private key of group member is generated by using a secure pre-homomorphic signature scheme $\mathcal{S}_{ph}$. Thus, the unforgeability of these keys are guaranteed by the security of $\mathcal{S}_{ph}$. Hence, the following discussion focuses only on the soundness for the integrity auditing of outsourced files.

Suppose a PPT adversary $\mathcal{A}$, who controls a set S_c of group members, can break the soundness of the proposed generic OPoS construction. Let the adversary play the security game described in Definition 4 by interacting with a challenger $\mathcal{C}$. We show that when the adversary outputs a (forged) valid pair of challenge/proof (C', R') for some file F' and a tag τ', $\mathcal{A}$ must have broken the DL assumption.

Setup: With a security parameter λ, the challenger generates a pair of group public key and private key (gpk, gsk) and the private keys for all member in S_c. Then, challenger $\mathcal{C}$ sends gpk and $\{\mathsf{sk}_i : \mathsf{id}_i \in S_c\}$ to $\mathcal{A}$.

Queries: The adversary adaptively interacts with the challenger $\mathcal{C}$ on the following queries, where $\mathcal{C}$ maintains all the intermediate information.

– *File processing:* For each query (F, id_ℓ) such that $\mathsf{id}_\ell \notin S_c$, if id_ℓ has not been involved in previous queries, then the challenger first invokes KeyExt to extract a private key sk_ℓ for id_ℓ. The challenger runs algorithm PrFile to process F with sk_ℓ, and then gives the produced file tag τ and a list of meta-data $\{\sigma_i\}_{1\leq i\leq r}$ to $\mathcal{A}$, where the file identifier fid (an element in τ) is randomly chosen by $\mathcal{C}$.
– *Integrity auditing:* For this type of queries, the challenger and the adversary play the roles as a verifier and a prover, respectively. For any file F that has been queried for processing, the challenger $\mathcal{C}$ can run Chall(gpk , τ) to generate $C = (fid, Q, \{\beta_i : i \in Q\})$ and challenge $\mathcal{A}$ using C. The adversary responds with a proof $R = (\mu, \sigma)$. The challenger verifies P by running Verify(gpk, τ, C, R) and gives the verification result to $\mathcal{A}$.

End-Game: Finally, the adversary outputs a forged pair of challenge/proof (C', P') for some file F' and a tag $\tau' = (\tau_0', \mathsf{tpk}, \vartheta')$, such that $C' = (fid, Q, \{\beta_i : i \in Q\})$ and $R' = (\mu', \sigma')$ satisfy the testing algorithm Ξ. File F' should have been queried for processing. Assume it belongs to some member $\mathsf{id}_\ell \notin S_c$ and has identifier fid, both of which are specified in τ'. As we discussed, the forged proof σ' and real proof σ have the following representations, respectively

$$\begin{aligned}\sigma' &= (u_0^{\sum_{i\in Q}\beta_i H_0(fid\|i)} \cdot \prod_{j=1}^{s} u_j^{\mu'_j})^{\varphi_{\mathsf{gsk}}(\mathsf{id}_\ell,r)}\\ &= (g^{\varphi_{\mathsf{gsk}}(\mathsf{id}_\ell,r)})^{\alpha_0\sum_{i\in Q}\beta_i H_0(fid\|i)+\sum_{j=1}^{s}\alpha_j\mu'_j}\end{aligned}$$

and

$$\begin{aligned}\sigma &= (u_0^{\sum_{i\in Q}\beta_i H_0(fid\|i)} \cdot \prod_{j=1}^{s} u_j^{\mu_j})^{\varphi_{\mathsf{gsk}}(\mathsf{id}_\ell,r)} \\ &= (g^{\varphi_{\mathsf{gsk}}(\mathsf{id}_\ell,r)})^{\alpha_0 \sum_{i\in Q}\beta_i H_0(fid\|i)+\sum_{j=1}^{s}\alpha_j\mu_j}\end{aligned}$$

Notice that, at least one pair of $\{(\mu'_j, \mu_j)\}_{1\le j\le s}$ should be different, since otherwise, $\sigma' = \sigma$ would also hold. Let $\Delta J \subseteq [1, s]$ be the set of all j such that $\delta_j = \mu'_j - \mu_j \neq 0$. To make σ' satisfy Equality (3) such that $\sigma' \neq \sigma$, the adversary must have known $\{\alpha_j : j \in \Delta J\}$, which contradicts the DL assumption. □

Theorem 3 *The proposed OPoS construction satisfies the ownership-hidden property against the cloud server.*

Proof On one hand, it is easy to see that, when auditing an outsourced file of some group member id_ℓ, the cloud storage server does not know its specific membership. On the other hand, the produced meta-data in the processed file looks random to the cloud server if the elements in $\mathbb{Z}$ and group $\mathbb{G}$ are both uniformly distributed. Since all α_i-es are randomly chosen from $\mathbb{Z}$, the meta-data σ_i are random elements in $\mathbb{G}$ and independent of the file owner's identity from the view of the cloud storage server. □

5 Instantiations

In this section, we first present an instantiation based on the Boneh–Boyen short signature [21]. We next show that a special type of OPoS instantiations can be optimized by employing a polynomial commitment technique [22]. The signature scheme $\mathcal{S}_t$ is the same as in Section 4. Let $H_0 : \{0,1\}^* \to \mathbb{Z}_p^*$ be a collision-resistant hash function.

5.1 A basic instantiation

$\mathsf{Setup}(1^\lambda)$: The group manager picks a bilinear pairing $\hat{e} : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$, where $\mathbb{G} = \langle g\rangle$ and $\mathbb{G}_T$ are cyclic groups with prime order p. Chooses a random value $\gamma \in_R \mathbb{Z}_p^*$ and computes $\varpi = g^\gamma$. Let $H : \{0,1\}^\rho \to \mathbb{Z}_p^*$ be a collision-resistant hash function, where ρ denotes the length of group members' identities. The group public key is $\mathsf{gpk} = (\hat{e}, \mathbb{G}, \mathbb{G}_T, g, \varpi, H, H_0)$ and the group private key is $\mathsf{gsk} = \gamma$.

$\mathsf{KeyExt}(\mathsf{gpk}, \mathsf{gsk}, \mathsf{id}_\ell)$: For each member id_ℓ $(1 \le \ell \le n)$ in the group, the manager computes a private key $\mathsf{sk}_\ell = g^{\frac{1}{\gamma+H(\mathsf{id}_\ell)}}$. After gets sk_ℓ, the member id_ℓ can validate it by checking $\hat{e}(\mathsf{sk}_\ell, \varpi \cdot g^{H(\mathsf{id}_\ell)}) \overset{?}{=} \hat{e}(g, g)$.

$\mathsf{PrFile}(\mathsf{gpk}, \mathsf{sk}_\ell, F)$: The member id_ℓ splits the file F into blocks as shown in Formula (1) on $\mathbb{Z}_p$. Chooses a random file identifier $fid \in_R \mathbb{Z}_p^*$ and $(s+1)$ random values $\alpha_0, \alpha_1, \cdots, \alpha_s \in_R \mathbb{Z}_p^*$, and computes $u_j = g^{\alpha_j} \in \mathbb{G}$ for each $0 \le j \le s$. Then, id_ℓ generates the file tag τ in the same way as in Section 4 and sends it to the group manager.

For each file block $\vec{f_i}$ $(1 \le i \le r)$, id_ℓ runs the follows steps:

- Computes $\theta_i = \alpha_0 H_0(fid\|i) + \sum_{j=1}^{s}\alpha_j f_{i,j} \in \mathbb{Z}_p$;
- Generates meta-data as $\sigma_i \leftarrow \mathsf{sk}_\ell^{\theta_i} \in \mathbb{G}$.

Then, id_ℓ sends the processed file $F^* = \{(\vec{f_i}, \sigma_i) : 1 \le i \le r\}$ to the cloud storage server, and deletes the random values $\alpha_0, \alpha_1, \cdots, \alpha_s$, private key tsk and the file locally.

$\mathsf{Chall}(\mathsf{gpk}, \tau)$: The auditor runs the following steps.

1. The same to Section 4 for validating the file tag. If τ is invalid, outputs "0" and terminates.
2. Picks a random subset $Q \subseteq [1, r]$ and chooses a random value $\beta_i \in_R \mathbb{Z}_p^*$ for each $i \in Q$. Sends the challenge $C = (fid, Q, \{\beta_i : i \in Q\})$ to the cloud storage server.

$\mathsf{PrfGen}(\mathsf{gpk}, F^*, C)$: Computes the aggregated file block $\vec{\mu} = (\mu_1, \cdots, \mu_s)$ where

$$\mu_j = \sum_{i\in Q} \beta_i f_{i,j} \mod p \text{ for each } j \in [1, s],$$

and calculates the aggregated meta-data σ as shown in Eq. 2. Sends the proof $R = (\mu, \sigma)$ to the auditor.

$\mathsf{Verify}(\mathsf{gpk}, \tau, C, R)$: If R cannot be parsed, outputs "0" and terminates. Otherwise, the auditor checks

$$\hat{e}(\sigma, \varpi \cdot g^{H(\mathsf{id}_\ell)}) \overset{?}{=} \hat{e}(u_0^{\sum_{i\in Q}\beta_i H_0(fid\|i)} \cdot \prod_{j=1}^{s} u_j^{\mu_j}, g) \quad (4)$$

If it holds, outputs "1"; otherwise, outputs "0".

Theorem 4 *The OPoS instantiation presented above is correct.*

Proof Observe the following equalities

$$\begin{aligned}&\hat{e}(\sigma, \varpi \cdot g^{H(\mathsf{id}_\ell)}) \\ &= \hat{e}(\prod_{i\in Q}(g^{\frac{1}{\gamma+H(\mathsf{id}_\ell)}})^{\theta_i\beta_i}, g^\gamma \cdot g^{H(\mathsf{id}_\ell)}) \\ &= \hat{e}(\prod_{i\in Q} g^{\beta_i(\alpha_0 H_0(fid\|i)+\sum_{j=1}^{s}\alpha_j f_{i,j})}, g) \\ &= \hat{e}(\prod_{i\in Q} g^{\beta_i\alpha_0 H_0(fid\|i)} \cdot \prod_{i\in Q}\prod_{j=1}^{s} g^{\beta_i\alpha_j f_{i,j}}, g) \\ &= \hat{e}(u_0^{\sum_{i\in Q}\beta_i H_0(fid\|i)} \cdot \prod_{j=1}^{s} u_j^{\mu_j}, g)\end{aligned}$$

Hence, the correctness follows. □

According to Theorem 2 and Theorem 3, we have the corollaries:

Corollary 1 *Suppose the signature scheme $\mathcal{S}_t$ for file tags and the Boneh–Boyen short signature scheme* [21] *are* *existentially unforgeable. The OPoS instantiation presented above is sound under the DL assumption.*

Corollary 2 *The OPoS instantiation presented above satisfies the ownership-hidden property against the cloud storage server.*

Remark 1 For ease to compare with the improved instantiation (see Section 5.2), the above OPoS instantiation is presented on symmetric bilinear groups. In fact, it can also be implemented on asymmetric bilinear groups by letting the group private key sk_ℓ, public parameters u_i $(0 \le i \le s)$ and meta-data $\{\sigma_i : 1 \le i \le r\}$ live in $\mathbb{G}_1$, while defining ϖ in $\mathbb{G}_2$.

5.2 Optimized instantiation

We proceed to show that when employing polynomial commitment technique [22], the above proposed OPoS instantiation over symmetric bilinear groups can be optimized, in this way the communication overheads for integrity auditing could be saved. Similar to [20, 24], the public parameters u_j-es are generated using a single random element α, that is, they are associated with different powers of α. When auditing an outsourced file, a polynomial commitment for the challenged blocks should be produced by the cloud storage server and then validated by the auditor.

$\mathsf{Setup}(1^\lambda)$: The same to Section 5.1.

$\mathsf{KeyExt}(\mathsf{gpk}, \mathsf{gsk}, \mathsf{id}_\ell)$: The same to Section 5.1.

$\mathsf{PrFile}(\mathsf{gpk}, \mathsf{sk}_\ell, F)$: The member id_ℓ splits the file F into blocks as shown in Formula (1) on $\mathbb{Z}_p$. Chooses a random file identifier $fid \in_R \mathbb{Z}_p^*$ and two random values $\alpha_0, \alpha \in_R \mathbb{Z}_p^*$, and computes $\upsilon = g^{\alpha_0} \in \mathbb{G}$ and $u_j = g^{\alpha^j} \in \mathbb{G}$ for each $0 \le j \le s-1$. Generates the file tag τ and sends it to the group manager in the same way as Section 4 while τ_0 denotes a concatenation string of $(\mathsf{gpk}, \mathsf{id}_\ell, \upsilon, u_0, u_1, \cdots, u_{s-1}, fid, r)$.

For each file block $\mathbf{f}_i$ $(1 \le i \le r)$, does the follows:

- Computes $\theta_i = \alpha_0 H_0(fid\|i) + \phi_{\boldsymbol{\pi}_i}(\alpha) \mod p$, where $\phi_{\boldsymbol{\pi}_i}(\alpha) = \sum_{j=0}^{s-1} f_{i,j}\alpha^j \mod p$;
- Generates meta-data as $\sigma_i \leftarrow \mathsf{sk}_\ell^{\theta_i} \in \mathbb{G}$.

Then, sends the processed file $F^* = \{(\mathbf{f}_i, \sigma_i) : 1 \le i \le r\}$ to the cloud storage server and locally discards the random values α_0, α, private key tsk and the file information.

$\mathsf{Chall}(\mathsf{gpk}, \tau)$: The auditor performs as follows.

1. The same to Section 4 for validating the file tag. If τ is invalid, outputs "0" and terminates.
2. Picks a random subset $Q \subseteq [1, r]$ and chooses two random values $z, \delta \in_R \mathbb{Z}_p^*$. Sends the challenge $C = (fid, Q, z, \delta)$ to the cloud server.

$\mathsf{PrfGen}(\mathsf{gpk}, F^*, C)$: For each $i \in Q$, the cloud storage server calculates $\beta_i = \delta^i \mod p$. Computes the aggregated file block $\vec{\mu} = (\mu_0, \mu_1, \cdots, \mu_{s-1})$ where

$$\mu_j = \sum_{i \in Q} \beta_i f_{i,j} \mod p \text{ for each } j \in [0, s-1],$$

and calculates the aggregated meta-data σ as shown in Eq. 2. Then, the server defines

$$\phi_\mu(x) = \sum_{j=0}^{s-1} \mu_j x^j \mod p$$

and calculates $\kappa = \phi_\mu(z) \mod p$. Computes the following polynomial

$$\psi_{\boldsymbol{\omega}}(x) = \frac{\phi_{\boldsymbol{\mu}}(x) - \phi_{\boldsymbol{\mu}}(z)}{x - z}$$

using polynomial long division. Let $\boldsymbol{\omega} = (\omega_0, \omega_1, \cdots, \omega_{s-2})$ be the coefficient vector of $\psi_{\boldsymbol{\omega}}(x)$. Computes

$$\zeta = g^{\psi_\omega(\alpha)} = \prod_{j=0}^{s-2} (g^{\alpha^j})^{\omega_j}$$

Sends the proof $R = (\zeta, \kappa, \sigma)$ to the auditor.

$\mathsf{Verify}(\mathsf{gpk}, \tau, C, R)$: If R cannot be parsed, outputs "0" and terminates. Otherwise, checks the equality:

$$\hat{e}(\sigma, \varpi \cdot g^{H(\mathsf{id}_\ell)}) \overset{?}{=} \hat{e}(\upsilon^{\sum_{i \in Q} \delta^i H_0(fid\|i)} \cdot g^\kappa \cdot \zeta^{-z}, g) \cdot \hat{e}(\zeta, u_1) \quad (5)$$

If it holds, outputs "1"; otherwise, outputs "0".

Theorem 5 *The above optimized OPoS instantiation is correct.*

Proof Since

$$\begin{aligned}
&\hat{e}(\sigma, \varpi \cdot g^{H(\mathsf{id}_\ell)}) \\
&= \hat{e}(\prod_{i \in Q} (g^{\frac{1}{\gamma + H(\mathsf{id}_\ell)}})^{\theta_i \beta_i}, g^\gamma \cdot g^{H(\mathsf{id}_\ell)}) \\
&= \hat{e}(\prod_{i \in Q} g^{\beta_i(\alpha_0 H_0(fid\|i) + \phi_{\boldsymbol{\pi}_i}(\alpha))}, g) \\
&= \hat{e}(\prod_{i \in Q} \upsilon^{\beta_i H_0(fid\|i)} \cdot \prod_{i \in Q} g^{\beta_i \phi_{\boldsymbol{\pi}_i}(\alpha)}, g) \\
&= \hat{e}(\upsilon^{\sum_{i \in Q} \delta^i H_0(fid\|i)}, g) \cdot \hat{e}(g^{\phi_\mu(\alpha)}, g)
\end{aligned}$$

and

$$\begin{aligned}
\hat{e}(g^\kappa \cdot \zeta^{-z}, g) \cdot \hat{e}(\zeta, u_1) &= \hat{e}(g^{(\alpha - z)\psi_\omega(\alpha) + \phi_\mu(z)}, g) \\
&= \hat{e}(g^{\phi_\mu(\alpha)}, g)
\end{aligned}$$

the verification (5) is satisfied. □

Theorem 6 *Suppose the signature scheme $\mathcal{S}_t$ for file tags and the Boneh–Boyen short signature scheme* [21] *are existentially unforgeable. The optimized OPoS instantiation is sound under the s-SDH assumption.*

Proof Suppose there is a PPT adversary $\mathcal{A}$ who can break the soundness of the optimized OPoS instantiation. Similarly to Theorem 2, at the end of the security game, the adversary outputs a forged pair of challenge/proof (C', R') for a file F' with tag τ' and member id_ℓ, where file F' has been queried for processing. If let $C' = (fid, Q, z, \delta)$ and $R' = (\zeta', \kappa', \sigma')$, then the pair (C', R') should satisfy (5). However, since it is a forged pair, R' must be unequal to that generated from the maintained information by the challenger. It further means that $(\zeta', \kappa') \neq (\zeta, \kappa)$, since otherwise, σ' would equal to σ due to Eq. 5. At this point, the simulator obtains two commitments (z, ζ', κ') and (z, ζ, κ) for the same polynomial for the evaluation at z. According to the security results [22] of polynomial commitment scheme due to Kate, Zaverucha and Goldberg, the challenger can find a solution $(-z, g^{\frac{1}{\alpha - z}})$ for the s-SDH instance $(\mathbb{G}, u_0, u_1, \cdots, u_{s-1})$, which contradicts the security assumption. □

According to Theorem 3, we have the corollary:

Corollary 3 *The above optimized OPoS instantiation satisfies the ownership-hidden property against the cloud storage server.*

5.3 Theoretical analysis

We evaluate and compare the performance of two OPoS instantiations (see Sections 5.1 – 5.2). For comparing the efficiency, we analyze their computation costs in each procedure. We only consider the most time-consuming computations, for example, exponentiation and pairing in $\mathbb{G}$, $\mathbb{G}_T$ and $\mathbb{Z}_p$, while all the other light-weight operations such as additions and multiplications are omitted. In Table 1, $\mathtt{H}$ denotes one hash evaluation for both H and H_0, and $\mathtt{E}$ represents one exponentiation in $\mathbb{G}$ or $\mathbb{Z}_p$. That is, these evaluations in different groups or ring are not discriminated. The computation time for polynomial long division and a pairing evaluation are denoted by $\mathtt{D}$ and $\mathtt{P}$, respectively. The time to randomly sample an element from a group or a ring is also omitted in the analysis, since it is in fact much less than that taken by an exponentiation. We also treat the digital signature scheme $\mathcal{S}_t$ for file tag as a black-box. Specifically, we let $O(\mathcal{S}_{kgen})$, $O(\mathcal{S}_{sign})$ and $O(\mathcal{S}_{vrfy})$ denote the computation complexity of each algorithm in $\mathcal{S}_t$, that is, $\mathcal{S}_t$.KGen, $\mathcal{S}_t$.Sign and $\mathcal{S}_t$.Vrfy.

Table 1 summarizes the computational costs of every procedure for both OPoS instantiations. Both instantiations have the same key extracting procedure to create private keys for group members. Each key extraction takes one hash evaluation and one exponentiation in $\mathbb{G}$. Regarding the file processing procedure, the secret values α^i $(2 \leq i \leq s-1)$ can be pre-computed by the group member in the optimized OPoS instantiation, which means that both instantiations take roughly the same computational complexity for processing a file. There are two ways for producing u_i in the optimized OPoS instantiation, that is, either by computing $u_i = u_{i-1}^{\alpha}$ or by raising g to the power of a pre-computed value α^i. Both approaches have the same complexity for preparing u_i. It can be seen from Table 1 that the procedure PrFile in basic OPoS instantiation requires $(r + s + 1)$ exponentiations in $\mathbb{G}$, which determines the overall efficiency for processing a file. In fact, r exponentiations are carried out for producing meta-data for r file blocks, while the other $(s+1)$ exponentiations are due to preparing public parameters $u_0, \cdots, u_s$. Consider a file F of L bytes. If it is split such that each sector has l bytes (as an element in $\mathbb{Z}_p$), then processing this file would take in total T exponentiations, where

$$T = \left\lceil \frac{L}{s \cdot l} \right\rceil + s + 1 \tag{6}$$

Similarly, processing this file by the optimized OPoS instantiation would take $T' = T - 1$ exponentiations in $\mathbb{G}$. Thus, for processing file F with either OPoS instantiation, a preferable way is to set $s = \sqrt{L/l}$ since it would cost the minimum exponentiations.

For auditing an outsourced file in the optimized instantiation, the cloud storage sever will carry out a bit more operations than the basic one. This is because that not only the coefficients β_i $(i \in Q)$ should be online calculated

Table 1 Computation costs of each procedure in both OPoS instantiations

Procedure	Basic instantiation	Optimized instantiation
Setup	$1\mathtt{E}$	$1\mathtt{E}$
KeyExt	$1\mathtt{H} + 1\mathtt{E}$	$1\mathtt{H} + 1\mathtt{E}$
PrFile	$r\mathtt{H} + (r + s + 1)\mathtt{E} + O(\mathcal{S}_{kgen}) + O(\mathcal{S}_{sign})$	$r\mathtt{H} + (r + s)\mathtt{E} + O(\mathcal{S}_{kgen}) + O(\mathcal{S}_{sign})$
Chall	$O(\mathcal{S}_{vrfy})$	$O(\mathcal{S}_{vrfy})$
PrfGen	$\lvert Q\rvert\mathtt{E}$	$(2\lvert Q\rvert + 2s - 3)\mathtt{E} + 1\mathtt{D}$
Verify	$(\lvert Q\rvert + 1)\mathtt{H} + (s + 2)\mathtt{E} + 2\mathtt{P}$	$(\lvert Q\rvert + 1)\mathtt{H} + (\lvert Q\rvert + 4)\mathtt{E} + 3\mathtt{P}$

through exponentiations, but also a polynomial evaluation for κ, polynomial long division and multi-exponentiation for ζ need to be computed. This would not degrade practicality of the instantiations as the cloud servers are usually powerful enough. The verification by the auditor depends on the parameters $|Q|$ and s. If the auditor challenges less than s file blocks, then the optimized OPoS instantiation is superior to the basic one. Note that most exponentiations taken by the auditor in the optimized instantiation are due to computing the coefficients β_i ($i \in Q$). These computations can be carried out during the period between sending out the challenge C and receiving the response R from the cloud storage server. In this way, the auditor will take only $((|Q|+1)\mathtt{H}+4\mathtt{E}+3\mathtt{P})$ operations, which is much superior to the basic instantiation.

We proceed to compare the communication overheads for both instantiations in auditing outsourced files. The details are summarized in Table 2 where S_G denotes the element size of $\mathbb{G}$. In the optimized instantiation, the coefficients β_i ($i \in Q$) are not transmitted directly across the network as in the basic instantiation, but generated by both the cloud storage server and auditor. Thus, these additional computations help to reduce the communications from the auditor to the cloud server. The similar communication reduction occurs for avoiding directly transmitting the aggregated file block from the cloud server to auditor. As shown in Section 5.2, this is realized by employing the polynomial commitment technique to commit at a random point z. It can be seen from the table that, the polynomial commitment technique brings great savings for the overall communication overheads.

5.4 Experimental analysis

We evaluate the performance of our OPoS instantiations by conducting experiments with Pairing Based Cryptography library (PBC, http://crypto.stanford.edu/pbc/). The experiments are compiled in C programming language and carried out on a system with Inter(R) Core(TM)2 Duo CPU E8500 @ 3.16GHz and 3.17GHz processor, and 4.00GB RAM. The elliptic curve is of type $y^2 = x^3 + x$ with $|p| = 160$ bits and $S_G = 512$ bits. Thus, the sector size is $l = 20$ bytes.

With algorithm KeyExt, the group manager takes roughly 4.6ms to generate a private key for a member. Thus, it can be expected that even for a company with 1000 employee, the IT department can create all the required private keys in less than 5 seconds. The simulation results are shown in Fig. 2.

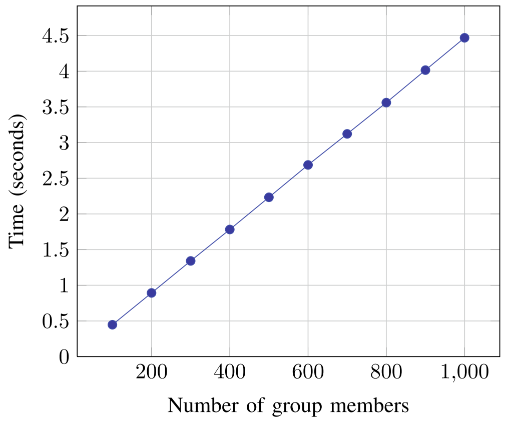

Fig. 2 Key extraction

Since producing/verifying a file tag are totally determined by the specific digital signature scheme $\mathcal{S}_t$, we omit them when evaluating the efficiency of procedure PrFile. In our experiment, for processing a given file of $L = 10\text{MB}$, we compare several cases of splitting the file, that is, $s = 100, \cdots, 600$. As discussed for Eq. 6, the minimum exponentiations happens when $s = \sqrt{10 \times 1024 \times 1024/20} \approx 3238$. In our simulation, the value of s in all cases are less than 3238. Thus, the larger of s, the more efficient for processing file F. The simulation results for two OPoS instantiations are shown in Fig. 3, which demonstrates that both instantiations take roughly the same processing time.

We proceed to simulate the integrity auditing procedures. In [3], Ateniese et al. discussed the relationship between the probability P of detecting corruption and the number $|Q|$ of challenged file blocks in C. We can deduce the same results for our OPoS scheme, that is, for an outsourced file where t percent has been (randomly) destroyed, the probability of detecting the existence of such corruption depends only on $|Q|$. Specifically, $P \approx 1-(1-t)^{|Q|}$. In our

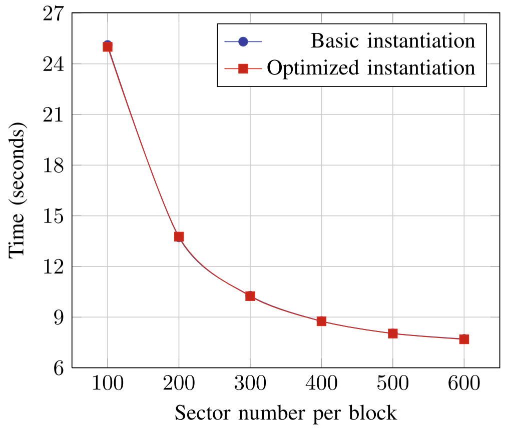

Fig. 3 Processing a 10MB file

Table 2 Communication overheads of integrity auditing for both OPoS instantiations

Instantiation	**Communication overheads**
Section 5.1	$(2\lvert Q\rvert+s)l+1S_G$
Section 5.2	$(\lvert Q\rvert+3)l+2S_G$

experiment, suppose there is a file with 1 % corruption and it has been split such that each block has s sectors ($s = 100, \cdots, 600$). For each splitting manner, we consider several cases of achieving different detecting probabilities. The simulation results for two OPoS instantiations at both sides of the cloud storage server and auditor are shown in Fig. 4, where each sub-figure is associated with a different case regarding s. Since the procedure Chall is mainly determined by a black-box component $\mathcal{S}_t$.Vrfy, it is not counted up in the experiment. It can be seen that for each splitting manner, the auditing time under both instantiations keep nearly constant in all cases with different detecting probability, although different number of file blocks should be challenged, for example, $P = 0.99$ requires $|Q| \approx 460$ and $P = 0.9$ requires $|Q| \approx 230$, etc. It also indicates that for achieving larger detecting probability about corruptions, the more computations should be carried out by the cloud storage server. Furthermore, it shows that the auditing procedure in the optimized instantiation is much more efficient than that in the basic one. This is due to that the majority exponentiations for the former case are carried out over $\mathbb{Z}_p$, which are much more efficient than those in $\mathbb{G}$ in the PBC library. However, the cloud storage server takes more computations in the optimized instantiation compared to the basic one. This is reasonable since the server is usually powerful enough.

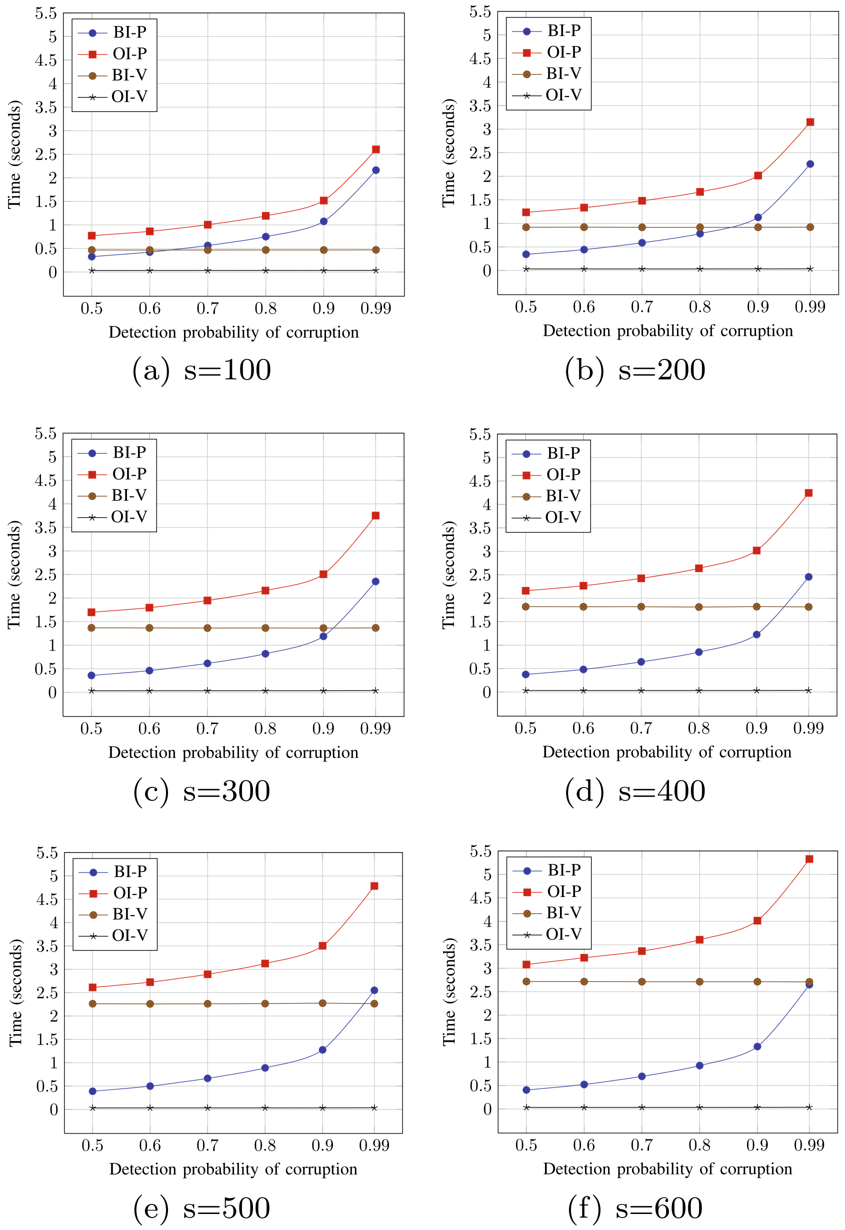

Fig. 4 Auditing a file with 1 % corruption

5.5 Discussions

In this section, we show that our OPoS instantiations can be modified to support dynamic corrupted set S_c and batch integrity auditing.

5.5.1 Dynamic corrupted set

Our OPoS instantiations presented in Sections 5.1 and 5.2 rely on a weakly secure signature scheme of Boneh and Boyen [21] to issue private keys for group members, in this way these instantiations are only secure against a static corrupted set of group members. In fact, if employing Boneh and Boyen's full signature scheme [21] to issue these private keys, the OPoS instantiations would be secure against a *dynamic* corrupted set.

$\mathsf{Setup}(1^\lambda)$: The group manager picks a bilinear pairing $\hat{e} : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$, where $\mathbb{G} = \langle g \rangle$ and $\mathbb{G}_T$ are cyclic groups with prime order p. Chooses two random values $\gamma_1, \gamma_2 \in_R \mathbb{Z}_p^*$, and computes $\varpi_1 = g^{\gamma_1}$ and $\varpi_2 = g^{\gamma_2}$. Let $H : \{0,1\}^\rho \rightarrow \mathbb{Z}_p^*$ be a collision-resistant hash function, where ρ denotes the length of group members' identities. The group public key is $\mathsf{gpk} = (\hat{e}, \mathbb{G}, \mathbb{G}_T, g, \varpi_1, \varpi_2, H, H_0)$ and the group private key is $\mathsf{gsk} = (\gamma_1, \gamma_2)$.

$\mathsf{KeyExt}(\mathsf{gpk}, \mathsf{gsk}, \mathsf{id}_\ell)$: For each member id_ℓ $(1 \leq \ell \leq n)$ in the group, the manager picks a random value $\eta \in_R \mathbb{Z}_p^* \backslash \{-\gamma_2^{-1}(\gamma_1 + H(\mathsf{id}_\ell)) \bmod p\}$ and computes a private key $\mathsf{sk}_\ell = (\mathsf{sk}_{\ell,1}, \mathsf{sk}_{\ell,2})$ where $\mathsf{sk}_{\ell,1} = g^{\frac{1}{\gamma_1 + H(\mathsf{id}_\ell) + \gamma_2 \eta}}$ and $\mathsf{sk}_{\ell,2} = \eta$. After gets sk_ℓ, the member id_ℓ can validate it by checking $\hat{e}(\mathsf{sk}_{\ell,1}, \varpi_1 \cdot g^{H(\mathsf{id}_\ell)} \cdot \varpi_2^{\mathsf{sk}_{\ell,2}}) \stackrel{?}{=} \hat{e}(g,g)$.

$\mathsf{PrFile}(\mathsf{gpk}, \mathsf{sk}_\ell, F)$: The same to Section 5.1.

$\mathsf{Chall}(\mathsf{gpk}, \tau)$: The same to Section 5.1.

$\mathsf{PrfGen}(\mathsf{gpk}, F^*, C)$: The same to Section 5.1.

$\mathsf{Verify}(\mathsf{gpk}, \tau, C, R)$: If R cannot be parsed, outputs "0" and terminates. Otherwise, the auditor checks:

$$\hat{e}(\sigma, \varpi_1 \cdot g^{H(\mathsf{id}_\ell)} \cdot \varpi_2^{\mathsf{sk}_{\ell,2}}) \stackrel{?}{=} \hat{e}(u_0^{\sum_{i \in Q} \beta_i H_0(fid \| i)} \cdot \prod_{j=1}^{s} u_j^{\mu_j}, g)$$

If so, outputs "1"; otherwise, outputs "0".

Correctness. Observe the following equalities

$$\begin{aligned}
&\hat{e}\ (\sigma, \varpi_1 \cdot g^{H(\mathsf{id}_\ell)} \cdot \varpi_2^{\mathsf{sk}_{\ell,2}})\\
&= \hat{e}(\prod_{i \in Q} (g^{\frac{1}{\gamma_1 + H(\mathsf{id}_\ell) + \gamma_2 \eta}})^{\theta_i \beta_i}, g^{\gamma_1} \cdot g^{H(\mathsf{id}_\ell)} \cdot g^{\gamma_2 \eta})\\
&= \hat{e}(\prod_{i \in Q} g^{\beta_i (\alpha_0 H_0(fid\|i) + \sum_{j=1}^{s} \alpha_j f_{i,j})}, g)\\
&= \hat{e}(u_0^{\sum_{i \in Q} \beta_i H_0(fid\|i)} \cdot \prod_{j=1}^{s} u_j^{\mu_j}, g)
\end{aligned}$$

Hence, the correctness follows.

Similarly, we continue to show that the optimized OPoS instantiation presented in Section 5.2 can also be enhanced.

$\mathsf{Setup}(1^\lambda)$: The same to the above enhanced OPoS instantiation.

$\mathsf{KeyExt}(\mathsf{gpk}, \mathsf{gsk}, \mathsf{id}_\ell)$: The same to the above enhanced OPoS instantiation.

$\mathsf{PrFile}(\mathsf{gpk}, \mathsf{sk}_\ell, F)$: The same to Section 5.2.

$\mathsf{Chall}(\mathsf{gpk}, \tau)$: The same to Section 5.2.

$\mathsf{PrfGen}(\mathsf{gpk}, F^*, C)$: The same to Section 5.2.

$\mathsf{Verify}(\mathsf{gpk}, \tau, C, R)$: If R cannot be parsed, outputs "0" and terminates. Otherwise, checks the equality:

$$\begin{aligned}
&\hat{e}(\sigma, \varpi_1 \cdot g^{H(\mathsf{id}_\ell)} \cdot \varpi_2^{\mathsf{sk}_{\ell,2}}) \stackrel{?}{=}\\
&\qquad \hat{e}(\upsilon^{\sum_{i \in Q} \delta^i H_0(fid\|i)} \cdot g^{\kappa} \cdot \zeta^{-z}, g) \cdot \hat{e}(\zeta, u_1)
\end{aligned}$$

If it holds, outputs "1"; otherwise, outputs "0".

Correctness. Observe the following equalities

$$\begin{aligned}
&\hat{e}\ (\sigma, \varpi_1 \cdot g^{H(\mathsf{id}_\ell)} \cdot \varpi_2^{\mathsf{sk}_{\ell,2}})\\
&= \hat{e}(\prod_{i \in Q} (g^{\frac{1}{\gamma_1 + H(\mathsf{id}_\ell) + \gamma_2 \eta}})^{\theta_i \beta_i}, g^{\gamma_1 + H(\mathsf{id}_\ell) + \gamma_2 \eta})\\
&= \hat{e}(\prod_{i \in Q} g^{\beta_i (\alpha_0 H_0(fid\|i) + \phi_{\vec{\pi}_i}(\alpha))}, g)\\
&= \hat{e}(\upsilon^{\sum_{i \in Q} \delta^i H_0(fid\|i)}, g) \cdot \hat{e}(g^{\phi_{\vec{\mu}}(\alpha)}, g)
\end{aligned}$$

Combining with the proof of Theorem 5, the correctness follows.

5.5.2 Batch integrity auditing

Batch integrity auditing is considered as an efficient mechanism for simultaneously auditing multiple outsourced files [32, 33]. We then show that our OPoS instantiations also support such an auditing mechanism. For ease of explanation, we assume the auditing files have the same number of blocks and sectors.

For the basic OPoS instantiation (Section 5.1), suppose there are $\mathtt{m}$ group members $\{\mathsf{id}_\ell : 1 \leq \ell \leq \mathtt{m}\}$. Each member holds a private key sk_ℓ that issued by the group manager, and has processed a file F_ℓ by running $(\tau_\ell, F_\ell^*) \leftarrow \mathsf{PrFile}(\mathsf{gpk}, \mathsf{sk}_\ell, F_\ell)$. Then, the auditor can perform the following protocol for batch auditing these $\mathtt{m}$ outsourced files by interacting with the cloud server.

Chall: The auditor runs the procedure in two steps.

1. The same to Section 4 for validating each file tag τ_ℓ for $1 \leq \ell \leq \mathtt{m}$. If any tag is invalid, outputs "0" and terminates.
2. Picks a random subset $Q \subseteq [1, r]$ and chooses a random value $\beta_i \in_R \mathbb{Z}_p^*$ for each $i \in Q$. Sends the

challenge $C = (\{fid_\ell : 1 \le \ell \le \mathtt{m}\}, Q, \{\beta_i : i \in Q\})$ to the cloud storage server.

PrfGen: Parses each processed file F_ℓ^* $(1 \le \ell \le \mathtt{m})$ as $F_\ell^* = \{\vec{f}_{\ell,i}, \sigma_{\ell,i} : 1 \le i \le r\}$ where $\vec{f}_{\ell,i} = (f_{\ell,i,1}, \cdots, f_{\ell,i,s})$. For each file F_ℓ^* $(1 \le \ell \le \mathtt{m})$, computes the aggregated file block $\vec{\mu}_\ell = (\mu_{\ell,1}, \cdots, \mu_{\ell,s})$ where

$$\mu_{\ell,j} = \sum_{i \in Q} \beta_i f_{\ell,i,j} \mod p \text{ for each } j \in [1, s].$$

Then computes

$$\sigma_\ell = \prod_{i \in Q} \sigma_{\ell,i}^{\beta_i} \in \mathbb{G}$$

Sends the proof $R = \{\vec{\mu}_\ell, \sigma_\ell : 1 \le \ell \le \mathtt{m}\}$ to the auditor.
Vrfy: If R cannot be parsed, outputs "0" and terminates. Otherwise, the auditor checks

$$\prod_{\ell=1}^{\mathtt{m}} \hat{e}(\sigma_\ell, \varpi \cdot g^{H(\mathsf{id}_\ell)}) \overset{?}{=} \hat{e}(\prod_{\ell=1}^{\mathtt{m}} (u_{\ell,0}^{\sum_{i \in Q} \beta_i H_0(fid_\ell \| i)} \cdot \prod_{j=1}^{s} u_{\ell,j}^{\mu_{\ell,j}}), g)$$

If so, outputs "1"; otherwise, outputs "0".

It can be seen that with batch auditing mechanism, $(\mathtt{m} - 1)$ pairings are saved at the auditor side compared with the separate integrity auditing case.

Correctness. According to Eq. 4, if all outsourced files F_ℓ^* $(1 \le \ell \le \mathtt{m})$ are keeping intact by the cloud server, then we have

$$\begin{aligned}
&\prod_{\ell=1}^{\mathtt{m}} \hat{e}(\sigma_\ell, \varpi \cdot g^{H(\mathsf{id}_\ell)}) \\
&= \prod_{\ell=1}^{\mathtt{m}} \hat{e}(u_{\ell,0}^{\sum_{i \in Q} \beta_i H_0(fid_\ell \| i)} \cdot \prod_{j=1}^{s} u_{\ell,j}^{\mu_{\ell,j}}, g) \\
&= \hat{e}(\prod_{\ell=1}^{\mathtt{m}} (u_{\ell,0}^{\sum_{i \in Q} \beta_i H_0(fid_\ell \| i)} \cdot \prod_{j=1}^{s} u_{\ell,j}^{\mu_{\ell,j}}), g)
\end{aligned}$$

That means the correctness for batch integrity verification is satisfied.

The optimized OPoS instantiation (Section 5.2) also supports batch integrity auditing for multiple outsourced files. Similarly, suppose each group member id_ℓ $(1 \le \ell \le \mathtt{m})$ holds a private key sk_ℓ that issued by the group manager, and has processed a file F_ℓ by running $(\tau_\ell, F_\ell^*) \leftarrow \mathsf{PrFile}(\mathsf{gpk}, \mathsf{sk}_\ell, F_\ell)$. Then, the auditor can audit these files in a batch by performing the follows steps by interacting with the cloud storage server.

Chall: The auditor runs the procedure as follows.

1. The same to Section 4 for validating each file tag τ_ℓ for $1 \le \ell \le \mathtt{m}$. If any tag is invalid, outputs "0" and terminates.
2. Picks a random subset $Q \subseteq [1, r]$ and chooses two random values $z, \delta \in_R \mathbb{Z}_p^*$. Sends the challenge $C = (\{fid_\ell : 1 \le \ell \le \mathtt{m}\}, Q, z, \delta)$ to the cloud server.

PrfGen: For each $i \in Q$, the cloud storage server calculates $\beta_i = \delta^i \mod p$. Then, for each processed file $F_\ell^* = \{\vec{f}_{\ell,i}, \sigma_{\ell,i} : 1 \le i \le r\}$ $(1 \le \ell \le \mathtt{m})$, the cloud server generates $R_\ell = (\zeta_\ell, \kappa_\ell, \sigma_\ell)$ in the same way as the procedure PrfGen in Section 5.2. Moreover, computes $\kappa = \sum_{\ell=1}^{\mathtt{m}} \kappa_\ell \mod p$. Sends the proof $R = (\{\zeta_\ell, \sigma_\ell : 1 \le \ell \le \mathtt{m}\}, \kappa)$ to the auditor.
Vrfy: If R cannot be parsed, outputs "0" and terminates. Otherwise, checks the equality:

$$\prod_{\ell=1}^{\mathtt{m}} \hat{e}(\sigma_\ell, \varpi \cdot g^{H(\mathsf{id}_\ell)}) \overset{?}{=} \prod_{\ell=1}^{\mathtt{m}} \hat{e}(\zeta_\ell, u_{\ell,1}) \cdot \hat{e}((\prod_{\ell=1}^{\mathtt{m}} \upsilon_\ell^{\sum_{i \in Q} \delta^i H_0(fid_\ell \| i)}) \cdot g^{\kappa} \cdot (\prod_{\ell=1}^{\mathtt{m}} \zeta_\ell)^{-z}, g)$$

If it holds, outputs "1"; otherwise, outputs "0".

Notice that with batch auditing mechanism, $(\mathtt{m} - 1)$ pairings and $(2\mathtt{m} - 2)$ exponentiations are saved at the auditor side compared to the separate auditing case.

Correctness. According to Eq. 5, if all outsourced files F_ℓ^* $(1 \le \ell \le \mathtt{m})$ are keeping intact by the cloud server, then we have

$$\begin{aligned}
&\prod_{\ell=1}^{\mathtt{m}} \hat{e}(\sigma_\ell, \varpi \cdot g^{H(\mathsf{id}_\ell)}) \\
&= \prod_{\ell=1}^{\mathtt{m}} (\hat{e}(\upsilon_\ell^{\sum_{i \in Q} \delta^i H_0(fid_\ell \| i)} \cdot g^{\kappa_\ell} \cdot \zeta_\ell^{-z}, g) \cdot \hat{e}(\zeta_\ell, u_{\ell,1})) \\
&= \hat{e}((\prod_{\ell=1}^{\mathtt{m}} \upsilon_\ell^{\sum_{i \in Q} \delta^i H_0(fid_\ell \| i)}) \cdot g^{\sum_{\ell=1}^{\mathtt{m}} \kappa_\ell} \cdot (\prod_{\ell=1}^{\mathtt{m}} \zeta_\ell^{-z}), g) \\
&\quad \cdot \prod_{\ell=1}^{\mathtt{m}} \hat{e}(\zeta_\ell, u_{\ell,1}) \\
&= \hat{e}((\prod_{\ell=1}^{\mathtt{m}} \upsilon_\ell^{\sum_{i \in Q} \delta^i H_0(fid_\ell \| i)}) \cdot g^{\kappa} \cdot (\prod_{\ell=1}^{\mathtt{m}} \zeta_\ell)^{-z}, g) \\
&\quad \cdot \prod_{\ell=1}^{\mathtt{m}} \hat{e}(\zeta_\ell, u_{\ell,1})
\end{aligned}$$

Thus, the correctness holds for batch integrity verification for the optimized OPoS instantiation.

6 Conclusion

In this paper, we introduced OPoS which guarantees the integrity of outsourced files in clouds for group-oriented applications. We first formalized the system framework and security model for OPoS schemes, and then proposed a generic OPoS construction from pre-homomorphic signatures using the trapdoor technique. Following the generic construction, we presented two instantiations on bilinear groups using Boneh–Boyen short signature, of them the second one is optimized from the first one using a polynomial commitment technique for saving communication overheads in auditing outsourced files. We provided security proof and comprehensive performance evaluations, which show that our instantiations are secure and practical for real-world applications. Moreover, we discussed that our instantiations can be strengthened for withstanding a dynamic colluding set of members, and support batch integrity auditing for improving auditing efficiency.

Acknowledgments This work was partially supported by the Natural Science Foundation of China (Nos. 61672083, 61672083, 61370190, 61272501, 61202465, 61402029, 61472429, 61202465, 61532021), by the Beijing Natural Science Foundation (No. 4132056) and by the Guangxi natural science foundation (2013 GXNSFBB053005).

References

1. Su Z, Xu Q, Qi Q (2016) Big data in mobile social networks: a QoE-oriented framework. IEEE Netw 30(1):52–57
2. Deng H, Wu Q, Qin B, Chow SSM, Domingo-Ferrer J, Shi W (2014) Tracing and revoking leaked credentials: Accountability in leaking sensitive outsourced data. In: Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security. ACM, pp 425–434
3. Ateniese G, Burns R, Curtmola R, Herring J, Kissner L, Peterson Z, Song D (2007) Provable data possession at untrusted stores Proceedings of the 14th ACM conference on computer and communications security. ACM, pp 598–609
4. Ateniese G, Kamara S, Katz J (2009) Proofs of storage from homomorphic identification protocols. In: Matsui M (ed) Advances in cryptology–ASIACRYPT 2009, vol 5912. Springer, Heidelberg, pp 319-333
5. Deng H, Wu Q, Qin B, Mao J, Liu X, Zhang L, Shi W (2014) Who is touching my cloud. In: Kutylowski M, Vaidya J (eds) Computer Security–ESORICS 2014, vol 8712. Springer International Publishing, pp 362–379
6. Xiong J, Li F, Ma J, Liu X, Yao Z, Chen PS (2015) A full lifecycle privacy protection scheme for sensitive data in cloud computing. Peer-to-Peer Networking and Applications 8(6):1025–1037
7. Chen X, Li J, Huang X, Ma J, Lou W (2015) New publicly verifiable databases with efficient updates. IEEE Transactions on Dependable and Secure Computing 12(5):546–556
8. Chen X, Li J, Weng J, Ma J, Lou W (2016) Verifiable computation over large database with incremental updates. IEEE Trans Comput
9. Wen M, Lu K, Lei J, Li F, Li J (2015) BDO-SD: An efficient scheme for big data outsourcing with secure deduplication. In: 2015 IEEE Conference on computer communications workshops (INFOCOM WKSHPS). IEEE, pp 214–219
10. Wen M, Ota K, Li H, Lei J, Gu C, Su Z (2015) Secure data deduplication with reliable key management for dynamic updates in cpss. IEEE Transactions on Computational Social Systems 2(4):137–147
11. Yu CM, Chen CY, Chao HC (2015) Proof of ownership in deduplicated cloud storage with mobile device efficiency. IEEE Netw 29(2):51–55
12. Huang X, Liu JK, Tang S, Xiang Y, Liang K, Xu L, Zhou J (2015) Cost-effective authentic and anonymous data sharing with forward security. IEEE Trans Comput 64(4):971–983
13. Huang X, Xiang Y, Bertino E, Zhou J, Xu L (2014) Robust multi-factor authentication for fragile communications. IEEE Transactions on Dependable and Secure Computing 11(6): 568–581
14. Wu Q, Qin B, Zhang L, Domingo-Ferrer J, Farràs O, Manjón JA (2016) Contributory broadcast encryption with efficient encryption and short ciphertexts. IEEE Trans Comput 65(2):466–479
15. Wu Q, Qin B, Zhang L, Domingo-Ferrer J, Manjón JA (2013) Fast transmission to remote cooperative groups: a new key management paradigm. IEEE/ACM Trans Networking 21(2):621–633
16. Juels A, Kaliski BS Jr (2007) PORs: Proofs of retrievability for large files. In: Proceedings of the 14th ACM Conference on Computer and Communications Security. ACM, pp 584–597
17. Ateniese G, Di Pietro R, Mancini LV, Tsudik G (2008) Scalable and efficient provable data possession. In: Proceedings of the 4th International Conference on Security and Privacy in Communication Networks. ACM
18. Erway C, Küpçü A, Papamanthou C, Tamassia R (2009) Dynamic provable data possession. In: Proceedings of the 16th ACM Conference on Computer and Communications Security. ACM, pp 213–222
19. Shacham H, Waters B (2013) Compact proofs of retrievability. J Cryptol 26(3):442–483
20. Yuan J, Yu S (2015) PCPOR: Public and constant-cost proofs of retrievability in cloud. J Comput Secur 23(3):403–425
21. Boneh D, Boyen X (2008) Short signatures without random oracles and the sdh assumption in bilinear groups. J Cryptol 21(2):149–177
22. Kate A, Zaverucha GM, Goldberg I (2010) Constant-size commitments to polynomials and their applications. In: Abe M (ed) Advances in cryptology–ASIACRYPT 2010, vol 6477. Springer, Heidelberg, pp 177-194
23. Wang Y, Wu Q, Qin B, Chen X, Huang X, Zhou Y (2015) Group-oriented proofs of storage. In: Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security. ACM, pp 73–84
24. Xu J, Chang EC (2012) Towards efficient proofs of retrievability. In: Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security. ACM, pp 79–80
25. Wang Y, Wu Q, Wong DS, Qin B, Chow SSM, Liu Z, Tan X (2014) Securely outsourcing exponentiations with single untrusted program for cloud storage. In: Kutylowski M, Vaidya J (eds) Computer Security–ESORICS 2014, vol 8712. Springer International Publishing, pp 326–343
26. Wang B, Li B, Li H (2012) Knox: Privacy-preserving auditing for shared data with large groups in the cloud. In: Bao F, Samarati P, Zhou J (eds) Applied cryptography and network security, vol 7341. Springer, Heidelberg, pp 507-525
27. Wang B, Li B, Li H (2012) Oruta: Privacy-preserving public auditing for shared data in the cloud. In: 2012 IEEE 5th international conference on Cloud computing (CLOUD), pp 295–302

28. Wang B, Chow SSM, Li M, Li H (2013) Storing shared data on the cloud via security-mediator. In: 2013 IEEE 33rd international conference on Distributed computing systems (ICDCS), pp 124–133
29. Wang H, Wu Q, Qin B, Domingo-Ferrer J (2014) Identity-based remote data possession checking in public clouds. IET Inf Secur 8(2):114–121
30. Yu Y, Mu Y, Ni J, Deng J, Huang K (2014) Identity privacy-preserving public auditing with dynamic group for secure mobile cloud storage. In: Au MH, Carminati B, Kuo CCJ (eds) Network and System Security, vol 8792. Springer International Publishing, pp 28–40
31. Freeman DM (2012) Improved security for linearly homomorphic signatures: a generic framework. In: Fischlin M, Buchmann J, Manulis M (eds) Public key cryptography–PKC 2012, vol 7293. Springer, Heidelberg, pp 697-714
32. Wang C, Chow SSM, Wang Q, Ren K, Lou W (2013) Privacy-preserving public auditing for secure cloud storage. IEEE Trans Comput 62(2):362–375
33. Ren Y, Shen J, Zheng Y, Wang J, Chao HC (2015) Efficient data integrity auditing for storage security in mobile health cloud. Peer-to-Peer Networking and Applications:1–10