This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2019.2925800, IEEE Transactions on Dependable and Secure Computing

Dynamic Group-Oriented Provable Data Possession in the Cloud

Kun He, Jing Chen, Quan Yuan, Shouling Ji, Debiao He, and Ruiying Du

***Abstract*—As an important security property of cloud storage, data integrity has not been sufficiently studied under the multi-writer model, where a group of users work on shared files collaboratively and any group member can update the data by modification, insertion, and deletion operations. Existing works under such multi-writer model would bring large storage cost to the third-party verifiers. Furthermore, to the best of our knowledge, none of the existing works for shared files supports fully dynamic operations, which implies that users cannot freely perform the update operations.**

In this paper, we propose the first public auditing scheme for shared data that supports fully dynamic operations and achieves constant storage cost for the verifiers. Our scheme, named PRAYS, is boosted by a new paradigm for remote data integrity checking. To implement the new paradigm, we proposed a specially designed authenticated structure, called *blockless Merkle tree*, and a novel cryptographic primitive, called *permission-based signature*. Extensive evaluation demonstrates that PRAYS is as efficient as the existing less-functional solutions. We believe that PRAYS is an important step towards designing practical multi-writer cloud storage systems.

***Index Terms*—provable data possession, blockless Merkle tree, permission-based signature**

I. Introduction

Cloud storage, which provides ubiquitous access to a pool of configurable remote storage resources on-demand, is an attractive paradigm to both individuals and enterprises. Along with this convenience, *data integrity* becomes a major concern about storage outsourcing, especially considering platform failures and human errors [1]–[3].

To guarantee data integrity in cloud storage services, many relevant cryptographic primitives have been proposed [4]–[8]. Generally, through assigning a cryptographic tag to each data block of a file and validating it, those primitives allow a verifier (i.e., the data owner or a special third party) to examine remote data integrity without downloading the whole file, and therefore reduce the communication cost. However, those primitives are limited to the *single-writer* model, where only the data owner can update the data in the cloud.

On the other hand, as online cooperation develops intensively, the *multi-writer* model, where shared files could be updated by a group of users for collaboration, is more preferred in nowadays cloud platforms (e.g., Dropbox and SugarSync). Protecting data integrity in the multi-writer cloud storage, i.e., for dynamic shared data, then turns to be an urgent challenge.

Most existing solutions under the multi-writer model simply apply the paradigm for the single-writer model, under which each data block is signed with a user's private key. When a user is revoked, all the data blocks signed by that user have to be re-signed by an unrevoked user or the cloud server [9]–[11]. Since the number of data blocks is huge in the cloud (e.g., 1 TB data can have 2.68×10^8 data blocks with each block of size 4 kB), these kinds of methods are inefficient in practice.

Some researchers regarded signers' identities as private under the multi-writer model, since they could reveal some significant information about the signed (even encrypted) data. Taking the e-Health records outsouring as an example, once the cloud finds that a patient's (maybe encrypted) record is signed by an oncologist, the cloud could infer some private information about that patient, which may violate patient rights. Many privacy-preserving solutions have been proposed to solve this issue [12]–[14]. However, cooperative users in those solutions cannot determine *by themselves* who updated the files stored in the cloud, which is an important function in real-world cloud storage systems (e.g., Dropbox and SugarSync), called *revision history*. This means that in the aforementioned example, a doctor is unable to learn who made the previous diagnosis by himself/herself. In summary, a privacy-preserving integrity checking scheme under the multi-writer model should achieve *anonymity* and *offline traceability*, simultaneously.

In addition, there are two other shortcomings in existing multi-writer solutions. The size of verification materials in those solutions, such as public keys, depends on the number of users or data blocks, which may result in unaffordable workload, especially when the data is huge [9]–[14]. That means the verification process only applies to dedicated servers and not to users' resource-constrained devices, such as smartphones and laptops. On the other hand, those solutions do not support *fully* dynamic operations, which includes unlimited times of modifications, insertions, and deletions of data blocks. Specifically, some schemes only supports modifications and deletions, but not insertions [10], [14].

Based on the above discussions, there still lacks an efficient

K. He and D. He are with Key Laboratory of Aerospace Information Security and Trusted Computing, Ministry of Education, School of Cyber Science and Engineering, Wuhan University, Wuhan, China.

J. Chen is with Key Laboratory of Aerospace Information Security and Trusted Computing, Ministry of Education, School of Cyber Science and Engineering, Wuhan University, Wuhan, with Shenzhen Institute of Wuhan University, Shenzhen, and with Science and Technology on Communication Security Laboratory, Chengdu, China.

Q. Yuan is with the Computer School, University of Texas-Permian Basin, TX, USA.

S. Ji is with the Computer School, Zhejiang University, Hangzhou, China.

R. Du is with Key Laboratory of Aerospace Information Security and Trusted Computing, Ministry of Education, School of Cyber Science and Engineering, Wuhan University, and with Collaborative Innovation Center of Geospatial Technology, Wuhan, China.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2019.2925800, IEEE Transactions on Dependable and Secure Computing

and privacy-preserving integrity checking scheme under the multi-writer model, in which a group of users are enabled to outsource and work on (i.e., read and write) shared files collaboratively. In practice, a preferred data integrity checking design is expected to have the following features (in addition to integrity):

- *Fully dynamic operations.* This property implies that group members can freely perform modification, insertion, and deletion operations.
- *Constant auditing metadata.* This property implies that the size of verification materials maintained by verifiers for integrity checking should be independent of the number of users and the data size.
- *Secure user revocation.* This property implies that group members can be efficiently revoked. Further, the system should resist the collusion between revoked users and the cloud, and between revoked users and third party verifiers.
- *Anonymity.* This property implies that a writer's identity should not be revealed from his/her signature to the cloud or third party verifiers.
- *Traceability.* This property implies that users from the same group can identify who updated the shared data from the signature, i.e., obtaining the revision history, without the help from any online entity.

In this paper, we follow the line of provable data possession [4], [15], and propose a dynamic group-oriented provable data possession scheme, called PRAYS, which holds all the above-mentioned features. Compared with existing solutions (i.e., generating the tags and then building the structure), PRAYS is boosted by a new paradigm: building the structure and then generating the tag. Our main contributions are summarized as follows.

1) We present a customized authenticated structure, named *blockless Merkle tree*. Compared with the traditional Merkle tree, the proposed structure supports blockless verification (i.e., to check remote data integrity without downloading the challenged data blocks) through an elaborate process for each data block.
2) We propose a novel cryptographic primitive, named *permission-based signature*. Permission-based signature is the first cryptographic primitive that achieves both anonymity and offline traceability. Further, this primitive could also be used independently in other privacy-preserving applications.
3) We design PRAYS based on the blockless Merkle tree and the permission-based signature. To the best of our knowledge, PRAYS is the first provable data possession scheme under the multi-writer model that supports fully dynamic operations as well as constant auditing metadata.
4) We conduct comprehensive security analysis and extensive evaluations for the proposed scheme. The results demonstrate that, compared with existing solutions, PRAYS can perform richer functions (e.g., fully dynamic operations) while maintaining reasonable computation and communication cost.

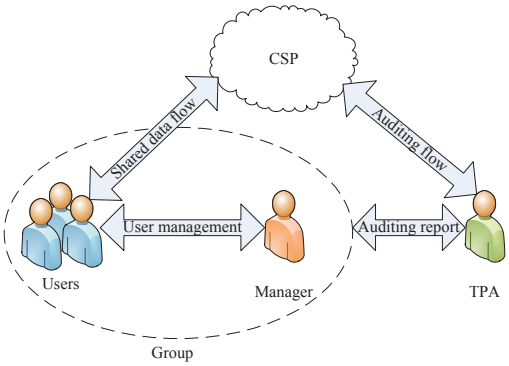


Fig. 1. The system model of multi-writer cloud storage.

The rest of this paper is organized as follows. In Section II, we describe the models and definitions. We present our solution, called PRAYS, in Section III. We conduct the security analysis and performance evaluation in Section IV and Section V, respectively. We review the related work in Section VI, and we conclude this paper and point out our future work in Section VII.

II. Models and Definitions

A. System Model

Our setting of interest focuses on the deployment of cloud storage systems under the multi-writer model, which consists of a Cloud Service Provider (CSP), a Third-Party Auditor (TPA), a group of users, and a manager, as shown in Fig. 1.

CSP offers storage service and TPA provides data auditing service. The manager creates a group of users and manages all the users in the group, i.e., user registration and user revocation. Note that the manager only acts for user management. Thus, he/she does not have to be online all the time. Users within a group have equal status, which means that every user can upload a new file to the cloud, and then other users can read and update that file. This model has been adopted in many other solutions [11], [12].

State Information Synchronization. Most existing integrity checking schemes for dynamic data (under both the single-writer and multi-writer models) are stateful [10], [16]; otherwise, CSP can use outdated data to pass the verification. Under the multi-writer model, the situation is more complex. That is, the latest state information has to be synchronized among the TPA and all the users. We highlight that state information synchronization is important while orthogonal to our work. Existing schemes (e.g., [10]) solve this problem by simply sending the state information to TPA every time when the data is updated. Then, users can obtain the latest state information from TPA. However, TPA may be offline when the data is updated. Therefore, in this paper, we employ an alternative solution that depends on a trusted public platform (which is not included in Fig. 1), such as bulletin boards or blockchains [17]–[19], for publishing and obtaining the latest state information.

Single-writer model vs. multi-writer model. In the single-writer model, although the data owner (i.e., the one who

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2019.2925800, IEEE Transactions on Dependable and Secure Computing

uploaded the data) can share his/her data with other users in the cloud, these users only have read permission. That is, only the data owner can update his/her data in the cloud. In the multi-writer model, a group of users are enabled to outsource and work on shared data collaboratively. Specifically, a shared data block stored in the cloud could be constantly read, modified, inserted, and deleted by any group member.

B. Threat Model

We assume that the manager and (unrevoked) users are honest as in [10], [14]. We consider a threat model under which CSP and TPA are honest-but-curious [20], [21].

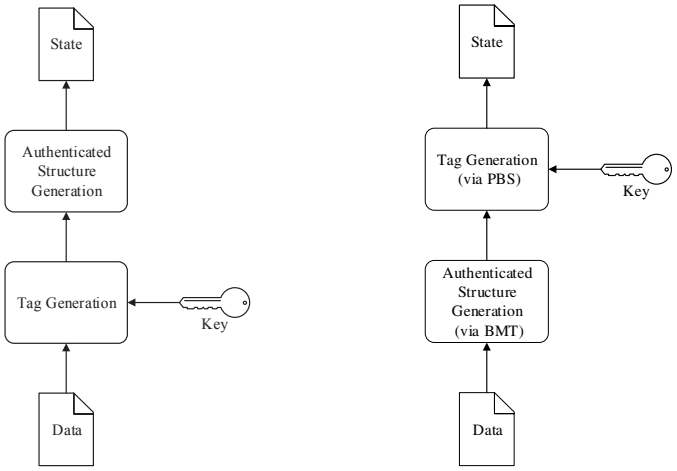
- CSP may delete a part of the stored data for saving storage cost and try to cheat users that all the data is stored faithfully. CSP may be curious about the identity of the uploader and updater [22], [23], which implies that CSP tries to extract identities from the stored data.
- TPA may also be curious about the identities of users and may try to extract identities in the checking process.
- Both CSP or TPA may collude with revoked users.

C. Definitions

We here present the syntax of dynamic group-oriented provable data possession. The security definitions are discussed in Section IV.

Definition 1. *A dynamic group-oriented provable data possession scheme consists of the following seven phases.*

- *Initialization Phase. This phase is launched by the manager to initialize the system, which only appears once during the entire life cycle of the system. With the input of a security parameter λ, the manager obtains a public-private key pair (pk, sk).*
- *Registration Phase. This phase is launched by a user to obtain a user key, which only occurs when a user registers to the system. With the input of an identity id, the user receives a user key uk_{id} from the manager.*
- *Revocation Phase. This phase is launched by the manager to disable a user key, which occurs when a user is corrupted or leaves the system. With the input of an identity id and the public-private key pair (pk, sk), the manager updates the public-private key pair and all the unrevoked user keys.*
- *Uploading Phase. This phase is launched by a user to upload a new file to the cloud. With the input of a file $\{d_i\}_{i=1}^{D}$ that consists of D data blocks and a user key uk_{id}, the user obtains the authenticated structure τ and state information st of the file.*
- *Reading Phase. This phase is launched by a user to obtain a data block. With the input of a block index i, the user key uk_{id}, and the state information st, the user obtains the data block d_i and the signer identity.*
- *Writing Phase. This phase is launched by a user to update an existing file. With the input of an update (i, d_i, op) which consists of a block index i, a data block d_i, and an operation op, and the user key uk_{id}, the user updates the file, authenticated structure, and state information.*



(a) Traditional approaches (b) Our approach

Fig. 2. Overview of the workflows.

- *Auditing Phase. This phase is launched by TPA to decide whether data is faithfully stored on the cloud. With the input of the public key pk and the state information st, TPA outputs a decision with a value 0 or 1.*

The correctness of dynamic group-oriented provable data possession is straightforward. On one hand, TPA always outputs an acceptance in the auditing phase if CSP is honest. On the other hand, a user always outputs the correct signer identity in the reading phase if the authenticated structure and state information are generated by some honest user.

III. Our Construction

In this section, we present a PRivacy-preserving Auditing scheme for dYnamic Shared data, named PRAYS. First, we give an overview of the design and the challenges in it. Then, we propose a customized authenticated structure and a novel cryptographic primitive, respectively. Finally, we describe the details of PRAYS.

A. Overview

Unlike most existing solutions under the single-writer or multi-writer models (cf. Fig. 2(a)), data blocks in our scheme are not signed with a user's private key. Instead, we design a new paradigm for remote data integrity checking as shown in Fig. 2(b). First, we design a customized authenticated structure, called *Blockless Merkle Tree* (BMT), and build this structure directly from the data blocks without involving any private keys. In our design, a TPA that holds the "correct" root of the blockless Merkle tree can verify whether the challenged data blocks have been tampered or are out of date. Then, to ensure that the root is generated by a legal user (i.e., to achieve secure user revocation) and to provide both anonymity and traceability, the root of the tree is signed with a user's private key via a novel cryptographic primitive, called *Permission-Based Signature* (PBS). The main notations used in our construction are summarized in Table I.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2019.2925800, IEEE Transactions on Dependable and Secure Computing

TABLE I
MAIN NOTATIONS USED IN PRAYS

Notation	Description
D	The number of data blocks
S	The number of segments in a data block
B	The number of challenged data blocks
$d_i, d_{i,j}$	The ith data block and its jth segment
ν_ι, ν_1	The ιth node and the root of BMT
ι_i	The index in BMT of the ith leaf node
(pk, sk)	The public-private key pair
$uk_{id}, cert_{id}$	The user key and the corresponding certificate of identity id

B. Design Challenge

In order to achieve blockless verification, existing solutions (e.g. [24], [25]) employ homomorphic authenticators [6], [26]. Then, constructing an authenticated structure, such as Merkle tree, from homomorphic tags yields a solution with both blockless verification and fully dynamic operations (cf. the paradigm in Fig. 2(a)). However, this approach is not suitable for the multi-writer model since every block signed by a revoked user should be re-signed in the revocation process, as indicated in [12] and [10]. Therefore, the computation cost is extremely high in the current paradigm if the number of data blocks is huge. The first challenge in our design is to construct an authenticated structure that supports blockless verification and fully dynamic operations without any private keys.

The second challenge is to achieve anonymity and offline traceability simultaneously as explained in Section I. We note that group signatures can solve the anonymity issue [27], [28]. However, due to the inherent property of group signature, it cannot solve the traceability problem, unless users keep contacting with the manager when they read data from the cloud, which requires extra communication overhead. Further, this requires the manager to be online all the time, which is a security bottleneck of the system. One may argue that this issue can be solved by giving each user the opening key of group signature. Unfortunately, this trivial solution (and other solutions that depend on a shared secret key) has two unacceptable shortcomings. First, it is impossible to trace the traitors when the opening key is leaked since all users possess the same one. Second, secure channels need to be established between the manager and unrevoked users for redistributing the opening key in the revocation process.

C. Blockless Merkle Tree

To tackle the first challenge in Section III-B and realize remote data integrity checking, we propose *Blockless Merkle Tree* (BMT) as a new building block. Compared with existing usages of Merkle tree which combine with homomorphic authenticators, BMT is designed for *blockless verification* (where TPA does not have to download all the challenged blocks) without any homomorphic authenticator, which makes BMT valuable for remote data integrity checking (even under the single-writer model). Since the tree building process requires no private key, BMT is especially suitable for the multi-writer model, i.e., dynamic group-oriented provable data possessions. In addition, the tree itself binds each data block and its position[1], therefore, the proof can also provide the position correctness. In summary, remote data integrity can be guaranteed with blockless verification through the proposed blockless Merkle tree if the verifier possesses the correct root.

1) Syntax: We here formally introduce the syntax of blockless Merkle tree. Note that the major difference between prior usages and our proposal is that the Merkle tree in previous approaches is built from the tags rather than the file. As a result, the proving algorithm and proof verification algorithm in BMT are very different from the traditional ones.

Definition 2. *A blockless Merkle tree scheme is a 4-tuple* $(\mathsf{Build}, \mathsf{Prove}, \mathsf{Verify}, \mathsf{Update})$.

- *The tree building algorithm* $\mathsf{Build}()$ *takes as input a file, and outputs an authenticated structure and a metadata of the authenticated structure.*
- *The proving algorithm* $\mathsf{Prove}()$ *takes as input a challenge, a file, and an authenticated structure, and outputs a proof. The proof is blockless if it does not contain every challenged file blocks.*
- *The proof verification algorithm* $\mathsf{Verify}()$ *takes as input a challenge, a proof, and a metadata, and outputs a decision about whether the proof is valid.*
- *The tree update algorithm* $\mathsf{Update}()$ *takes as input an update, a file, an authenticated structure, and a metadata, and outputs the updated file, authenticated structure, and metadata.*

The correctness of blockless Merkle tree is straightforward. Intuitively, it means that the honestly generated proof should always be valid.

2) Security Definition: The expected security property of blockless Merkle tree is *integrity*, which is defined by the following game between a challenger and an adversary.

1) The adversary chooses a file according to some distribution and sends it to the challenger.
2) The challenger runs $\mathsf{BMT.Build}$ and sends the authenticated structure to the adversary.
3) The adversary may ask the challenger to run $\mathsf{BMT.Update}$ with adversary-specified updates for polynomial times.
4) The challenger sends a challenge to the adversary, and receives a proof from the adversary. We say that the adversary wins if the challenger accepts the proof, i.e., the proof verification algorithm $\mathsf{BMT.Verify}$ return 1.

Definition 3 (Integrity)**.** *A blockless Merkle tree scheme guarantees the integrity if for any probabilistic polynomial time adversary that wins the game, the challenger can reconstruct the challenged blocks in polynomial time.*

3) Construction: We now detail a concrete construction of blockless Merkle tree. Let $\mathbb{G}$ be a multiplicative cyclic group of prime order p, and $H : \{0,1\}^* \rightarrow \{0,1\}^*$ be a cryptographic hash function. We assume that the file consists of D data blocks $\{d_i\}_{i=1}^{D}$ and a data block is the basic unit (e.g., 4kB and 16kB) when CSP stores and processes the file. However, a data block is too large to be handled in $\mathbb{Z}_p$. Thus,

[1]This trick has been used in some literatures [25], [29], [30].

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2019.2925800, IEEE Transactions on Dependable and Secure Computing

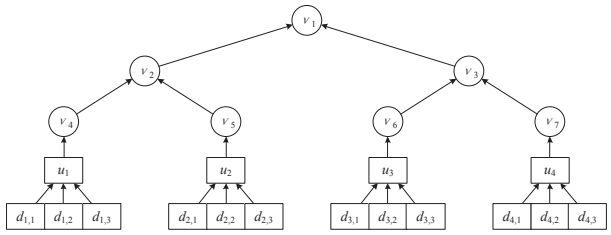


Fig. 3. An illustration of tree building.

we divide each data block d_i into S segments $\{d_{i,j}\}_{j=1}^{S}$ where $d_{i,j} \in \mathbb{Z}_p$. The four Probabilistic Polynomial Time (PPT) algorithms are described below.

The Tree Building Algorithm BMT.Build()**.** With the input of D data blocks $\{d_i\}_{i=1}^{D}$, this algorithm outputs an authenticated structure and some metadata. The detailed procedure is as follows.

1) Choose S random generators $g_1, \ldots, g_S \in \mathbb{G}$.
2) For each data block d_i, compute $u_i := \prod_{j=1}^{S} g_j^{d_{i,j}}$.
3) Build a complete binary tree τ with D leaf nodes as shown in Fig. 3, in which each node stores a triple $\nu_\iota = (\iota, l_\iota, s_\iota)$, where ι is the unique index of the node in the tree, l_ι is the number of leaf nodes that can be reached from the ιth node, and s_ι is a hash value. The index of root is 1, and the index increases from top to bottom and from left to right. We explain how to assign l_ι and s_ι in the following steps.
4) For the ith leaf node whose index in the tree is ι_i, set $l_{\iota_i} := 1$ and compute $s_{\iota_i} := H(u_i)$.
5) For each non-leaf node whose index in the tree is ι, compute $l_\iota := l_{2\iota} + l_{2\iota+1}$ and $s_\iota := H(\nu_{2\iota} \| \nu_{2\iota+1})$, where $\nu_{2\iota} = (2\iota, l_{2\iota}, s_{2\iota})$ and $\nu_{2\iota+1} = (2\iota+1, l_{2\iota+1}, s_{2\iota+1})$ are ν_ι's left child and right child, respectively.
6) Return the authenticated structure $(\tau, \{g_j\}_{j=1}^{S})$ and the metadata $(\nu_1, \{g_j\}_{j=1}^{S})$.

Compared with the Merkle tree, there are two major differences in the construction of BMT. First, instead of d_i, we use u_i to compute the hash value stored in the leaf node. That is why our proposal does not need authenticators and tags. This modification enables blockless verfication and reduces the communication cost in the integrity checking process. Second, extra information (i.e., ι and l_ι) is embedded in the hash values which fixes the vulnerability of the traditional Merkle tree.

Fig. 3 shows an illustration of the tree building algorithm with $D = 4$ and $S = 3$. In this illustration, $u_1 := g_1^{d_{1,1}} g_2^{d_{1,2}} g_3^{d_{1,3}}$ and $\nu_4 := (4, 1, H(u_1))$.

The Proving Algorithm BMT.Prove()**.** With the input of the challenge $\{(i_b, n_b)\}_{b=1}^{B}$ where i_b indicates the challenged index, $n_b \in \mathbb{Z}_p^*$ is a coefficient, and B is the number of challenged blocks, the data blocks $\{d_i\}_{i=1}^{D}$, the tree τ, and $\{g_j\}_{j=1}^{S}$, this algorithm generates a proof. The detailed procedure is as follows.

1) Compute $u_{i_b} := \prod_{j=1}^{S} g_j^{d_{i_b,j}}$ and obtain $\{u_{i_b}\}_{b=1}^{B}$.
2) Compute $\mu_j := \sum_{b=1}^{B} n_b d_{i_b,j}$ and obtain $\{\mu_j\}_{j=1}^{S}$.
3) Compute the path from the root to the challenged leaf nodes, and the siblings θ of the path.
4) Return the proof $\varpi = (\{\mu_j\}_{j=1}^{S}, \theta, \{(\iota_{i_b}, u_{i_b})\}_{b=1}^{B})$.

The first step of the proving algorithm is not necessary in practice, since u_{i_b} can be computed in advance. One can also let the tree building algorithm output $\{u_i\}_{i=1}^{D}$ and store $(\tau, \{u_i\}_{i=1}^{D})$ as the authenticated structure.

The Proof Verification Algorithm BMT.Verify()**.** With the input of the challenge $\{(i_b, n_b)\}_{b=1}^{B}$, the proof ϖ, the root ν_1 of the tree τ, and $\{g_j\}_{j=1}^{S}$, this algorithm checks whether the proof is valid. The detailed procedure is as follows.

1) Parse ϖ as $\{\mu_j\}_{j=1}^{S}$, θ, and $\{(\iota_{i_b}, u_{i_b})\}_{b=1}^{B}$.
2) Return 0 if $\prod_{j=1}^{S} g_j^{\mu_j} = \prod_{b=1}^{B} u_{i_b}^{n_b}$ does not hold.
3) For each u_{i_b}, compute $s_{\iota_{i_b}} := H(u_{i_b})$.
4) Reconstruct the root from θ and $\{\nu_{\iota_{i_b}}\}_{b=1}^{B}$ where $\nu_{\iota_{i_b}} = (\iota_{i_b}, 1, s_{\iota_{i_b}})$. This reconstruction process is similar to the tree building algorithm. Return 0 if the reconstructed root is not equal to ν_1.
5) Return 1 which denotes the proof is valid. That means, every d_{i_b} corresponds to the i_bth leaf node and is not tampered $(1 \le b \le B)$.

The Tree Update Algorithm BMT.Update()**.** With the input of the tree τ, $\{g_j\}_{j=1}^{S}$, and an update which consists of a block index i, an operation op, and a data block $d_i = \{d_{i,j}\}_{j=1}^{S}$, this algorithm updates the data blocks and authenticated structure according to the operation[2]. If the operation is modification, the original ith data block will be replaced by d_i. If the operation is insertion, d_i will be inserted in front of the ith data block. If the operation is deletion, the ith data block will be removed. The detailed procedure is as follows.

1) Compute $u_i := \prod_{j=1}^{S} g_j^{d_{i,j}}$ and $s_{\iota_i} := H(u)$ if the operation is modification or insertion. Skip this step if the operation is deletion.
2) Migrate the original ith leaf node to the $(2\iota_i + 1)$th node if the operation is insertion, and to the $\lfloor \iota_i/2 \rfloor$th node if the operation is deletion. Skip this step if the operation is modification. This is the same with the dynamic operations in other binary trees.
3) Update the affected path via s_{ι_i} and the siblings in τ.
4) Return d_i and the updated path, which can be used for updating the data blocks, authenticated structure, and metadata.

The process for updating multiple data blocks at once is similar. After multiple rounds of updates, the tree needs to be rebalanced. The rebalance process of BMT is similar to other balanced trees, which includes rotation and value recomputation. Note that only the values stored in the affected nodes need to be recomputed, and the most time-consuming computation in the rebalancing process is hashing. Therefore, the rebalance process in BMT is as efficient as in the other balanced trees.

D. Permission-based Signature

In order to employ BMT in our scheme, the root ν_1 should be generated by unrevoked users. That means, ν_1 needs to

[2]To simplify the description, the syntax of the tree update algorithm here is slightly different from the one introduced in previous section.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2019.2925800, IEEE Transactions on Dependable and Secure Computing

be signed with a legal user's private key. To tackle the second challenge in Section III-B, we propose a novel cryptographic primitive, called *Permission-Based Signature* (PBS), to preserve anonymity and (offline) traceability, simultaneously. The setting of permission-based signature is similar to group signature, under which there is a manager and a group of users. Users can sign messages on behalf of the group. Unlike group signature where only the manager can reveal the signer's identity, permission-based signature allows every user in the group to obtain the signer's identity using a *unique* revealing key. Thus, in our system, users in the same group can obtain the signer's identity, i.e., revision history, without the help of an online manager. Furthermore, the proposed scheme supports revocation and the revoked users cannot collude with CSP or TPA.

1) Syntax: Before describing the construction, we first formally introduce the syntax of permission-based signature. Compared with group signature, a user in permission-based signature possesses two keys, one is called signing key, and the other is called revealing key. Either of the two keys can be empty, which means a user may only have the signing capability or the revealing capability. Obviously, permission-based signature implies group signature. To construct a group signature scheme from permission-based signature, the manager in PBS simply does not generate any revealing keys for users. Then, only the manager can obtain the signer's identity as in group signature.Nevertheless, it is difficult to construct a permission-based signature scheme from a group signature scheme, since it is not obvious how to generate unique revealing key for each user. In this way, permission-based signature is a stronger notion.

Definition 4. *A (dynamic) permission-based signature scheme is a 6-tuple* $(\mathsf{Init}, \mathsf{Gen}, \mathsf{Revoke}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Reveal})$.

- *The initialization algorithm* $\mathsf{Init}()$ *takes as input a security parameter, and outputs a public-private key pair.*
- *The key generation algorithm* $\mathsf{Gen}()$ *takes as input a private key and a user identity, and outputs a user key which consists of a signing key and a revealing key.*
- *The revocation algorithm* $\mathsf{Revoke}()$ *takes as input a private key and a user identity, and outputs the updated public-private key pair and user keys for all the unrevoked users.*
- *The signing algorithm* $\mathsf{Sign}()$ *takes as input a signing key and a message, and outputs a signature.*
- *The signature verification algorithm* $\mathsf{Verify}()$ *takes as input a public key, a message, and a signature, and outputs a decision about whether the signature is valid.*
- *The revealing algorithm* $\mathsf{Reveal}()$ *takes as input a revealing key and a valid signature, and outputs an identity.*

The correctness of permission-based signature is twofold. Roughly speaking, it means that the honestly generated signature should always be valid, and the revealing algorithm should always output the signer identity of a valid signature. For the sake of simplicity, we use user key rather than signing key and revealing key in the rest of this paper, which means every user has both of these two keys.

2) Security Definitions: The security of permission-based signature consists of anonymity and traceability. Note that the fundamental security property that any signature scheme needs to satisfy is unforgeability. In Appendix A, we will show that the traceability of PBS implies this unforgeability.

The anonymity of PBS is defined by the following game between a challenger and an adversary.

1) With the security parameter λ, the challenger launches the initialization algorithm $\mathsf{PBS.Init}(1^\lambda)$ to obtain a public-private key pair (pk, sk), and sends $(1^\lambda, pk)$ to the adversary.
2) The adversary can query the key generation oracle and the signing oracle for polynomial times. When id is submitted to the key generation oracle, the challenger launches the key generation algorithm $\mathsf{PBS.Gen}(sk, id)$ to obtain a user key uk_{id}, and sends uk_{id} to the adversary. Then, the challenger would launch the revocation algorithm $\mathsf{PBS.Revoke}(sk, id)$ immediately. When (id^*, m^*) is submitted to the signing oracle, the challenger first generates a user key uk_{id^*} if the user key does not exist, and launches the signing algorithm $\mathsf{PBS.Sign}(uk_{id^*}, m^*)$ to obtain the signature σ^*. Then σ^* is sent to the adversary.
3) The adversary chooses two identities (id_0, id_1) and a message m, and sends (id_0, id_1, m) to the challenger.
4) The challenger first chooses a random bit $\kappa \in \{0, 1\}$, and generates a user key uk_{id_κ} if it does not exist. Then, m is signed with the user key uk_{id_κ}, and the signature is sent to the adversary. Further, the challenger generates certificates for these two identities and sends the certificates to the adversary.
5) The adversary outputs a bit κ'. We say that the adversary wins if $\kappa' = \kappa$.

Definition 5 (Anonymity)**.** *A permission-based signature scheme guarantees the anonymity if for any PPT adversary, the probability that the adversary wins is negligible greater than 1/2.*

The traceability of PBS is defined by the following game between a challenger and an adversary.

1) With the security parameter λ, the challenger launches the initialization algorithm $\mathsf{PBS.Init}(1^\lambda)$ to obtain a public-private key pair (pk, sk), and sends $(1^\lambda, pk)$ to the adversary. The challenger also initializes an empty set I.
2) The adversary can query the key generation oracle and the signing oracle for polynomial times. When id is submitted to the key generation oracle, the challenger launches the key generation algorithm $\mathsf{PBS.Gen}(sk, id)$ to obtain a user key uk_{id}, and sends uk_{id} to the adversary. Then, id is added into I. When (id^*, m^*) is submitted to the signing oracle, the challenger first generates a user key uk_{id^*} if the user key does not exist, launches the signing algorithm $\mathsf{PBS.Sign}(uk_{id^*}, m^*)$ to obtain the signature σ^*, and then sends σ^* to the adversary.
3) The adversary outputs a message-signature pair (m, σ). We say that the adversary wins if the following three

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2019.2925800, IEEE Transactions on Dependable and Secure Computing

conditions hold: 1) m is never submitted to the signing oracle; 2) (m, σ) is valid; 3) the returned identity via the revealing algorithm is not in the set I.

Definition 6 (Traceability)**.** *A PBS scheme guarantees the traceability if for any PPT adversary, the probability that the adversary wins is negligible.*

3) Construction: We now detail a concrete construction of permission-based signature. Let $\mathbb{G}$ and $\mathbb{G}_T$ be two multiplicative cyclic groups of prime order p, and g be a fixed generator of $\mathbb{G}$. Let $H_G : \{0,1\}^* \to \mathbb{G}$ and $H_Z : \{0,1\}^* \to \mathbb{Z}_p^*$ be two cryptographic hash functions. Our implementation utilizes a bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$, which satisfies the following conditions. 1) Bilinearity: $\forall u, v \in \mathbb{G}$, and $\forall a, b \in \mathbb{Z}_p^*$, $e(u^a, v^b) = e(u, v)^{ab}$. 2) Non-degeneracy: $e(g, g) \neq 1_{\mathbb{G}_T}$. 3) Efficient computation: the operations on group and the bilinear map are efficiently computable. The six PPT algorithms of our PBS construction are described below. Note that instead of directly using the identity, we generate a certificate for each user, which can be derived from the signing key. This is because the identity can be arbitrary strings, while the computation in our construction is for group elements.

The Initialization Algorithm PBS.Init()**.** With the input of a security parameter λ, this algorithm generates a public-private key pair (pk, sk). The detailed procedure is as follows.

1) Choose a group $\mathbb{G}$ of prime order p, where p is of λ bits of length.
2) Choose two random generators $g, h \in \mathbb{G}$.
3) Choose three random elements $\alpha, \beta \in \mathbb{Z}_p^*$ and $\eta \in \mathbb{G}$, and compute $v := h^\alpha$, $w := h^{-\beta}$, and $A := e(\eta, h)$.
4) Set the public key $pk = (g, h, v, w, A)$ and the private key $sk = (\alpha, \beta, \eta)$.

The Key Generation Algorithm PBS.Gen()**.** With the input of a private key sk and a user identity $id \in \{0,1\}^*$, this algorithm produces a user key uk_{id} and a corresponding certificate $cert_{id}$ for that user. The detailed procedure is as follows.

1) Choose a random element $x_{id} \in \mathbb{Z}_p^*$, and compute $y_{id} := g^{\frac{1}{\alpha + x_{id}}}$ and $z_{id} := \eta H_G(id)^\beta$.
2) Compute $C_{id} := e(y_{id}, v)$.
3) Set the user key $uk_{id} = (x_{id}, y_{id}, z_{id})$ and the certificate $cert_{id} = C_{id}$.

We call (x_{id}, y_{id}) the signing key and z_{id} the revealing key since (x_{id}, y_{id}) is only used in the signing algorithm while z_{id} is only used in the revealing algorithm.

The Revocation Algorithm PBS.Revoke()**.** With the input of a private key sk and a user identity id, this algorithm updates the public-private key pair for the system and user keys for all the unrevoked users. The detailed procedure is as follows.

1) Extract (x_{id}, y_{id}) from the user key uk_{id}.
2) Choose two random elements $\beta'' \in \mathbb{Z}_p^*$ and $\eta'' \in \mathbb{G}$, and compute $\beta' := \beta + \beta''$ and $\eta' := \eta\eta''$.
3) Set the updated private key $sk' = (\alpha, \beta', \eta')$.
4) Compute $g' := g^{\frac{1}{\alpha + x_{id}}}$, $w' := h^{-\beta'}$, and $A' := e(\eta', h)$.
5) Set the updated public key $pk' = (g', h, v, w', A')$.
6) For each unrevoked id^*, compute $y'_{id^*} := (y_{id}/y_{id^*})^{\frac{1}{x_{id^*} - x_{id}}}$, $z''_{id^*} := \eta'' H_G(id^*)^{\beta''}$, and $C'_{id^*} := e(y'_{id^*}, v)$.
7) Set the updated user key $uk'_{id^*} = (x_{id^*}, y'_{id^*}, z_{id^*} z''_{id^*})$ and the updated certificate $cert'_{id^*} = C'_{id^*}$.

The revocation algorithm could be extended to support batch revocation, which reduces the times of time-consuming operations when revoking multiple users.

The Signing Algorithm PBS.Sign()**.** With the input of a user key uk_{id}, a certificate $cert_{id}$, and a message $m \in \{0,1\}^*$, this algorithm outputs a signature σ. The detailed procedure is as follows.

1) Choose two random elements $t_1, t_2 \in \mathbb{Z}_p^*$ and compute $c_1 := y_{id}^{t_1}$, $c_2 := h^{t_2}$, $c_3 := w^{t_2}$, and $c_4 := C_{id} A^{t_2}$.
2) Choose six random elements $r_x, r_{t_1}, r_{t_2}, r_{\zeta_1}, r_{\zeta_2}, r_\xi \in \mathbb{Z}_p^*$ and compute the following values $r_1 := e(c_1, h^{r_{\zeta_2}} v^{r_{t_2}}) e(g, h)^{-r_\xi}$, $r_2 := c_4^{r_{t_1}} e(c_1, h)^{r_x} A^{-r_\xi} e(g, h)^{-r_{t_1}}$, $r_3 := h^{r_{t_2}}$, $r_4 := w^{r_{t_2}}$, $r_5 := c_2^{r_x} h^{-r_{\zeta_2}}$, $r_6 := c_3^{r_x} w^{-r_{\zeta_2}}$, $r_7 := c_2^{r_{t_1}} h^{-r_\xi}$, and $r_8 := c_3^{r_{t_1}} w^{-r_\xi}$.
3) Compute the hash value
$$c := H_Z(m, c_1, c_2, c_3, c_4, r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8).$$
4) Compute $s_x := r_x + c x_{id}$, $s_{t_1} := r_{t_1} + c t_1$, $s_{t_2} := r_{t_2} + c t_2$, $s_{\zeta_1} := r_{\zeta_1} + c x_{id} t_1$, $s_{\zeta_2} := r_{\zeta_2} + c x_{id} t_2$, and $s_\xi := r_\xi + c t_1 t_2$.
5) Set $\sigma = (c_1, c_2, c_3, c_4, c, s_x, s_{t_1}, s_{t_2}, s_{\zeta_1}, s_{\zeta_2}, s_\xi)$.

The Signature Verification Algorithm PBS.Verify()**.** With the input of a public key pk, a message m, and a signature σ, this algorithm outputs 1 if the message-signature pair is valid, and 0 otherwise. The detailed procedure is as follows.

1) Compute the following values $\tilde{r}_1 := e(c_1, h^{s_{\zeta_2}} v^{s_{t_2}}) e(g, h)^{-s_\xi}$, $\tilde{r}_2 := c_4^{s_{t_1}} e(c_1, h)^{s_x} A^{-s_\xi} e(g, h)^{-s_{t_1}}$, $\tilde{r}_3 := h^{s_{t_2}} c_2^{-c}$, $\tilde{r}_4 := w^{s_{t_2}} c_3^{-c}$, $\tilde{r}_5 := c_2^{s_x} h^{-s_{\zeta_2}}$, $\tilde{r}_6 := c_3^{s_x} w^{-s_{\zeta_2}}$, $\tilde{r}_7 := c_2^{s_{r_1}} h^{-s_\xi}$, and $\tilde{r}_8 := c_3^{s_{r_1}} w^{-s_\xi}$.
2) Return 1 if $H_Z(m, c_1, c_2, c_3, c_4, \tilde{r}_1, \tilde{r}_2, \tilde{r}_3, \tilde{r}_4, \tilde{r}_5, \tilde{r}_6, \tilde{r}_7, \tilde{r}_8)$ is equal to c, and 0 otherwise.

The Revealing Algorithm PBS.Reveal()**.** With the input of a user key uk_{id} and a valid signature σ, this algorithm outputs an identity id^* or $\perp$ to declare a failure. The detailed procedure is as follows.

1) Compute $C := \frac{c_4}{e(z_{id}, c_2) e(H_G(id), c_3)}$.
2) Output id^* if $cert_{id^*}$ is equal to C, and $\perp$ otherwise.

E. PRAYS

Now, we are ready to present PRAYS, which consists of seven phases: *initialization*, *registration*, *revocation*, *uploading*, *reading*, *writing*, and *auditing*.

Initialization Phase. In this phase, the manager, who decides a security parameter λ, executes as follows.

1) Call the initialization algorithm PBS.Init(1^λ) to obtain a public-private key pair (pk, sk). Publish the public key pk and keep the private key sk secret.
2) Create two databases as shown in Fig. 4: the certificate database, which can be delivered to users, and the user

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2019.2925800, IEEE Transactions on Dependable and Secure Computing

Identity	Certificate
id	$cert_{id}$
...	...

(a) The certificate database

Identity	User key
id	uk_{id}
...	...

(b) The user key database

Fig. 4. The databases maintained by the manager.

key database, which is only possessed and accessed by the manager.

Registration Phase. The registration phase is the only phase that needs a secure channel between the manager and the user. The user launches this phase by sending his/her identity id to the manager. Then, the manager executes as follows with (pk, sk) after authenticating the user identity.

1) Call the key generation algorithm $\mathsf{PBS.Gen}(sk, id)$ to obtain a user key uk_{id} and a certificate $cert_{id}$.
2) Insert $(id, cert_{id})$ and (id, uk_{id}) into the certificate database and user key database, respectively.
3) Send $(uk_{id}, cert_{id})$ to the user.

The user stores his/her user key and the corresponding certificate if all the following three equations hold: 1) $e(y_{id}, v) = C_{id}$; 2) $e(y_{id}, vh^{x_{id}}) = e(g, h)$; and 3) $e(z_{id}, h)e(H_G(id), w) = A$. Otherwise, the user may resubmit his/her identity id to the manager.

Revocation Phase. Assuming the identity of the revoked user is id, the manager executes as follows with (pk, sk).

1) Call the revocation algorithm $\mathsf{PBS.Revoke}(sk, id)$ to obtain an updated public-private key pair (pk', sk'). Publish the public key pk' and keep the private key sk' secret.
2) Update the user keys and the corresponding certificates of all the unrevoked users in the databases as shown in Fig. 4.
3) Remove row (id, uk_{id}) from the user key database, while reserve row $(id, cert_{id})$ in the certificate database for revealing revision history.
4) Send $(x_{id}, y_{id}, z''_{id^*})$, which can be obtained from the revocation algorithm $\mathsf{PBS.Revoke}(sk, id)$, to an unrevoked user whose identity is id^*.
5) Regenerate the state information with the manager's private key if the latest state information is generated by this revoked user (see the uploading and writing phases for the process of generating the state information)[3].

Upon receiving $(x_{id}, y_{id}, z''_{id^*})$, the unrevoked user whose identity is id^* executes as follows.

1) Compute $y'_{id^*} := (y_{id}/y_{id^*})^{\frac{1}{x_{id^*} - x_{id}}}$, $z'_{id^*} := z_{id^*} z''_{id^*}$, and $C'_{id^*} := e(y'_{id^*}, v)$.
2) Examine y'_{id^*}, z'_{id^*}, and C'_{id^*} as shown in the registration phase. Update its user key $uk'_{id^*} = (x_{id^*}, y'_{id^*}, z'_{id^*})$ and certificate $cert'_{id^*} = C'_{id^*}$ if all equations hold.

Note that the revocation phase requires neither a secure channel nor online unrevoked users. In practice, the information in Step 4 of the revocation phase (i.e., $(x_{id}, y_{id}, z''_{id^*})$) can be published to bulletin boards rather than directly being sent to unrevoked users.

Uploading Phase. We assume that the file to be uploaded consists of D data blocks $\{d_i\}_{i=1}^D$. To upload this file, the user, who possesses a user key uk_{id}, executes as follows.

1) Call the tree building algorithm $\mathsf{BMT.Build}(\{d_i\}_{i=1}^D)$ to obtain τ, $\{g_j\}_{j=1}^S$, and $\{u_i\}_{i=1}^D$.
2) Call the signing algorithm $\mathsf{PBS.Sign}(uk_{id}, m)$ to obtain a signature σ, where $m = \nu_1 \| g_1 \| \dots \| g_S$, i.e., m is the metadata of the authenticated structure.
3) Set the state information $st = (m, \sigma)$.
4) Send $\{d_i\}_{i=1}^D$ along with τ, $\{u_i\}_{i=1}^D$, and st to CSP[4], and send the state information st to TPA (or to the bulletin board as explained in Section II).

When CSP and TPA receive st from a user, they can call the verification algorithm $\mathsf{PBS.Verify}(pk, m, \sigma)$ to check whether these information is sent from a legal user.

Reading Phase. We assume that the user possesses the latest state information st which can be downloaded from the bulletin board. The user simply sends a block index i to CSP, who then executes as follows.

1) Compute the path from the root to the ith leaf node and the siblings θ of the path.
2) Send (ι_i, d_i, θ) to the user.

The user, who possesses a user key uk_{id} and state information st, then executes as follows.

1) Parse $st = (m, \sigma)$, where $m = \nu_1 \| g_1 \| \dots \| g_S$.
2) Compute $u_i := \prod_{j=1}^S g_j^{d_{i,j}}$.
3) Terminate this process and report a failure if the proof verification algorithm $\mathsf{BMT.Verify}((i, 1), (d_i, \theta, (\iota_i, u_i)), \nu_1, \{g_j\}_{j=1}^S)$ returns 0. Otherwise, d_i is accepted.
4) Call the revealing algorithm $\mathsf{PBS.Reveal}(uk_{id}, \sigma)$ to obtain the revision history.

Writing Phase. Our scheme supports fully dynamic operations, including modification, insertion, and deletion. The user, who possesses a user key uk_{id}, a block index i, a data block d_i, and the state information st, executes as follows.

1) Parse $st = (m, \sigma)$, where $m = \nu_1 \| g_1 \| \dots \| g_S$.
2) Choose op according to the operation, and call the tree update algorithm $\mathsf{BMT.Update}$ to obtain u_i and the updated path. We assume that the updated root is ν'_1.
3) Call the signing algorithm $\mathsf{PBS.Sign}(uk_{id}, m')$ to obtain a signature σ', where $m' = \nu'_1 \| g_1 \| \dots \| g_S$.
4) Send d_i along with $st' = (m', \sigma')$, u_i, and the updated path to CSP, and send st' to TPA (or to the bulletin board).

Meanwhile, CSP and TPA can verify st' via the verification algorithm $\mathsf{PBS.Verify}(pk, m', \sigma')$ as in the uploading phase.

Auditing Phase. With the input of the public key pk and the latest state information st, TPA executes as follows.

1) Choose B random indexes $\{i_b\}_{b=1}^B$ and B random elements $\{n_b\}_{b=1}^B$ where $n_j \in \mathbb{Z}_p^*$.

[3]The manager's private key is obtained by performing the registration phase locally on the manager side.

[4]Actually, $\{u_i\}_{i=1}^D$ can be computed by CSP, however, that increases the computation cost of CSP. See the discussion on the proving algorithm in Section III-C.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2019.2925800, IEEE Transactions on Dependable and Secure Computing

2) Send $\{(i_b, n_b)\}_{b=1}^{B}$ to CSP.

Note that the indexes can be generated from a pseudorandom permutation, and the elements can be generated from a pseudorandom function. Then, TPA only needs to send B and two keys (one is used in the pseudorandom permutation, and the other is used in the pseudorandom function) to CSP.

When receiving $\{(i_b, n_b)\}_{b=1}^{B}$, CSP executes as follows.

1) Call the proving algorithm $\mathsf{BMT.Prove}(\{(i_b, n_b)\}_{b=1}^{B}, \{d_i\}_{i=1}^{D}, \tau, \{u_i\}_{i=1}^{D})$ to obtain the proof ϖ.
2) Send ϖ to TPA.

TPA verifies the response as follows.

1) Terminate and report a failure if the proof verification algorithm $\mathsf{BMT.Verify}(\{(i_b, n_b)\}_{b=1}^{B}, \varpi, \nu_1, \{g_j\}_{j=1}^{S})$ returns 0.
2) Accept the response and report a success.

IV. Security Analysis

In this section, we examine the security properties of PRAYS, including integrity, anonymity, and traceability (see Section I). Secure user revocation is considered in the latter two properties. Note that, the formal definitions for dynamic group-oriented provable data possession are almost the same with the definitions for blockless Merkle tree and permission-based signature, and we formalize the latter two in Section III-C and III-D, respectively. Therefore, we omit the formal definitions for dynamic group-oriented provable data possession to avoid redundant definitions, and explain why the security of BMT and PBS schemes implies the security of PRAYS. Generally, integrity of PRAYS is guaranteed by the BMT scheme while anonymity and traceability of PRAYS are achieved by the PBS scheme. The proofs for anonymity (Theorem 2) and traceability (Theorem 3) are given in Appendix A. Roughly speaking, the security (i.e., anonymity and traceability) of proposed PBS scheme is based on a zero-knowledge proof protocol (also see Appendix A).

A. Integrity

Integrity means that TPA can reconstruct the challenged data blocks if CSP passed the checking process [31]. Since TPA reports a success if and only if the proof verification algorithm of BMT returns 1, it is obvious that the integrity of PRAYS can be reduced to the integrity of the BMT scheme once TPA possesses the latest state information, that is guaranteed by the unforgeability of the PBS scheme (which is implied by the traceability of the PBS scheme).

Theorem 1. *The proposed BMT scheme guarantees the integrity if the hash function is collision-resistant and the discrete logarithm problem is hard in $\mathbb{G}$.*

Proof: We first prove that every u_{i_b} corresponds to the i_bth leaf node and is not tampered and is up-to-date where $1 \leq b \leq B$ if the reconstructed root is equal to ν_1. Since the hash function is collision-resistant, the adversary could not find two values ν_2^* and ν_3^* such that $H(\nu_2^* \| \nu_3^*) = s_1$; otherwise, a collision is found immediately. Likewise, the adversary cannot tamper ν_4, ν_5, ..., which implies that the siblings θ and $\{(\iota_{i_b}, u_{i_b})\}_{b=1}^{B}$ (i.e., the last two parts of the proof ϖ) can only be generated from the latest tree τ. Since (ι, l_ι) in ν_ι can be used to determine the structure of the tree, the adversary cannot send wrong positions either.

Then, we prove that the challenger can reconstruct challenged blocks if $\prod_{j=1}^{S} g_j^{\mu_j} = \prod_{b=1}^{B} u_{i_b}^{n_b}$ (i.e., pass the second step of the proof verification algorithm). Note that the right side of the equation is a constant from the perspective of the challenger since every u_{i_b} corresponds to the i_bth leaf node and n_i is generated by the challenger. Since the discrete logarithm problem is hard in $\mathbb{G}$, the adversary cannot output $\{\mu_1^*, \ldots, \mu_S^*\}$ that satisfies the following two conditions: 1) exists $\mu_j^* \neq \mu_j$ for some j; 2) $\prod_{j=1}^{S} g_j^{\mu_j^*} = \prod_{i=1}^{B} u_i^{n_i}$. Therefore, every μ_j is a linear combination of challenged segments $\{d_{i_b,j}\}_{b=1}^{B}$. Then, the challenger could generate other challenges with the same indexes $\{i_b\}_{b=1}^{B}$ but different coefficients $\{n_b\}_{b=1}^{B}$ for B times, and reconstruct the challenged blocks by solving a system of linear equations. Thus, the proposed BMT scheme guarantees integrity. ■

B. Anonymity

Anonymity means that the entities outside of a group, including CSP, TPA, and revoked users, cannot learn the signer's identity. That is, PRAYS guarantees anonymity if the proposed PBS scheme provides anonymity.

Theorem 2. *The proposed PBS scheme guarantees the anonymity under the random oracle model if the Computational Diffie-Hellman (CDH) problem is hard in $\mathbb{G}$, and the Decisional Bilinear Diffie-Hellman (DBDH) problem is hard.*

C. Traceability

Traceability means that the users within the same group can learn the signer's identity in the reading phase, while the entities outside of the group, including CSP, TPA, and the revoked users, cannot forge or tamper the signature. That is, PRAYS guarantees traceability if the proposed PBS scheme provides traceability.

Theorem 3. *The proposed PBS scheme guarantees the traceability under the random oracle model if the Strong Diffie-Hellman (SDH) problem is hard in $\mathbb{G}$.*

V. Performance Evaluation

In this section, we first theoretically analyze the performance of PRAYS. Then, we examine PRAYS through extensive experiments.

A. Theoretical Analysis

The theoretical analysis focuses on the storage cost and the revocation cost. In the analysis, N denotes the number of users; D denotes the number of data blocks; U denotes the number of unrevoked users; and R denotes the number of data blocks signed by the revoked user.

We first analyze the storage cost of PRAYS which is $O(N)$ on the user side, $O(1)$ on the TPA side, and $O(D)$ on the CSP side, respectively. Although both the user key and the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2019.2925800, IEEE Transactions on Dependable and Secure Computing

TABLE II
STORAGE COST COMPARISON

	User	TPA	CSP
PRAYS	$O(N)$	$O(1)$	$O(D)$
Oruta [12]	$O(N)$	$O(N+D)$	$O(D)$
Panda [11]	$O(N)$	$O(N+D)$	$O(D)$
YY [10]	$O(N)$	$O(N+D)$	$O(D)$
JCM [14]	$O(D)$	$O(D)$	$O(D)$

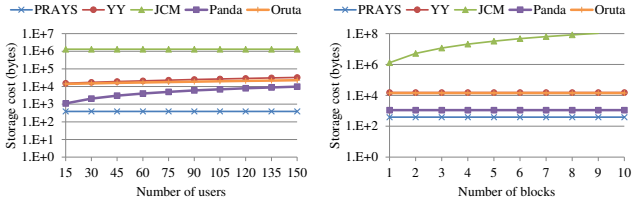
public key in PRAYS are $O(1)$, the user needs to possess the certificate database to obtain the revision history, which makes the storage cost of PRAYS on the user side be $O(N)$[5]. TPA maintains auditing metadata which consists of the public key and the state information. CSP stores users' data and extra information used for integrity checking, and both of them are $O(D)$ in PRAYS. Table II shows the storage cost of PRAYS compared with other schemes[6]. From Table II, PRAYS is the only scheme that achieves constant auditing metadata.

Then, we analyze the cost in the revocation phase. When a user is revoked, both the computation cost and the communication cost of PRAYS are $O(1)$ on the unrevoked user side and $O(U)$ on the manager side, respectively. That is because the manager needs to compute and publish z''_{id^*} for every unrevoked user while each unrevoked user only receives and updates its own user key. Table III shows the computation cost and communication cost of PRAYS compared with other schemes. Since U is much smaller than D and R in practice, we believe that PRAYS in the revocation phase is more efficient than most of the existing schemes.

B. Experiments and Analysis

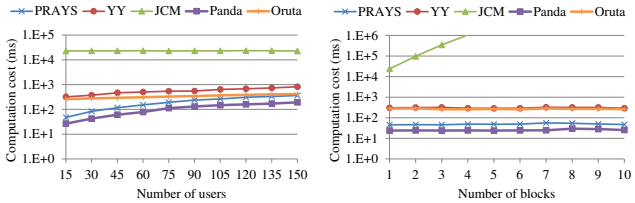
We implemented PRAYS and related schemes by the PBC library 0.5.14. CSP and TPA are desktops running Ubuntu 16.04 with an Intel 2.6GHz CPU and 8GB memory. Users and the manager are laptops running Ubuntu 14.04 with an Intel 2.5GHz CPU and 4GB memory. In all implementations, we fixed the security parameter to 160 bits and the block size to 4kB as in [10]. The experiments focused on four phases: initialization, registration, uploading, and auditing. The reading and writing phases are similar to the auditing and uploading phases, respectively, except for that there is only one block involved. The revocation phase has been analyzed in Section V-A. Therefore, we did not consider these phases. The files of specified sizes used in our experiments are randomly generated by a Python script. All experimental results are the average of 10 trials.

We first examine the size of the public key generated in the initialization phase, which is later delivered to users and TPA. Note that the scheme in JCM needs to determine D in advance. Therefore, we first consider that there is only one block, and Fig. 5(a) shows the result in this case. The size of the public key is independent with N in PRAYS and JCM, while the size grows with N increasing in other schemes. Fig. 5(b) presents the relationship between the size of the public key and D when $N = 15$. The size grows as D is raised in JCM, while the size is independent with D in other schemes. Therefore, among the five examined schemes, PRAYS is the only scheme whose public key size is independent with N and D.



(a) Storage cost vs. N (b) Storage cost vs. D

Fig. 5. Storage cost for the public key.



(a) Computation cost vs. N (b) Computation cost vs. D

Fig. 6. Computation cost for key generation.

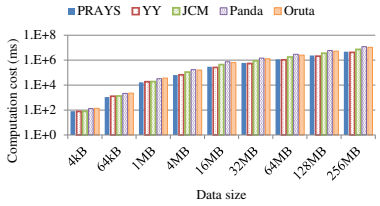


Fig. 7. Computation cost in the upload phase.

Then, we examine the computation cost in the initialization and registration phases, under which the system generates a public key and all user keys. Fig. 6(a) shows the results when $D = 1$. The computation cost increases with N raised for all the five schemes, since the system needs to generate a private key for each user. Then, we fix $N = 15$, and investigate the computation cost with different D as shown in Fig. 6(b). The computation cost for all schemes except JCM is independent with D. Since the computation cost in JCM grows as D rising and is 96 seconds when there are 2 blocks (i.e., the file is only 8kB), it is not suitable for large files. As a result, PRAYS is as efficient as YY, Panda, and Oruta for key generation.

In the uploading phase, we investigate both the computation and the communication cost on the user side. Fig. 7 presents the computation cost, under which PRAYS, YY, and JCM have almost the same performance. That is because the uploader needs to execute one exponentiation for each element in $\mathbb{Z}_p$. Panda and Oruta are less efficient than the other three schemes in this phase since they need more exponentiations for each element in $\mathbb{Z}_p$. The uploading phase is the most time-consuming phase for all these five schemes.

The communication cost in the uploading phase consists of

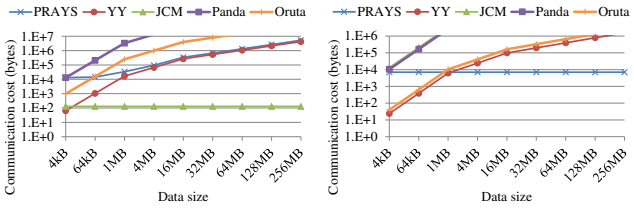
[5]This storage cost can be reduced to $O(1)$ via the encryption scheme in [32] and any digital signature scheme. However, we do not consider this extension for simplicity.

[6]Some schemes do not indicate that whether users should possess the public key. We think that users need to verify the tags when they read a data block from the cloud as shown in Section III-E. Therefore, the storage cost on the user sider considers the public key in all the schemes.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2019.2925800, IEEE Transactions on Dependable and Secure Computing

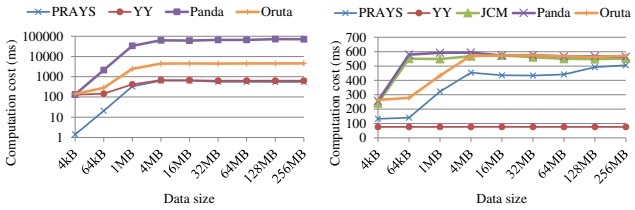
TABLE III
REVOCATION COST COMPARISON

	Unrevoked User		Manager		CSP	
	Computation	Communication	Computation	Communication	Computation	Communication
PRAYS	$O(1)$	$O(1)$	$O(U)$	$O(U)$	N/A	$O(1)$
Oruta [12]	$O(D)$	$O(D)$	N/A	N/A	N/A	$O(D)$
Panda [11]	$O(1)$	$O(1)$	N/A	N/A	$O(R)$	$O(1)$
YY [10]	N/A	N/A	N/A	$O(1)$	$O(R)$	$O(1)$
JCM [14]	$O(1)$	$O(1)$	$O(1)$	$O(1)$	N/A	$O(1)$



(a) Extra storage on the cloud (b) State information

Fig. 8. Communication cost in the upload phase.



(a) Cost on CSP (b) Cost on TPA

Fig. 9. Computation cost in the auditing phase.

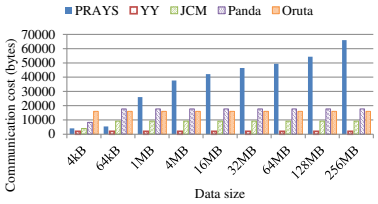


Fig. 10. Communication cost in the auditing phase.

three parts: data size, the size of the authenticated structure, and the size of the state information. In our experiments, we omit the communication cost for transmitting the data to the cloud since this cost is the same for all schemes, and focus on the other two parts (which have been theoretical analyzed in Table II). Fig. 8(a) shows the size of the authenticated structure with respect to the data size when $N = 15$, which needs to be stored at CSP. In PRAYS, CSP needs to store the entire tree τ and $\{u_i\}_{i=1}^{D}$. Therefore, the communication cost of PRAYS grows with D raised. The communication cost of Panda grows with D and N raising, since for each data block, the signer has to generate a ring signature whose size grows linearly with N. Thus, Panda is not suitable for large groups. YY and JCM are efficient in terms of communication cost in this phase, however, they do not support insertion operation at all. Fig. 8(b) presents the size of the state information, which needs to be stored at TPA (or the bulletin boards). The cost is constant for PRAYS while the cost grows linearly as D rises in other schemes. As a result, PRAYS and YY have their own advantages in this phase while other schemes are inefficient in practice.

In the auditing phase, we examine the computation costs at CSP and TPA, respectively, and the communication cost between CSP and TPA. As in other schemes, we fix $B = 460$, which has been proved that it is sufficient for auditing [4]. Fig. 9(a) shows the computation cost on CSP[7]. When the data size is less than 1.8MB, D is less than 460 in all schemes, and TPA challenges all the data blocks in this case. Therefore, the computation cost grows with D for all schemes. When $D > 460$, the computation cost grows with D for PRAYS, which is caused by computing siblings, while the cost is constant in other schemes[8]. Fig. 9(b) presents the computation cost on TPA, in which the result is similar to Fig. 9(a). Note that the growth rate of PRAYS in Figure 9 is extremely slow when the file is larger than 4MB. That is because the time-consuming operations are constant when the number of challenged blocks stops increasing.

Fig. 10 shows the communication cost between CSP and TPA. The communication cost grows slowly with D for PRAYS, while the cost is constant in other schemes when the data size is larger than 1.8MB. This communication cost is acceptable since PRAYS is the only solution that supports fully dynamic operations.

[7]In JCM, CSP simply reads 460 blocks and sends them to TPA without any computation, and therefore the computation cost is omitted.

[8]The computation cost in YY grows with the number of users who generate the challenged blocks increasing. However, we fix this number to 1.

VI. RELATED WORK

Single-Writer Solutions. Integrity checking in the cloud was first explored under the single-writer model for personal data management. Juels and Kaliski introduced the concept of Proof of Retrievability (PoR) and proposed a concrete construction [5]. Unfortunately, their scheme only allows limited times of integrity checking. Ateniese *et al.* independently introduced a similar concept, called Provable Data Possession (PDP) [4]. Their scheme allows unlimited times of integrity checking, and supports *public auditing*, which means anyone can check the data integrity. This property is highly preferred since users can delegate the checking capability to third-party verifiers for alleviating the computation burden. Nevertheless, their scheme does not support dynamic operations.

Subsequent works devoted to integrity checking schemes for dynamic data [15], [16], [24], [30], [33]–[35]. In public auditing, in addition to the public key, the verifiers usually maintain some information about the current status of the audited data, called *state information*. The state information

is crucial to dynamic cloud storage and leverages which the verifier determines whether the stored data on the cloud is up-to-date. However, applying those schemes to the multi-writer model would raise performance concern in the revocation process.

Multi-Writer Solutions. Researchers then focus on integrity checking schemes under the multi-writer model which supports data sharing among a group of users [25].Wang *et al.* introduced the concept of group-oriented proofs of storage, but did not consider any dynamic data operations [13].

Wang *et al.* proposed a public auditing scheme, called Oruta, which guarantees identity privacy [12]. However, Oruta does not support non-trivial user revocation due to the inherent property of the ring signature. To support user revocation, Wang *et al.* proposed another solution, called Panda [9], [11]. Nevertheless, Panda could not resist the collusion between the cloud and the revoked user.

Yuan and Yu proposed a public integrity checking scheme for data sharing, which supports secure user revocation [10]. Unfortunately, their scheme does not satisfy the security definition for integrity checking in the cloud [31] as opposed to previous schemes. That is, no one can extract the challenged blocks during the checking process in their scheme.

Jiang *et al.* proposed a public integrity auditing scheme for shared data based on group signature and vector commitment [14]. Their solution supports secure user revocation and guarantees identity privacy. However, it cannot reveal *revision history* to users, since with group signature, even group members could not identify who has updated the shared data. Furthermore, their solution requires that the data size has to be fixed and determined at the beginning of system initialization, which makes their solution less flexible.

VII. Conclusion and Future Work

In this paper, we proposed a privacy-preserving auditing scheme for dynamic shared data, named PRAYS. It is the first group-oriented provable data possession scheme that supports fully dynamic operations as well as constant auditing metadata to our knowledge. The proposed scheme is boosted by a new two-step paradigm designed for group-oriented integrity checking. In order to realize this paradigm, we presented a blockless Merkle tree for the first step, and presented a permission-based signature for the second step. With these two tools, PRAYS provides all the essential features in the multi-writer storage services, including fully dynamic operations, constant auditing metadata, secure user revocation, anonymity, and traceability.

In our future work, we will extend PRAYS from the following aspects. 1) Reducing the storage cost on the user side to $O(1)$ as mentioned in Section V-A. 2) Optimizing the computation cost in the revocation phase. Unlike the traditional paradigm whose lower bound of the computation cost in the revocation phase is $O(R)$, it is possible to improve PRAYS by enhancing PBS.

Acknowledgment

This research was supported in part by the National Natural Science Foundation of China under grants 61702379, 61772383, U1836202, 61572380; by the China Postdoctoral Science Foundation under grant No. 2018M630877; by Science, Technology and Innovation Commission of Shenzhen Municipality under grant No. JCYJ20170303170108208. The corresponding author is Jing Chen.

References

[1] H. Wang, D. He, and S. Tang, "Identity-Based Proxy-Oriented Data Uploading and Remote Data Integrity Checking in Public Cloud," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 6, pp. 1165–1176, 2016.

[2] Y. Yu, M. H. Au, G. Ateniese, X. Huang, W. Susilo, Y. Dai, and G. Min, "Identity-Based Remote Data Integrity Checking With Perfect Data Privacy Preserving for Cloud Storage," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 4, pp. 767–778, 2017.

[3] Z. Ren, L. Wang, Q. Wang, and M. Xu, "Dynamic Proofs of Retrievability for Coded Cloud Storage Systems," *IEEE Transactions on Services Computing*, vol. 11, no. 4, pp. 685–698, 2018.

[4] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable Data Possession at Untrusted Stores," in *Proc. of CCS*, 2007.

[5] A. Juels and B. S. Kaliski, Jr., "PORs: Proofs of Retrievability for Large Files," in *Proc. of CCS*, 2007.

[6] G. Ateniese, S. Kamara, and J. Katz, "Proofs of Storage from Homomorphic Identification Protocols," in *Proc. of ASIACRYPT*, 2009.

[7] H. Wang, "Identity-based distributed provable data possession in multicloud storage," *IEEE Transactions on Services Computing*, vol. 8, no. 2, pp. 328–340, 2015.

[8] S. Guarino, E. S. Canlar, M. Conti, R. D. Pietro, and A. Solanas, "Provable storage medium for data storage outsourcing," *IEEE Transactions on Services Computing*, vol. 8, no. 6, pp. 985–997, 2015.

[9] B. Wang, B. Li, and H. Li, "Public Auditing for Shared Data with Efficient User Revocation in the Cloud," in *Proc. of INFOCOM*, 2013.

[10] J. Yuan and S. Yu, "Efficient Public Integrity Checking for Cloud Data Sharing with Multi-User Modification," in *Proc. of INFOCOM*, 2014.

[11] B. Wang, B. Li, and H. Li, "Panda: Public Auditing for Shared Data with Efficient User Revocation in the Cloud," *IEEE Transactions on Services Computing*, vol. 8, no. 1, pp. 92–106, 2015.

[12] ——, "Oruta: Privacy-Preserving Public Auditing for Shared Data in the Cloud," *IEEE Transactions on Cloud Computing*, vol. 2, no. 1, pp. 43–56, 2014.

[13] Y. Wang, Q. Wu, B. Qin, X. Chen, X. Huang, and Y. Zhou, "Group-oriented proofs of storage," in *Proceedings of ASIACCS*, 2015.

[14] T. Jiang, X. Chen, and J. Ma, "Public Integrity Auditing for Shared Dynamic Cloud Data with Group User Revocation," *IEEE Transactions on Computers*, vol. 65, no. 8, pp. 2363–2373, 2016.

[15] C. C. Erway, A. Küpçü, C. Papamanthou, and R. Tamassia, "Dynamic provable data possession," *ACM Trans. Inf. Syst. Secur.*, vol. 17, no. 4, pp. 15:1–15:29, 2015.

[16] E. Shi, E. Stefanov, and C. Papamanthou, "Practical Dynamic Proofs of Retrievability," in *Proc. of CCS*, 2013.

[17] C. Garman, M. Green, and I. Miers, "Decentralized Anonymous Credentials," in *Proceedings of NDSS*, 2014.

[18] J. Chen, S. Yao, Q. Yuan, K. He, S. Ji, and R. Du, "CertChain: Public and efficient certificate audit based on blockchain for tls connections," in *Proc. of INFOCOM*, 2018.

[19] S. Yao, J. Chen, K. He, R. Du, T. Zhu, and X. Chen, "PBCert: Privacy-preserving blockchain-based certificate status validation toward mass storage management," *IEEE Access*, vol. 7, pp. 6117–6128, 2019.

[20] M. Maffei, G. Malavolta, M. Reinert, and D. Schroder, "Privacy and Access Control for Outsourced Personal Records," in *Proc. of S&P*, 2015.

[21] J. Chen, K. He, Q. Yuan, M. Chen, R. Du, and Y. Xiang, "Blind filtering at third parties: An efficient privacy-preserving framework for location-based services," *IEEE Transactions on Mobile Computing*, vol. 17, no. 11, pp. 2524–2535, 2018.

[22] N. Unger, S. Dechand, J. Bonneau, S. Fahl, H. Perl, I. Goldberg, and M. Smith, "SoK: Secure Messaging," in *Proc. of S&P*, 2015.

[23] H. Corrigan-Gibbs, D. Boneh, and D. Mazieres, "Riposte: An Anonymous Messaging System Handling Millions of Users," in *Proc. of S&P*, 2015.

[24] Q. Wang, C. Wang, K. Ren, W. Lou, and J. Li, "Enabling Public Auditability and Data Dynamics for Storage Security in Cloud Computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 5, pp. 847–859, 2011.

[25] E. Stefanov, M. van Dijk, A. Juels, and A. Oprea, "Iris: A Scalable Cloud File System with Efficient Integrity Checks," in *Proceedings of the 28th Annual Computer Security Applications Conference*, 2012.

[26] D. Catalano, "Homomorphic Signatures and Message Authentication Codes," in *Proc. of SCN*, 2014.

[27] M. F. Ezerman, H. T. Lee, S. Ling, K. Nguyen, and H. Wang, "A Provably Secure Group Signature Scheme from Code-Based Assumptions," in *Proc. of ASIACRYPT*, 2015.

[28] B. Libert, S. Ling, K. Nguyen, and H. Wang, "Zero-Knowledge Arguments for Lattice-Based Accumulators: Logarithmic-Size Ring Signatures and Group Signatures Without Trapdoors," in *Proc. of EUROCRYPT*, 2016.

[29] A. Oprea and M. K. Reiter, "Integrity Checking in Cryptographic File Systems with Constant Trusted Storage," in *Proc. of USENIX Security*, 2007.

[30] C. Erway, A. Küpcü, C. Papamanthou, and R. Tamassia, "Dynamic Provable Data Possession," in *Proc. of CCS*, 2009.

[31] H. Shacham and B. Waters, "Compact Proofs of Retrievability," *Journal of Cryptology*, vol. 26, no. 3, pp. 442–483, 2013.

[32] Q. Wu, Y. Mu, W. Susilo, B. Qin, and J. Domingo-Ferrer, "Asymmetric Group Key Agreement," in *Proc. of EUROCRYPT*, 2009.

[33] G. Ateniese, R. Di Pietro, L. V. Mancini, and G. Tsudik, "Scalable and efficient provable data possession," in *Proc. of SecureComm*, 2008.

[34] D. Cash, A. Küpçü, and D. Wichs, "Dynamic Proofs of Retrievability via Oblivious RAM," in *Proc. of EUROCRYPT*, 2013.

[35] K. He, J. Chen, R. Du, Q. Wu, G. Xue, and X. Zhang, "DeyPoS: deduplicatable dynamic proof of storage for multi-user environments," *IEEE Transactions on Computers*, vol. 65, no. 12, pp. 3631–3645, 2016.

[36] A. Fiat and A. Shamir, "How to Prove Yourself: Practical Solutions to Identification and Signature Problems," in *Proceedings of CRYPTO*, 1986.

APPENDIX
SECURITY PROOFS FOR THE PBS SCHEME

In this section, we prove the security of the proposed PBS scheme. To prove the security, i.e., Theorem 2 and Theorem 3, we first show how the proposed scheme can be converted from a zero-knowledge proof protocol.

Let $\mathbb{G}$ and $\mathbb{G}_T$ be two multiplicative cyclic groups of prime order p, and $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ be a bilinear map. The system chooses two random generates $g, h \in \mathbb{G}$, and three random elements $\alpha, \beta \in \mathbb{Z}_p^*$ and $\eta \in \mathbb{G}$. Then, the public parameter is $(g, h, v := h^\alpha, w := h^{-\beta}, A := e(\eta, h))$. The proposed zero-knowledge proof protocol for an instance of the SDH problem consists of four stage: *commit*, *challenge*, *response*, and *verify*.

Commit stage. A prover possesses a pair of solution $(x, y := g^{\frac{1}{\alpha+x}})$ for a certain SDH problem, where $x \in \mathbb{Z}_p^*$, $y \in \mathbb{G}$, and the equation $e(y, vh^x) = e(g, h)$ holds. In order to prove the possession of such a solution, the prover chooses two random values $t_1, t_2 \in \mathbb{Z}_p^*$, and computes $c_1 := y^{t_1}$, $c_2 := h^{t_2}$, $c_3 := w^{t_2}$, $c_4 := e(y, v)A^{t_2}$. It also computes three auxiliary values $\zeta_1 := xt_1$, $\zeta_2 := xt_2$, and $\xi := t_1 t_2$. Then, it must prove to the verifier that it possesses the six-tuple $(x, t_1, t_2, \zeta_1, \zeta_2, \xi)$ which satisfies the following eight relations:

$$e(c_1, h^{\zeta_2} v^{t_2}) e(g, h)^{-\xi} = 1, \qquad c_4^{t_1} e(c_1, h)^x A^{-\xi} e(g, h)^{-t_1} = 1,$$
$$h^{t_2} = c_2, \qquad w^{t_2} = c_3,$$
$$c_2^x h^{-\zeta_2} = 1, \qquad c_3^x w^{-\zeta_2} = 1,$$
$$c_2^{t_1} h^{-\xi} = 1, \qquad c_3^{t_1} w^{-\xi} = 1.$$

Therefore, it chooses six random blinding values r_x, r_{t_1}, r_{t_2}, r_{ζ_1}, r_{ζ_2}, and r_ξ from $\mathbb{Z}_p^*$, and computes $r_1 := e(c_1, h^{r_{\zeta_2}} v^{r_{t_2}}) e(g, h)^{-r_\xi}$, $r_2 := c_4^{r_{t_1}} e(c_1, h)^{r_x} A^{-r_\xi} e(g, h)^{-r_{t_1}}$, $r_3 := h^{r_{t_2}}$, $r_4 := w^{r_{t_2}}$, $r_5 := c_2^{r_x} h^{-r_{\zeta_2}}$, $r_6 := c_3^{r_x} w^{-r_{\zeta_2}}$, $r_7 := c_2^{r_{t_1}} h^{-r_\xi}$, and $r_8 := c_3^{r_{t_1}} w^{-r_\xi}$. The 12-tuple $(c_1, c_2, c_3, c_4, r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8)$ is sent to the verifier.

Challenge stage. After receiving from the prover the 12-tuple $(c_1, c_2, c_3, c_4, r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8)$, the verifier chooses a random value $c \leftarrow \mathbb{Z}_p^*$, and sends c to the prover.

Response stage. When receiving the challenge c, the prover computes $s_x := r_x + cx_{id}$, $s_{t_1} := r_{t_1} + ct_1$, $s_{t_2} := r_{t_2} + ct_2$, $s_{\zeta_1} := r_{\zeta_1} + cx_{id}t_1$, $s_{\zeta_2} := r_{\zeta_2} + cx_{id}t_2$, and $s_\xi := r_\xi + ct_1 t_2$. Then, $(s_x, s_{t_1}, s_{t_2}, s_{\zeta_1}, s_{\zeta_2}, s_\xi)$ is sent to the verifier.

Verify stage. Finally, the verifier accepts the proof only if all the following equations hold.

$$e(c_1, h^{s_{\zeta_2}} v^{s_{t_2}}) e(g, h)^{-s_\xi} \stackrel{?}{=} r_1,$$
$$c_4^{s_{t_1}} e(c_1, h)^{s_x} A^{-s_\xi} e(g, h)^{-s_{t_1}} \stackrel{?}{=} r_2,$$
$$h^{s_{t_2}} c_2^{-c} \stackrel{?}{=} r_3, w^{s_{t_2}} c_3^{-c} \stackrel{?}{=} r_4, c_2^{s_x} h^{-s_{\zeta_2}} \stackrel{?}{=} r_5,$$
$$c_3^{s_x} w^{-s_{\zeta_2}} \stackrel{?}{=} r_6, c_2^{s_{r_1}} h^{-s_\xi} \stackrel{?}{=} r_7, c_3^{s_{r_1}} w^{-s_\xi} \stackrel{?}{=} r_8.$$

Lemma 1. *The protocol is complete.*

This lemma can be proved via verifying the equations in the verify stage, therefore, we omit the proof here.

Lemma 2. *The transcripts of the protocol can be simulated.*

Proof: The system selects three random values $t_1, t_2 \leftarrow \mathbb{Z}_p^*$ and $y \leftarrow \mathbb{G}$, and sets $c_1 := y^{t_1}$, $c_2 := h^{t_2}$, $c_3 := w^{t_2}$, $c_4 := e(y, v)A^{t_2}$. The distribution of the 4-tuple (c_1, c_2, c_3, c_4) is identical with any prover.

The 4-tuple (c_1, c_2, c_3, c_4) is then given to the simulator, and the simulator picks a random value $c \leftarrow \mathbb{Z}_p^*$. Then, after selecting six random value s_x, s_{t_1}, s_{t_2}, s_{ζ_1}, s_{ζ_2}, and s_ξ in $\mathbb{Z}_p^*$, the simulator computes $r_1 := e(c_1, h^{s_{\zeta_2}} v^{s_{t_2}}) e(g, h)^{-s_\xi}$, $r_2 := c_4^{s_{t_1}} e(c_1, h)^{s_x} A^{-s_\xi} e(g, h)^{-s_{t_1}}$, $r_3 := h^{s_{t_2}} c_2^{-c}$, $r_4 := w^{s_{t_2}} c_3^{-c}$, $r_5 := c_2^{s_x} h^{-s_{\zeta_2}}$, $r_6 := c_3^{s_x} w^{-s_{\zeta_2}}$, $r_7 := c_2^{s_{r_1}} h^{-s_\xi}$, and $r_8 := c_3^{s_{r_1}} w^{-s_\xi}$. Obviously, the distributions of generated $(s_x, s_{t_1}, s_{t_2}, s_{\zeta_1}, s_{\zeta_2}, s_\xi)$ and $(r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8)$ are identical to the output of the prover. Finally, the simulator outputs $(c_1, c_2, c_3, c_4, r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8, c, s_x, s_{t_1}, s_{t_2}, s_{\zeta_1}, s_{\zeta_2}, s_\xi)$, whose distribution is identical to the transcripts of the protocol. That is, the transcripts of the protocol can be simulated. ■

Lemma 3. *There is an extractor for the protocol.*

Proof: We assume that an extractor can rewind the protocol, and submit two different values c and c' to the prover. Since the prover answers c and c' with the same $(t_1, t_2, r_x, r_{t_1}, r_{t_2}, r_{\zeta_1}, r_{\zeta_2}, r_\xi)$, the extractor can obtain two transcripts $(c_1, c_2, c_3, c_4, r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8, c, s_x, s_{t_1}, s_{t_2}, s_{\zeta_1}, s_{\zeta_2}, s_\xi)$ and $(c_1, c_2, c_3, c_4, r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8, c', s'_x, s'_{t_1}, s'_{t_2}, s'_{\zeta_1}, s'_{\zeta_2}, s'_\xi)$. Let $\Delta c := c - c'$, $\Delta s_x := s_x - s'_x$, $\Delta s_{t_1} := s_{t_1} - s'_{t_1}$, $\Delta s_{t_2} := s_{t_2} - s'_{t_2}$, $\Delta s_{\zeta_1} := s_{\zeta_1} - s'_{\zeta_1}$, $\Delta s_{\zeta_2} := s_{\zeta_2} - s'_{\zeta_2}$, and , $\Delta s_\xi := s_\xi - s'_\xi$. Finally, the extractor outputs

$$\tilde{t}_1 := \frac{\Delta s_{t_1}}{\Delta c}, \quad \tilde{t}_2 := \frac{\Delta s_{t_2}}{\Delta c}, \quad \tilde{x} := \frac{\Delta s_x}{\Delta c}, \quad \text{and} \quad \tilde{y} := c_1^{1/\tilde{t}_1}.$$

Since $e(\tilde{y}, vh^{\tilde{x}}) = e(g, h)$, $(\tilde{x}, \tilde{y})$ is a solution of SDH problem, which proves this lemma. ■

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2019.2925800, IEEE Transactions on Dependable and Secure Computing

Theorem 4. *The protocol proposed in this section is an honest-verifier zero-knowledge proof of knowledge of an SDH pair.*

This theorem can be directly obtained from Lemma 1, Lemma 2, and Lemma 3.

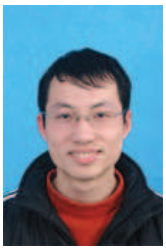
Let $H_Z : \{0,1\}^* \rightarrow \mathbb{Z}_p^*$ be a collision-resistant hash function, and $m \in \{0,1\}^*$ be a message. Then, we can obtain a secure signature scheme in the random oracle model via the Fiat-Shamir heuristic [36] from the proposed protocol. The signing algorithm and verification algorithm of the signature scheme are exactly the same with those in the proposed PBS scheme in Section III-D. Now, we are able to prove Theorem 2. That is, the proposed PBS scheme guarantees the anonymity under the random oracle model if the CDH problem is hard in $\mathbb{G}$, and the DBDH problem is hard.

Proof: Since the proposed signature scheme is converted from a zero-knowledge proof protocol via the Fiat-Shamir heuristic under the random oracle model, $(c, s_x, s_{t_1}, s_{t_2}, s_{\zeta_1}, s_{\zeta_2}, s_\xi)$ does not contain any information. Therefore, we focus on (c_1, c_2, c_3, c_4) in σ.

Recall that $c_1 := y_{id_\kappa}^{t_1}$, $c_2 := h^{t_2}$, $c_3 := w^{t_2}$, and $c_4 := C_{id_\kappa} A^{t_2}$, where y_{id_κ} is part of the user key uk_{id_κ} and $C_{id_\kappa} = e(y_{id_\kappa}, v)$ is the corresponding certificate. From [32], the probability that the adversary distinguishes (c_2, c_3, c_4) from (c_2, c_3, Z) is negligible greater than 1/2 if the CDH problem is hard in $\mathbb{G}$ and the DBDH problem is hard, where Z is a random element in $\mathbb{G}_T$. Therefore, for each signing oracle query, we can choose a random $y \in \mathbb{G}$ and two random number $t_1, t_2 \in \mathbb{Z}_p^*$, and compute $(y^{t_1}, h^{t_2}, w^{t_2}, e(y,v)A^{t_2})$ which is indistinguishable from the truly (c_1, c_2, c_3, c_4) generated from y_{id_κ}. Thus, the probability that the adversary wins is negligible greater than 1/2, which implies that the proposed PBS scheme guarantees anonymity. ■

Then, we prove Theorem 3. That is, the proposed PBS scheme guarantees the traceability under the random oracle model if the Strong Diffie-Hellman (SDH) problem is hard in $\mathbb{G}$. Note that traceability implies unforgeability which is the basic requirement for any digital signature scheme. To implement the traditional unforgeability, we only need to disable the capability of querying the key generation oracle in Definition 6.

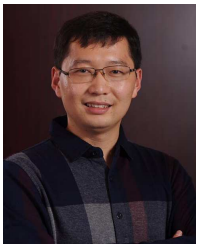
Proof: The strong Diffie-Hellman problem is that with the input of $(g, h, h^\alpha, \ldots, h^{\alpha^q}) \in \mathbb{G}^{q+2}$, any PPT algorithm could not output $(x^*, g^{1/(\alpha+x^*)})$ except for a negligible probability, where q is a system parameter, and x^* is selected by the algorithm. We prove that we can extract a pair of $(x^*, g^{1/(\alpha+x^*)})$ from the signature σ^*, i.e., solve the SDH problem, if we view the hash function H_Z as a random oracle. This process is trivial from Lemma 3, in which the extractor can compute $(x^*, g^{1/(\alpha+x^*)})$ by rewinding the zero-knowledge proof protocol. Since the SDH problem is hard, which means that the adversary cannot output a valid message-signature pair (m^*, σ^*) with unattained SDH pair (i.e., with unrevoked user key) except for negligible probability. Thus, the proposed PBS scheme guarantees traceability. ■



Kun He is a postdoctor of Wuhan University. He received a Ph.D. in computer science from the computer school, Wuhan University. His research interests include cryptography, network security, mobile computing, and cloud computing.

Jing Chen received the Ph.D. degree in computer science from Huazhong University of Science and Technology, Wuhan. He worked as an associate professor from 2010. His research interests in computer science are in the areas of network security, cloud security. He is the Chief Investigator of several projects in network and system security, funded by the National Natural Science Foundation of China (NSFC). He has published more than 60 research papers in many international journals and conferences, such as IEEE Transactions on Parallel and Distributed System, International Journal of Parallel and Distributed System, INFOCOM, SECON, TrustCom, NSS. He acts as a reviewer for many Journals and conferences, such as IEEE Transactions on Wireless Communication, IEEE Transactions on Industrial Informatics, Computer Communications, and GLOBCOM.

Quan Yuan is an assistant professor in the Department of Math and Computer Science at the University of Texas-Permian Basin, TX, USA. His research interests include mobile computing, routing protocols, peer-to-peer computing, parallel and distributed systems, and computer networks. He has published more than 30 research papers in many international journals and conferences, such as IEEE Transactions on Parallel and Distributed Systems, INFOCOM, MobiHoc, SECON, and TrustCom.



Shouling Ji is a ZJU 100-Young Professor in the College of Computer Science and Technology at Zhejiang University and a Research Faculty in the School of Electrical and Computer Engineering at Georgia Institute of Technology. He received a Ph.D. in Electrical and Computer Engineering from Georgia Institute of Technology and a Ph.D. in Computer Science from Georgia State University. His current research interests include Big Data Security and Privacy, Big Data Driven Security and Privacy, and Adversarial Learning. He is a member of IEEE and ACM and was the Membership Chair of the IEEE Student Branch at Georgia State (2012-2013).

Debiao He received the PhD degree in applied mathematics from the School of Mathematics and Statistics, Wuhan University, in 2009. He is currently a professor in School of Cyber Science and Engineering, Wuhan University. His main research interests include cryptography and information security, in particular, and cryptographic protocols.

Ruiying Du received the BS, MS, PH.D degrees in computer science in 1987, 1994 and 2008, from Wuhan University, Wuhan, China. She is a professor at School of Cyber Science and Engineering, Wuhan University. Her research interests include network security, wireless network, cloud computing and mobile computing. She has published more than 80 research papers in many international journals and conferences, such as IEEE Transactions on Parallel and Distributed System, International Journal of Parallel and Distributed System, INFOCOM, SECON, TrustCom, NSS.