Lightweight and Privacy-Preserving Delegatable Proofs of Storage*

Jia Xu[1], Anjia Yang[1,2], Jianying Zhou[1], and Duncan S. Wong[2]

Institute for Infocomm Research, Singapore[1],
{xuj,jyzhou}@i2r.a-star.edu.sg
City University of Hong Kong, China[2],
ayang3-c@my.cityu.edu.hk, duncan@cityu.edu.hk

Abstract. Proofs of storage (POR or PDP) is a cryptographic tool, which enables data owner or third party auditor to audit integrity of data stored remotely in a cloud storage server, without keeping a local copy of data or downloading data back during auditing. We observe that all existing publicly verifiable POS schemes suffer from a serious drawback: It is extremely slow to compute authentication tags for all data blocks, due to many expensive group exponentiation operations. Surprisingly, it is even much slower than typical network uploading speed, and becomes the bottleneck of the setup phase of the POS scheme. We propose a new variant formulation called "Delegatable Proofs of Storage". In this new relaxed formulation, we are able to construct a POS scheme, which on one side is as efficient as private key POS schemes, and on the other side can support third party auditor and can switch auditors at any time, close to the functionalities of publicly verifiable POS schemes. Compared to traditional publicly verifiable POS schemes, we speed up the tag generation process by at least several hundred times, without sacrificing efficiency in any other aspect. Like many existing schemes, we can also speed up our tag generation process by N times using N CPU cores in parallel. We prove that our scheme is sound under Bilinear Strong Diffie-Hellman Assumption, and it is privacy preserving against auditor under Discrete Log Assumption. Both proofs are given in standard model.

1 Introduction

Since POR [18] and PDP [4] are proposed in 2007, a lot of effort of research community is devoted to constructing proofs of storage schemes with more advanced features. The new features include, public key verifiability [24], supporting dynamic operations [10, 15, 28] (i.e. inserting/deleting/editing a data block), supporting multiple cloud servers [12], supporting data sharing [30], etc. We look back into the very first feature—public verifiability, and observe that all existing publicly verifiable POS schemes suffer from serious drawbacks: (1) Merkle Hash Tree based method is not disk IO-efficient and not even a sub-linear memory authenticator [19]: Every bit of the file has to be accessed by the cloud storage server in each remote integrity auditing process. (2) By our knowledge, all other publicly verifiable POS schemes employ expensive operation (e.g. group exponentiation) to generate authentication tags for data blocks. As a result, it is prohibitively expensive to generate authentication tags

* Anjia Yang contributes to this work when he takes his internship in Institute for Infocomm Research, Singapore.

for medium or large size data file. For example, Wang *et al.* [31] achieves throughput of data pre-processing at speed 17.2KB/s with an Intel Core 2 1.86GHz workstation CPU, which means it will take about 17 hours to generate authentication tags for a 1GB file. Even if the user has a CPU with 8 cores, it still requires more than 2 hours' heavy computation. Such amount heavy computation is not appropriate for a laptop, not to mention tablet computer (e.g. iPad) or smart phone.

The benefit of publicly verifiable POS schemes is that, anyone with the public key can audit the integrity of data in cloud storage, to relieve the burden from the data owner. However, one should not allow any third party to audit his/her data at their will, and delegation of auditing task has to be in a controlled and organized manner. Otherwise, we have no guarantee to prevent extreme cases: (1) on one hand, some data file could attract too much attention from public, and are audited unnecessarily too frequently by the public, which might actually result in distributed denial of service attack against the cloud storage server; (2) on the other hand, some unpopular data file may be audited by the public too rarely, so that the possible data loss event might be detected and alerted to the data owner too late and no effective countermeasure can be done to reduce the damage.

Instead, the data owner could delegate the auditing task to some semi-trusted third party auditor, and this auditor is *fully* responsible to audit the data stored in cloud storage on behalf of the data owner, in a controlled way, with proper frequency. We call such an exclusive auditor as *Owner-Delegated-Auditor* or ODA for short. In real world applications, ODA could be another server that provides free or paid auditing service to many cloud users.

To address the issues of existing publicly verifiable POS schemes, we propose a *hybrid* POS scheme, which on one hand supports delegation of data auditing task, like publicly verifiable POS schemes, and on the other hand is as efficient as a privately verifiable POS scheme.

1.1 Overview of Our Scheme

Our scheme generates two pairs of public/private keys: (pk, sk) and (vpk, vsk). The verification public/private key pair (vpk, vsk) is delegated to the ODA and will be updated (or re-randomized) once an ODA is revoked and a new ODA is chosen. The master public/private key pair (pk, sk) will always keep unchanged. Our scheme proposes a novel linear homomorphic authentication tag function [5], which is extremely lightweight, without any expensive operations (e.g. group exponentiation or bilinear map). Our tag function generates two tags (σ_i, t_i) for each data block, where tag σ_i is generated in a similar way as Shacham and Waters' scheme [24], and tag t_i is generated in a completely new way. The size of all tags $\{(\sigma_i, t_i)\}$ is $2/m$-fraction of the whole file size, where system parameter m can take any positive integer value and typical value is from a hundred to a thousand. ODA is able to verify data integrity remotely by checking consistency among the data blocks and both tags $\{(\sigma_i, t_i)\}$ that are stored in the cloud storage server, using the verification secret key vsk. The data owner retains the capability to verify data integrity by checking consistency between the data blocks and tags $\{\sigma_i\}$, using the master secret key sk, in a way similar to Shacham and Waters' scheme [24]. When an ODA is revoked and replaced by a new ODA, all authentication tags $\{t_i\}$ will be updated (or re-randomized) together with the verification key pair (vpk, vsk), but tags $\{\sigma_i\}$ will keep unchanged.

We combine our new linear homomorphic authentication tag function with existing techniques, in order to reduce communication cost and achieve privacy protection. We customize the polynomial commitment scheme proposed by Kate *et al.* [2] and integrate it into our scheme, in order to reduce proof size from $O(m)$ to $O(1)$. We also customize the "generalized Okamoto identification scheme" [1,20] and integrate it in our scheme to prevent information leakage to the ODA during the auditing process.

We emphasize that: (1) The naive method that runs an existing private key POS scheme on an input file twice to generate two key pairs and two authentication tags, is unsatisfactory. This naive method does not support efficient updating of verification key (i.e. switching auditors), and the data owner has to download the whole data file to refresh the verification key pair, for every time that an auditor is revoked. (2) If the data file is encrypted using some semantic-secure cipher, and the POS scheme is applied over the ciphertext, then the privacy-preserving feature of the POS is not necessary. However, in some application scenario, the semi-trusted cloud server has to access the data file, in order to provide more kinds of services (e.g. analyzing/querying the data) rather than pure storage backup service. It is not storage efficient to require cloud server to keep both ciphertext copy and plaintext copy, where the ciphertext copy is audited by third party auditor and the plaintext copy is used to provide other services.

Table 1. Performance Comparison of Proofs of Storage (POR,PDP) Schemes. In this table, publicly verifiable POS schemes appear above our scheme, and privately verifiable POS schemes appear below our scheme.

Scheme	Computation (Data Pre-process)			Communication bits		Storage Overhead (Server)	Computation (Verifier)				Computation (Prover)			
	exp.	*mul.*	*add.*	Challenge	Response		*exp.*	*mul.*	*pair.*	*add.*	*exp.*	*mul.*	*pair.*	*add.*
[3,4]	$2\frac{\lvert F\rvert}{m\lambda}$	$\frac{\lvert F\rvert}{m\lambda}$	0	$\log \ell + 2\kappa$	2λ	$\frac{\lvert F\rvert}{m}$	ℓ	ℓ	0	0	ℓ	2ℓ	0	ℓ
[24,25]	$\frac{\lvert F\rvert}{\lambda} + \frac{\lvert F\rvert}{m\lambda}$	$\frac{\lvert F\rvert}{\lambda}$	0	$\ell\lambda + \ell\log(\frac{\lvert F\rvert}{m\lambda})$	$(m+1)\lambda$	$\frac{\lvert F\rvert}{m}$	$\ell+m$	$\ell+m$	2	0	ℓ	$m\ell+\ell$	0	$m\ell$
[36,37]	$2\frac{\lvert F\rvert}{\lambda}$	$\frac{\lvert F\rvert}{\lambda}$	0	$\ell\lambda + \ell\log(\frac{\lvert F\rvert}{m\lambda})$	$(\ell+3)\lambda + \ell(\lceil\log(\frac{\lvert F\rvert}{m\lambda})\rceil - 1)\ \lvert h\rvert$	$\lvert F\rvert$	ℓ	ℓ	4	0	ℓ	2ℓ	0	ℓ
[35]	$2\frac{\lvert F\rvert}{\lambda}$	$\frac{\lvert F\rvert}{\lambda}$	0	$\ell\lambda + \ell\log(\frac{\lvert F\rvert}{m\lambda})$	3λ	$\lvert F\rvert$	ℓ	ℓ	2	0	ℓ	2ℓ	0	ℓ
[31]†	$\frac{\lvert F\rvert}{\lambda} + \frac{\lvert F\rvert}{m\lambda}$	$\frac{\lvert F\rvert}{\lambda}$	0	$\ell\lambda + \ell\log(\frac{\lvert F\rvert}{m\lambda})$	$(2m+1)\lambda$	$\frac{\lvert F\rvert}{m}$	$\ell+m$	$\ell+2m$	2	0	$\ell+m$	$m\ell+\ell$	1	$m\ell$
[44]	$\frac{\lvert F\rvert}{m\lambda} + m$	$2\frac{\lvert F\rvert}{\lambda} + m$	$\frac{\lvert F\rvert}{\lambda} + m$	$\ell\lambda + \ell\log(\frac{\lvert F\rvert}{m\lambda})$	$(m+3)\lambda$	$\frac{\lvert F\rvert}{m}$	$\ell+m$	$\ell+m$	3	0	$\ell+m$	$m\ell+2\ell+2m$	1	$m\ell$
[17]	$\frac{\lvert F\rvert}{m\lambda}$	0	0	$\lambda + k$	λ	0††	$\frac{\lvert F\rvert}{m\lambda}$	$\frac{\lvert F\rvert}{m\lambda}$	0	0	$\log(\frac{\lvert F\rvert}{m\lambda}) + m$	$\frac{\lvert F\rvert}{m\lambda}$	0	$\frac{\lvert F\rvert}{m\lambda}$
[40]	$\frac{\lvert F\rvert}{\lambda} + \frac{\lvert F\rvert}{m\lambda} + m$	$\frac{\lvert F\rvert}{\lambda}$	0	$(\ell+1)\lambda + \ell\log(\frac{\lvert F\rvert}{m\lambda})$	2λ	$\frac{\lvert F\rvert}{m}$	ℓ	2ℓ	2	0	$\ell+m$	$m\ell+m+\ell$	m	$m\ell$
[41]	$\frac{\lvert F\rvert}{\lambda} + \frac{2\lvert F\rvert}{m\lambda}$	$\frac{\lvert F\rvert}{\lambda}$	0	$2\lambda + \ell\log(\frac{\lvert F\rvert}{m\lambda})$	3λ	$\frac{\lvert F\rvert}{m}$	ℓ	2ℓ	4	0	$\ell+m$	$m\ell+m+\ell$	0	$m\ell$
Our Scheme	0	$\frac{2\lvert F\rvert}{\lambda}$	$\frac{2\lvert F\rvert}{\lambda}$	$5\lambda + 280$	10λ	$\frac{2\lvert F\rvert}{m}$	6	ℓ	7	ℓ	$5m$	$m\ell+2\ell+6m$	0	$m\ell+2\ell+2m$
[24,25]†††	0	$\frac{\lvert F\rvert}{\lambda}$	$\frac{\lvert F\rvert}{\lambda}$	$\ell\lambda + \ell\log(\frac{\lvert F\rvert}{m\lambda})$	$(m+1)\lambda$	$\frac{\lvert F\rvert}{m}$	0	$\ell+m$	0	$\ell+m$	0	$m\ell+\ell$	0	$m\ell+\ell$

† [31] is a journal version of [35], and the main scheme is almost the same as [35]. We now consider the one that divides each data block into m sectors.

†† In Hao *et al.*'s paper [17], the authentication tags are stored at both the client and the verifier side, rather than the server side.

††† The private POS scheme of Shacham and Waters [24,25]. Notice that the public POS scheme of [24,25] also appears in this table.

κ, k are system parameters, $|h|$ is the length of a hash output. $|F|$ is the data file size. λ is group element size. m is the number of sectors in each data block. ℓ is the sampling size.

1.2 Contributions

Our contributions can be summarized as below:

- We propose a new formulation called "Delegatable Proofs of Storage" (DPOS), as a relaxed version of public key POS. We design a new scheme under this formulation. Our scheme is as efficient as private key POS: The tag generation throughput is slightly larger than 10MB/s per CPU core. On the other side, our scheme allows delegation of auditing task to a semi-trusted third party auditor, and also supports switching auditor at any time, like a publicly verifiable POS scheme. We compare the performance complexity of our scheme with the state of arts in Table 1, and experiment shows the tag generation speed of our scheme is more than hundred times faster than the state of art of publicly verifiable POS schemes.
- We prove that our scheme is sound under Bilinear Strong Diffie-Hellman Assumption, and privacy preserving under Discrete Log Assumption, in standard model.

2 Related Work

Recently, much growing attention has been paid to integrity check of data stored at untrusted servers [3–6, 8, 9, 11–15, 17, 18, 22–27, 29–44]. In CCS'07, Ateniese *et al.* [4] defined the *provable data possession* (PDP) model and proposed the first publicly verifiable PDP scheme. Their scheme used RSA-based homomorphic authenticators and sampled a number of data blocks rather than the whole data file to audit the outsourced data, which can reduce the communication complexity significantly. However, in their scheme, a linear combination of sampled blocks are exposed to the third party auditor (TPA) at each auditing, which may leak the data information to the TPA. At the meantime, Juels and Kaliski [18] described a similar but stronger model: *proof of retrievability* (POR), which enables auditing of not only the integrity but also the retrievability of remote data files by employing spot-checking and error-correcting codes. Nevertheless, their proposed scheme allows for only a bounded number of auditing services and does not support public verification.

Shacham and Waters [24, 25] presented a publicly verifiable POR scheme and gave a comprehensive proof of security under the POR model [18]. Similar to [4], their scheme utilized homomorphic authenticators built from BLS signatures [7]. Subsequently, Zeng *et al.* [42], Wang *et al.* [36, 37] proposed some similar constructions for publicly verifiable remote data integrity check, which adopted the BLS based homomorphic authenticators. With the same reason as [4], these protocols do not support data privacy. In [31, 35], Wang *et al.* extended their scheme to be privacy preserving. The idea is to mask the linear combination of sampled blocks in the server's response with some random value. With the similar masking technique, Zhu *et al.* [44] introduced another privacy-preserving public auditing scheme. Later, Hao *et al.* [17] and Yang *et al.* [40] proposed two privacy-preserving public auditing schemes without applying the masking technique. Yuan *et al.* [41] gave a POR scheme with public verifiability and constant communication cost.

However, all of the publicly verifiable PDP/POR protocols require the data owner to do a large amount of computation of exponentiation on big numbers for generating

the authentication tags upon preprocessing the data file. This makes these schemes impractical for file of large size.

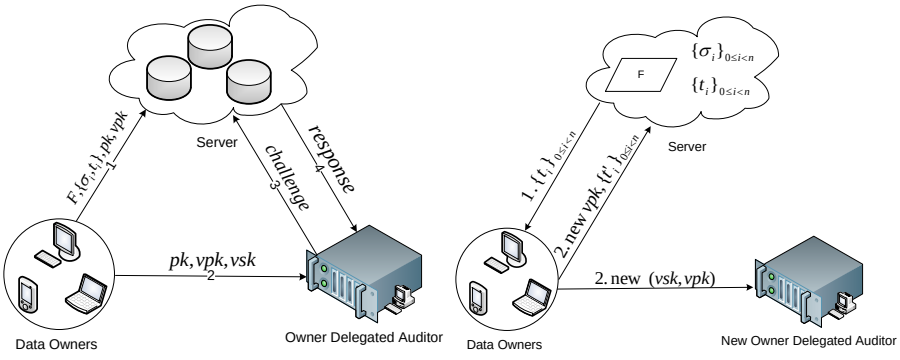
3 Formulation

We propose a formulation called DPOS, based on existing POR and PDP formulations. We provide the system model in Sec 3.1 and the trust model in Sec 3.2. We will defer the security definition to Sec 5, where the security analysis of our scheme will be provided.

3.1 System Model

Definition 1 *A* Delegatable Proofs of Storage *(*DPOS*) scheme consists of three algorithms (*KeyGen, Tag, UpdVK*), and a pair of interactive algorithms* $\langle \mathsf{P}, \mathsf{V} \rangle$*, where each algorithm is described as below*

- $\mathsf{KeyGen}(1^\lambda) \rightarrow (pk, sk, vpk, vsk)$: *Given a security parameter* 1^λ*, this randomized key generating algorithm generates a pair of public/private master keys* (pk, sk) *and a pair of public/private verification keys* (vpk, vsk)*.*
- $\mathsf{Tag}(sk, vsk, F) \rightarrow (\mathtt{Param}_F, \{(\sigma_i, t_i)\})$: *Given the master secret key* sk*, the verification secret key* vsk*, and a data file* F *as input, the tagging algorithm generates a file parameter* $\mathtt{Param}_F$ *and authentication tags* $\{(\sigma_i, t_i)\}$*, where a unique file identifier* $\mathtt{id}_F$ *is a part of* $\mathtt{Param}_F$*.*
- $\mathsf{UpdVK}(vpk, vsk, \{t_i\}) \rightarrow (vpk', vsk', \{t'_i\})$: *Given the current verification key pair* (vpk, vsk) *and the current authentication tags* $\{t_i\}$*, this updating algorithm generates the new verification key pair* (vpk', vsk') *and the new authentication tags* $\{t'_i\}$*.*
- $\langle \mathsf{P}(pk, vpk, \{\vec{\boldsymbol{F}}_i\}), \mathsf{V}(vsk, vpk, pk, \mathtt{Param}_F) \rangle \rightarrow \mathtt{Accept}$ *or* $\mathtt{Reject}$: *The prover algorithm* P *interacts with the verifier algorithm* V *to output a decision bit* $\mathtt{Accept}$ *or* $\mathtt{Reject}$*, where the input of* P *consists of the master public key* pk*, the verification public key* vpk*, and file blocks* $\{\vec{\boldsymbol{F}}_i\}$*, and the input of* V *consists of the verification secret key* vsk*, verification public key* vpk*, master public key* pk*, and file information* $\mathtt{Param}_F$*.*



(a) The framework of DPOS (b) Updating Verification key (vpk, vsk)

Fig. 1. Illustration of system model of DPOS.

A DPOS system is described as below and illustrated in Fig 1(a) and Fig 1(b).

Definition 2 *A* DPOS *system among three parties—data owner, cloud storage server and auditor, can be implemented by running a* DPOS *scheme (*KeyGen, Tag, UpdVK,$\langle \mathsf{P}, \mathsf{V} \rangle$*)*

in the following three phases, where the setup phase will execute at the very beginning, for only once (for one file); the proof phase and revoke phase can execute for multiple times and in any (interleaved) order.

Setup phase *The data owner runs the key generating algorithm* $\mathsf{KeyGen}(1^\lambda)$ *for only once, to generate the master key pair* (pk, sk) *and the verification key pair* (vpk, vsk)*. For every input data file, the data owner may choose to apply some error erasure code [21] on this file, and runs the tagging algorithm* Tag *over the (erasure encoded) file, to generate authentication tags* $\{(\sigma_i, t_i)\}$ *and file parameter* $\mathtt{Param}_F$*. At the end of setup phase, the data owner sends the (erasure encoded) file* F*, all authentication tags* $\{(\sigma_i, t_i)\}$*, file parameter* $\mathtt{Param}_F$*, and public keys* (pk, vpk) *to the cloud storage server. The data owner also chooses an exclusive third party auditor, called* Owner-Delegated-Auditor *(ODA, for short), and delegates the verification key pair* (vpk, vsk) *and file parameter* $\mathtt{Param}_F$ *to the ODA. After that, the data owner may keep only keys* (pk, sk, vpk, vsk) *and file parameter* $\mathtt{Param}_F$ *in local storage, and delete everything else from local storage.*

Proof phase *The proof phase consists of multiple proof sessions. In each proof session, the ODA, who runs algorithm* V*, interacts with the cloud storage server, who runs algorithm* P*, to audit the integrity of data owner's file, on behalf of the data owner. Therefore, ODA is also called* verifier *and cloud storage server is also called* prover*.*

Revoke phase *In the revoke phase, the data owner downloads all tags* $\{t_i\}$ *from cloud storage server, revokes the current verification key pair, and generates a fresh verification key pair and new tags* $\{t'_i\}$*. The data owner also chooses a new ODA, and delegates the new verification key pair to this new ODA, and sends the updated tags* $\{t'_i\}$ *to the cloud storage server to replace the old tags* $\{t_i\}$*.*

A DPOS scheme (KeyGen, Tag, UpdVK, $\langle \mathsf{P}, \mathsf{V} \rangle$) is *complete*, if for all keys (pk,sk, vpk,vsk) output by KeyGen, for all files F, if all parties follow our scheme exactly and the data stored in cloud storage is intact, then interactive proof algorithms $\langle \mathsf{P}, \mathsf{V} \rangle$ will always output `Accept`.

3.2 Trust Model

In this paper, we aim to protect data integrity and privacy of data owner's file. The data owner is fully trusted, and the cloud storage server and ODA are semi-trusted in different sense: (1) The cloud storage server is trusted in data privacy (We assume the server has to access plaintext to provide additional services to the data owner), and is *not* trusted in maintaining data integrity (e.g. the server might delete some rarely accessed data for economic benefits, or hide the data corruption events caused by server failures or attacks to maintain reputation). (2) Before he/she is revoked, the ODA is trusted in performing the delegated auditing task and protecting his/her verification secret key securely, but is *not* trusted in data privacy. A revoked ODA could be potentially malicious and might surrender his/her verification secret key to the cloud storage server.

We assume that all communication among the data owner, the cloud storage server and ODA is via some secure channel (i.e. channel privacy and integrity are protected). Framing attack among these three parties can be dealt with existing technique and is out of scope of this paper.

4 Our Proposed Scheme

4.1 Preliminaries

Let $\mathbb{G}$ and $\mathbb{G}_T$ be two multiplicative cyclic groups of prime order p. Let g be a randomly chosen generator of group $\mathbb{G}$. A bilinear map is a map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ with the following properties:

(1) Bilinearity: $e(u^a, v^b) = e(u, v)^{ab}$ for all $u, v \in \mathbb{G}$ and $a, b \in \mathbb{Z}_p$.
(2) Non-degeneracy: If g is a generator of $\mathbb{G}$, then $e(g, g)$ is a generator of $\mathbb{G}_T$, i.e. $e(g, g) \neq 1$.
(3) Computable: There exists an efficient algorithm to compute $e(u, v)$ for all $u, v \in \mathbb{G}$.

In the rest of this paper, the term "bilinear map" will refer to the non-degenerate and efficiently computable bilinear map only.

For vector $\vec{\boldsymbol{a}} = (a_1, \ldots, a_m)$ and $\vec{\boldsymbol{b}} = (b_1, \ldots, b_m)$, the notation $\left\langle \vec{\boldsymbol{a}},\ \vec{\boldsymbol{b}} \right\rangle \stackrel{\text{def}}{=} \sum_{j=1}^{m} a_j b_j$ denotes the dot product (a.k.a inner product) of the two vectors $\vec{\boldsymbol{a}}$ and $\vec{\boldsymbol{b}}$. For vector $\vec{\boldsymbol{v}} = (v_0, \ldots, v_{m-1})$ the notation $\mathsf{Poly}_{\vec{\boldsymbol{v}}}(x) \stackrel{\text{def}}{=} \sum_{j=0}^{m-1} v_j x^j$ denotes the polynomial in variable x with $\vec{\boldsymbol{v}}$ being the coefficient vector.

4.2 Construction of the Proposed DPOS Scheme

We define our DPOS scheme $(\mathsf{KeyGen}, \mathsf{Tag}, \mathsf{UpdVK}, \langle \mathsf{P}, \mathsf{V} \rangle)$ as below, and these algorithms will run in the way as specified in Definition 2 (on page 5).

$\mathsf{KeyGen}(1^\lambda) \to (\boldsymbol{pk}, \boldsymbol{sk}, \boldsymbol{vpk}, \boldsymbol{vsk})$ `Executed by Data Owner`

Choose at random a λ-bits prime p and a bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$, where $\mathbb{G}$ and $\mathbb{G}_T$ are both multiplicative cyclic groups of prime order p. Choose at random a generator $g \in \mathbb{G}$. Choose at random $\alpha, \beta, \gamma, \rho \in_R \mathbb{Z}_p^*$. For each $j \in [1, m]$, define $\alpha_j := \alpha^j \mod p$ and $\beta_j := \beta^j \mod p$, and compute $g_j := g^{\alpha_j}$, $h_j := g^{\rho \cdot \beta_j}$. Let $g_0 = g^{\alpha^0} = g$, $h_0 = g^{\rho \cdot \beta^0} = g^\rho$, vector $\vec{\boldsymbol{\alpha}} := (\alpha_1, \alpha_2, \ldots, \alpha_m)$, and $\vec{\boldsymbol{\beta}} := (\beta_1, \beta_2, \ldots, \beta_m)$, Choose two random seeds s_0, s_1 for pseudorandom function $\mathcal{PRF}_{\mathtt{seed}} : \{0,1\}^\lambda \times \{0,1\}^{\mathbb{N}} \to \mathbb{Z}_p$.

The secret key is $sk = (\alpha, \beta, s_0)$ and the public key is $pk = (g_0, g_1, \ldots, g_m)$. The verification secret key is $vsk = (\rho, \gamma, s_1)$ and the verification public key is $vpk = (h_0, h_1, \ldots, h_m)$.

$\mathsf{Tag}(\boldsymbol{sk}, \boldsymbol{vsk}, \boldsymbol{F}) \to (\mathtt{Param}_{\boldsymbol{F}}, \{(\boldsymbol{\sigma_i}, \boldsymbol{t_i})\})$ `Executed by Data Owner`

Split file[1] F into n blocks, where each block is a vector of m elements from $\mathbb{Z}_p$: $\{\vec{\boldsymbol{F}}_i = (F_{i,0}, \ldots, F_{i,m-1}) \in \mathbb{Z}_p^m\}_{i \in [0, n-1]}$. Choose a unique identifier $\mathtt{id}_F \in \{0,1\}^\lambda$. Define a customized[2] pseudorandom function w.r.t. the file F: $\mathsf{PRF}_s(i) = \mathcal{PRF}_s(\mathtt{id}_F, i)$.

[1] Possibly, the input has been encoded by the data owner using some error erasure code.
[2] With such a customized function PRF, the input $\mathtt{id}_F$ will become *implicit* and this will simplify our expression.

For each block $\vec{\boldsymbol{F}}_{\boldsymbol{i}}$, $0 \le i \le n-1$, compute

$$\sigma_i = \left\langle \vec{\boldsymbol{\alpha}},\ \vec{\boldsymbol{F}}_{\boldsymbol{i}} \right\rangle + \mathsf{PRF}_{s_0}(i) = \alpha \cdot \mathsf{Poly}_{\vec{\boldsymbol{F}}_{\boldsymbol{i}}}(\alpha) + \mathsf{PRF}_{s_0}(i) \mod p \tag{1}$$

$$t_i = \rho \left\langle \vec{\boldsymbol{\beta}},\ \vec{\boldsymbol{F}}_{\boldsymbol{i}} \right\rangle + \gamma \mathsf{PRF}_{s_0}(i) + \mathsf{PRF}_{s_1}(i) \mod p \tag{2}$$

$$= \rho \cdot \beta \mathsf{Poly}_{\vec{\boldsymbol{F}}_{\boldsymbol{i}}}(\beta) + \gamma \mathsf{PRF}_{s_0}(i) + \mathsf{PRF}_{s_1}(i) \mod p \tag{3}$$

The general information of F is $\mathtt{Param}_F := (\mathtt{id}_F, n)$.

$\mathsf{UpdVK}(\boldsymbol{vpk}, \boldsymbol{vsk}, \{\boldsymbol{t_i}\}_{i\in[0,n-1]}) \rightarrow (\boldsymbol{vpk'}, \boldsymbol{vsk'}, \{\boldsymbol{t'_i}\}_{i\in[0,n-1]})$

`Executed by Data Owner`

Parse vpk as $(h_0, \ldots, h_m)$ and vsk as (ρ, γ, s_1). Verify the integrity of all tags $\{t_i\}$ (We will discuss how to do this verification later), and abort if the verification fails. Choose at random $\gamma' \in_R \mathbb{Z}_p^*$ and choose a random seed s'_1 for pseudorandom function PRF. For each $j \in [0, m]$, compute $h'_j := h_j^{\gamma'} = g^{(\rho\cdot\gamma')\cdot\beta_j} \in \mathbb{G}$. For each $i \in [0, n-1]$, compute a new authentication tag

$$t'_i := \gamma' \left(t_i - \mathsf{PRF}_{s_1}(i)\right) + \mathsf{PRF}_{s'_1}(i) \mod p.$$

$$= \gamma' \cdot \rho \left\langle \vec{\boldsymbol{\beta}},\ \vec{\boldsymbol{F}}_{\boldsymbol{i}} \right\rangle + (\gamma' \cdot \gamma)\, \mathsf{PRF}_{s_0}(i) + \mathsf{PRF}_{s'_1}(i) \mod p$$

The new verification public key is $vpk' := (h'_0, \ldots, h'_m)$ and the new verification secret key is $vsk' := (\gamma' \cdot \rho,\ \gamma' \cdot \gamma,\ s'_1)$.

$\langle \mathsf{P}(\boldsymbol{pk}, \boldsymbol{vpk}, \{\vec{\boldsymbol{F}}_{\boldsymbol{i}}\}_{i\in[0,n-1]}), \mathsf{V}(\boldsymbol{vsk}, \boldsymbol{vpk}, \boldsymbol{pk}, \mathtt{Param}_{\boldsymbol{F}}) \rangle$

V1: Verifier parses $\mathtt{Param}_F$ as $(\mathtt{id}_F, n)$, and sends the file identifier $\mathtt{id}_F$ to the prover to initiate a proof session.

P1: Prover locates the file F corresponding to $\mathtt{id}_F$ and will proceed on the following procedures based on file F and its associated tags $\{\sigma_i, t_i\}$. The prover chooses at random a vector $\vec{\boldsymbol{y}} = (y_1, \ldots, y_m) \in_R \mathbb{Z}_p^m$ and two elements $y_\sigma, y_t \in_R \mathbb{Z}_p$, and computes $(Y_\alpha, Y_\beta, Y_\sigma, Y_t) \in \mathbb{G}^4$ as below

$$Y_\alpha := \prod_{j=1}^{m} g_j^{y_j} = g^{\alpha \mathsf{Poly}_{\vec{\boldsymbol{y}}}(\alpha)};\ \ Y_\beta := \prod_{j=1}^{m} h_j^{y_j} = g^{\rho\cdot\beta \mathsf{Poly}_{\vec{\boldsymbol{y}}}(\beta)};\ \ Y_\sigma := g^{y_\sigma};\ \ Y_t := g^{y_t}. \tag{4}$$

Prover sends $(Y_\alpha, Y_\beta, Y_\sigma, Y_t)$ to the verifier, in order to commit the secret values $(\vec{\boldsymbol{y}}, y_\sigma, y_t)$.

V2: Verifier chooses at random $r, r_\sigma, r_t, \xi \in_R \mathbb{Z}_p^*$, and a random subset $\mathbf{C} \subset [0, n-1]$ of size ℓ. For each $i \in \mathbf{C}$, choose at random a weight $w_i \in_R \mathbb{Z}_p$. Verifier sends $(r, r_\sigma, r_t, \xi, \{(i, w_i) : i \in \mathbf{C}\})$ to the prover.

P2: Prover computes $\vec{\vec{\boldsymbol{F}}} \in \mathbb{Z}_p^m$, and $\bar{\sigma}, \bar{t} \in \mathbb{Z}_p$ as below, where the random numbers $(\vec{\boldsymbol{y}}, y_\sigma, y_t)$ are used to blind secret information and prevent data leakage to the verifier.

$$\vec{\vec{\boldsymbol{F}}} := r \sum_{i\in\mathbf{C}} w_i \vec{\boldsymbol{F}}_i + \vec{\boldsymbol{y}};\quad \bar{\sigma} := r_\sigma \sum_{i\in\mathbf{C}} w_i \sigma_i + y_\sigma;\quad \bar{t} := r_t \sum_{i\in\mathbf{C}} w_i t_i + y_t.$$

Evaluate polynomial $\mathsf{Poly}_{\vec{\vec{\boldsymbol{F}}}}(x)$ at point $x = \xi$ to obtain $z := \mathsf{Poly}_{\vec{\vec{\boldsymbol{F}}}}(\xi) \mod p$. Divide the polynomial (in variable x) $\mathsf{Poly}_{\vec{\vec{\boldsymbol{F}}}}(x) - \mathsf{Poly}_{\vec{\vec{\boldsymbol{F}}}}(\xi)$ with $(x - \xi)$ using polynomial long division, and denote the resulting quotient polynomial as $\vec{\boldsymbol{v}} = (v_0, \ldots v_{m-2})$, that is, $\mathsf{Poly}_{\vec{\boldsymbol{v}}}(x) \equiv \frac{\mathsf{Poly}_{\vec{\vec{\boldsymbol{F}}}}(x) - \mathsf{Poly}_{\vec{\vec{\boldsymbol{F}}}}(\xi)}{x-\xi}$. Compute $(\psi_\alpha, \psi_\beta, \phi_\alpha) \in \mathbb{G}^3$

as below

$$\psi_\alpha := \prod_{j=0}^{m-1} g_j^{\vec{\vec{F}}[j]} = \prod_{j=0}^{m-1} \left(g^{\alpha^j}\right)^{\vec{\vec{F}}[j]} = g^{\sum_{j=0}^{m-1} \vec{\vec{F}}[j]\alpha^j} = g^{\mathsf{Poly}_{\vec{\vec{F}}}(\alpha)} \quad \in \mathbb{G}; \tag{5}$$

$$\psi_\beta := \prod_{j=0}^{m-1} h_j^{\vec{\vec{F}}[j]} = \prod_{j=0}^{m-1} \left(g^{\rho\cdot\beta^j}\right)^{\vec{\vec{F}}[j]} = g^{\rho\cdot\sum_{j=0}^{m-1} \vec{\vec{F}}[j]\beta^j} = g^{\rho\mathsf{Poly}_{\vec{\vec{F}}}(\beta)} \quad \in \mathbb{G}; \tag{6}$$

$$\phi_\alpha := \prod_{j=0}^{m-2} g_j^{v_j} = \prod_{j=0}^{m-2} \left(g^{\alpha^j}\right)^{v_j} = g^{\sum_{j=0}^{m-2} v_j\alpha^j} = g^{\mathsf{Poly}_{\vec{v}}(\alpha)} \quad \in \mathbb{G}. \tag{7}$$

Prover sends $(z, \phi_\alpha, \bar{\sigma}, \bar{t}, \psi_\alpha, \psi_\beta)$ to the verifier.

V3: Verifier checks whether the following equalities hold:

$$e(\psi_\alpha, g) \stackrel{?}{=} e(\phi_\alpha,\ g^\alpha/g^\xi)\cdot e(g,g)^z \tag{8}$$

$$\left(\frac{e(\psi_\alpha,\ g^\alpha)}{e\left(g,\ Y_\alpha\cdot(g^{\bar{\sigma}}/Y_\sigma)^{r_\sigma^{-1}\cdot r}\right)}\right)^\gamma \stackrel{?}{=} \frac{e(\psi_\beta,\ g^\beta)}{e\left(g,\ Y_\beta\cdot\left(g^{\bar{t}}/Y_t\right)^{r_t^{-1}\cdot r}\cdot g^{-r\sum_{i\in\mathbf{C}} w_i\mathsf{PRF}_{s_1}(i)}\right)} \tag{9}$$

If all of the above equalities hold, then output `Accept`, otherwise output `Reject`.

We remark that the data owner retains the capability to audit the data integrity by checking the data blocks and tags $\{\sigma_i\}$, in a similar way as Shacham and Waters [24]. We leave the details to the full paper.

4.2.1 Completeness If all parties are honest and data are intact, we have

$$e(g,g)^{\alpha\mathsf{Poly}_{\vec{\vec{F}}}(\alpha)} = e(\psi_\alpha, g^\alpha),; \qquad \left(g^{\bar{\sigma}}/Y_\sigma\right)^{r_\sigma^{-1}} = g^{\sum_{i\in\mathbf{C}} w_i\sigma_i}, \tag{10}$$

$$e(g,g)^{\rho\beta\mathsf{Poly}_{\vec{\vec{F}}}(\beta)} = e(\psi_\beta, g^\beta), \qquad \left(g^{\bar{t}}/Y_t\right)^{r_t^{-1}} = g^{\sum_{i\in\mathbf{C}} w_i t_i}. \tag{11}$$

$$e(g,g)^{\mathsf{Poly}_{\vec{\vec{F}}}(\alpha)} = e(\phi_\alpha, g^\alpha/g^\xi)\cdot e(g,g)^z, \tag{12}$$

and the left hand side and right hand side of Eq (9) should be

$$LHS = \left(\frac{1}{g^{r\sum_{i\in\mathbf{C}} w_i\mathsf{PRF}_{s_0}(i)}}\right)^\gamma; \qquad RHS = \frac{1}{g^{r\sum_{i\in\mathbf{C}} w_i\cdot\gamma\mathsf{PRF}_{s_0}(i)}} \tag{13}$$

The detailed completeness (or correctness) proof is given in Appendix A.

4.3 Discussion

How to verify the integrity of all tag values $\{t_i\}$ in algorithm UpdVK*?* The data owner is able to verify the integrity of all tag values $\{t_i\}$ using his/her secret keys. Note that the data owner is able to replace the auditor to run the interactive proof algorithm $\langle\mathsf{P},\mathsf{V}\rangle$ with the cloud storage server, since the data owner also holds key vsk. At first, the data owner downloads all tags $\{t_i\}$. After receiving all tags t_i's, the data owner audits all data blocks at once by running $\langle\mathsf{P},\mathsf{V}\rangle$ with the cloud storage server using sampling set $\mathbf{C} = [0, n-1]$. The data owner checks an additional equality: $g^{\bar{t}} \stackrel{?}{=} g^{r_t\sum_{i\in[0,n-1]} w_i t_i}\cdot Y_t$, where r_t and w_i's are chosen by the data owner,

$\bar{t}$ and Y_t come from the server's response, and t_i's on the right hand side are downloaded from server before this interactive proof. If and only if $\langle \mathsf{P}, \mathsf{V} \rangle$ outputs `Accept` and the additional equality check succeeds, the data owner will consider that the downloaded tags $\{t_i\}$ are intact.

Alternatively, a simpler method is that: The data owner keeps tack a hash (e.g. SHA256) value of $t_0 \| t_1 \ldots \| t_{n-1}$ in local storage, and updates this hash value when executing UpdVK.

How to reduce the size of challenge $\{(i, w_i) : i \in \mathbf{C}\}$*?* Dodis *et al.* [14]'s result can be used to represent a challenge $\{(i, w_i) : i \in \mathbf{C}\}$ compactly as below:

1. Choose the subset $\mathbf{C}$ using Goldreich [16]'s (δ, ϵ)-hitter[3], where the subset $\mathbf{C}$ can be represented compactly with only $\log n + 3\log(1/\epsilon)$ bits. Assume $n < 2^{40}$ (sufficient for practical file size) and let $\epsilon = 2^{-80}$. Then $\mathbf{C}$ can be represented with 280 bits.
2. The sequence $(\ldots, w_i, \ldots)$ of ℓ weights w_i, $i \in \mathbf{C}$, ordered by increasing i, forms a simple geometric sequence $(w^1, w^2, \ldots, w^\ell)$ for some $w \in \mathbb{Z}_p^*$.

4.4 Experiment Result

We implement a prototype of our scheme in C language and using PBC [4] library. We run the prototype in a Laptop PC with a 2.5GHz Intel Core 2 Duo mobile CPU (model T9300, released in 2008). Our test files are randomly generated and of size from 128MB to 1GB. We achieve a throughput of data preprocessing at speed slightly larger than 10 megabytes per second. Detailed experiment data will be provided in the full paper.

In contrast, Atenesis *et al.* [3, 4] achieves throughput of data preprocessing at speed 0.05 megabytes per second with a 3.0GHz desktop CPU [4]. Wang *et al.* [31] achieves throughput of data pre-processing at speed 9.0KB/s and 17.2KB/s with an Intel Core 2 1.86GHz workstation CPU, when a data block is a vector of dimension $m = 1$ and $m = 10$, respectively. According to the pre-processing complexity of [31] shown in Table 1, the theoretical optimal throughput speed of [31] is twice of the speed for dimension $m = 1$, which can be approached only when m tends to $+\infty$.

Therefore, the data pre-processing in our scheme is 200 times faster than Atenesis et al. [3, 4], and 500 times faster than Wang *et al.* [31], using a single CPU core. We remark that, all of these schemes (ours and [3, 4, 31]) and some others can be speedup by N times using N CPU cores in parallel. However, typical cloud user who runs the data pre-processing task, might have CPU cores number ≤ 4.

5 Security Analysis

5.1 Security Formulation

5.1.1 Definition of Soundness Based on the existing Provable Data Possession formulation [4] and Proofs of Retrievability formulation [18, 24]. The DPOS soundness security game $\mathsf{Game}_{\mathtt{sound}}$ between a *probabilistic polynomial time* (PPT) adversary $\mathcal{A}$ (i.e. dishonest prover/cloud storage server) and a PPT challenger $\mathcal{C}$ w.r.t. a DPOS scheme $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Tag}, \mathsf{UpdVK}, \langle \mathsf{P}, \mathsf{V} \rangle)$ is as below.

[3] Goldreich [16]'s (δ, ϵ)-hitter guarantees that, for any subset $W \subset [0, n-1]$ with size $|W| \geq (1-\delta)n$, $\mathsf{Pr}[\mathbf{C} \cap W \neq \emptyset] \geq 1 - \epsilon$. Readers may refer to [14] for more details.

[4] The Pairing-Based Cryptography Library: `http://crypto.stanford.edu/pbc/`

Setup: The challenger $\mathcal{C}$ runs the key generating algorithm $\mathsf{KeyGen}(1^\lambda)$ to obtain two pair of public-private keys (pk, sk) and (vpk, vsk). The challenger $\mathcal{C}$ gives the public key (pk, vpk) to the adversary $\mathcal{A}$ and keeps the private key (sk, vsk) securely.
Learning: The adversary $\mathcal{A}$ adaptively makes polynomially many queries, where each query is one of the following:

- STORE-QUERY($\mathtt{F}$): Given a data file $\mathtt{F}$ chosen by $\mathcal{A}$, the challenger $\mathcal{C}$ runs tagging algorithm $(\mathtt{Param}_F, \{(\sigma_i, t_i)\}) \leftarrow \mathsf{Tag}(sk, vsk, \mathtt{F})$, where $\mathtt{Param}_F = (\mathtt{id}_F, n)$, and sends the data file $\mathtt{F}$, authentication tags $\{(\sigma_i, t_i)\}$, public keys (pk, vpk), and file parameter $\mathtt{Param}_F$, to $\mathcal{A}$.
- VERIFY-QUERY($\mathtt{id}_F$): Given a file identifier $\mathtt{id}_F$ chosen by $\mathcal{A}$, if $\mathtt{id}_F$ is not the (partial) output of some previous STORE-QUERY that $\mathcal{A}$ has made, ignore this query. Otherwise, the challenger $\mathcal{C}$ initiates a proof session with $\mathcal{A}$ w.r.t. the data file $\mathtt{F}$ associated to the identifier $\mathtt{id}_F$ in this way: The adversary $\mathcal{C}$, who runs the verifier algorithm $\mathsf{V}(vsk, vpk, pk, \mathtt{Param}_F)$, interacts with the adversary $\mathcal{A}$, who replaces the prover algorithm P with any PPT algorithm of its choice, and obtains an output $b \in \{\mathtt{Accept}, \mathtt{Reject}\}$. The challenger sends the decision bit b to the adversary as feedback.
- REVOKEVK-QUERY: To respond to this query, the challenger runs the verification key update algorithm to obtain a new pair of verification keys $(vpk', vsk'$, $\{t_i'\}) := \mathsf{UpdVK}(vpk, vsk, \{t_i\})$, and sends the revoked verification secret key vsk and the new verification public key vpk' and new authentication tags $\{t_i'\}$ to the adversary $\mathcal{A}$, and keeps vsk' private.

Commit: Adversary $\mathcal{A}$ chooses a file identifier $\mathtt{id}^*$ among all file identifiers it obtains from $\mathcal{C}$ by making STORE-QUERIES in **Learning** phase. Let $\mathtt{F}^*$ denote the data file associated to identifier $\mathtt{id}^*$. $\mathcal{A}$ also chooses a subset $\mathbf{C} \subset [0, n_{F^*} - 1]$, where n_{F^*} is the number of blocks in file F^*. $\mathcal{A}$ commits identifier $\mathtt{id}_F^*$ and subset $\mathbf{C}$ of indices to $\mathcal{C}$.
Retrieve: The challenger $\mathcal{C}$ initiates polynomially many proof sessions with $\mathcal{A}$ w.r.t. the data file $\mathtt{F}^*$ and subset $\mathbf{C}$, where challenger $\mathcal{C}$ plays the role of verifier and $\mathcal{A}$ plays the role of prover, as in the **Learning** phase. Let $\mathtt{transcript}_\mathcal{A}$ denote all random coins chosen by the adversary $\mathcal{A}$, and $\mathtt{response}_\mathcal{A}$ denote all responses made by the adversary $\mathcal{A}$, during these proof sessions. The challenger $\mathcal{C}$ extracts file blocks $\{\mathtt{F}_i' : i \in \mathbf{C}\}$ from $\mathcal{A}$'s storage by applying *some* PPT knowledge extractor on input $(\mathtt{transcript}_\mathcal{A}, \mathtt{response}_\mathcal{A}, pk, sk, vpk, vsk)$.

The adversary $\mathcal{A}$ wins this DPOS security game, if the challenger $\mathcal{C}$ accepts $\mathcal{A}$'s responses in these proof sessions with some noticeable probability $1/\lambda^\tau$ for some positive integer τ. The challenger $\mathcal{C}$ wins this game, if the extracted blocks $\{(i, \mathtt{F}_i') : i \in \mathbf{C}\}$ are identical to the original $\{(i, \mathtt{F}_i) : i \in \mathbf{C}\}$.
Note: Events "adversary $\mathcal{A}$ wins" and "challenger $\mathcal{C}$ wins" are not mutual exclusive.

Definition 3 (Soundness) *A* DPOS *scheme is* sound *against dishonest cloud storage server, if for any PPT adversary $\mathcal{A}$, the probability that $\mathcal{A}$ wins the above* DPOS *security game is negligibly close to the probability that $\mathcal{C}$ wins the same security game. That is*

$$\mathsf{Pr}[\mathcal{A} \textit{ wins } \mathsf{Game}_{\mathtt{sound}}] \leq \mathsf{Pr}[\mathcal{C} \textit{ wins } \mathsf{Game}_{\mathtt{sound}}] + negl(\lambda), \tag{14}$$

where λ is the security parameter.

Discussion In case of POR, the knowledge extract does not require $\mathtt{transcript}_{\mathcal{A}}$ as input.

5.1.2 Privacy-Preserving against TPA The DPOS privacy security game $\mathsf{Game}_{\mathtt{private}}$ between a *probabilistic polynomial time* (PPT) adversary $\mathcal{A}$ (i.e. dishonest verifier/auditor) and a PPT challenger $\mathcal{C}$ w.r.t. a DPOS scheme $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Tag}, \mathsf{UpdVK}, \langle \mathsf{P}, \mathsf{V} \rangle)$ is as below.

Setup: The challenger $\mathcal{C}$ runs the key generating algorithm $\mathsf{KeyGen}(1^{\lambda})$ to obtain two pair of public-private keys (pk, sk) and (vpk, vsk). The challenger $\mathcal{C}$ gives the public key (pk, vpk) and verification secret key vsk to the adversary $\mathcal{A}$ and keeps the secret key sk private. The challenger $\mathcal{C}$ chooses a random file F^* with bit-length $\geq m\lambda$, and computes $(\mathtt{Param}^*, \{(\sigma_i, t_i)\}) \leftarrow \mathsf{Tag}(sk, vsk, \mathtt{F}^*)$, where $\mathtt{Param}^* = (\mathtt{id}^*, n^*)$.

Learning: The adversary $\mathcal{A}$ adaptively makes polynomially many queries, where each query is one of the following:

- STORE-QUERY($\mathtt{F}$): Given a data file $\mathtt{F}$ chosen by $\mathcal{A}$, the challenger $\mathcal{C}$ runs tagging algorithm $(\mathtt{Param}_F, \{(\sigma_i, t_i)\}) \leftarrow \mathsf{Tag}(sk, vsk, \mathtt{F})$, where $\mathtt{Param}_F = (\mathtt{id}_F, n)$, and sends the data file $\mathtt{F}$, the authentication tags $\{(\sigma_i, t_i)\}$, public keys (pk, vpk) together with $\mathtt{Param}_F$ to $\mathcal{A}$.
- VERIFY-QUERY($\mathtt{id}_F$): Given a file identifier $\mathtt{id}_F$ chosen by $\mathcal{A}$, if $\mathtt{id}_F \neq \mathtt{id}^*$ and it is not the (partial) output of some previous STORE-QUERY that $\mathcal{A}$ has made, ignore this query. Otherwise, the adversary $\mathcal{A}$ initiates a proof session with the challenger $\mathcal{C}$ w.r.t. the data file $\mathtt{F}$ associated to the identifier $\mathtt{id}_F$ in this way: The adversary $\mathcal{A}$, who replaces the verifier algorithm $\mathsf{V}(vsk, vpk, pk, \mathtt{Param}_F)$ with any PPT algorithm of its choice, interacts with the challenger $\mathcal{C}$, who runs the prover algorithm $\mathsf{P}(pk, vpk, \{\vec{\boldsymbol{F}}_i\})$, and obtains an output $b \in \{\mathtt{Accept}, \mathtt{Reject}\}$.
- REVOKEVK-QUERY: To respond to this query, the challenger runs the verification key update algorithm to obtain a new pair of verification keys and update authentication tags t_i's for all files: $(vpk', vsk', \{t_i'\}) := \mathsf{UpdVK}(vpk, vsk, \{t_i\})$, and sends the all verification keys (vpk, vsk, vpk', vsk') to the adversary $\mathcal{A}$.

Guess: The adversary outputs a tuple (i, j, b), wins this game if $b = \vec{\boldsymbol{F}}_i^*[j]$, i.e. b equals to the j-th dimension of the i-th data block $\vec{\boldsymbol{F}}_i^*$.

Definition 4 (Privacy-Preserving) *A* DPOS *scheme is privacy-preserving against TPA, if for any PPT adversary $\mathcal{A}$ (dishonest verifier/auditor), $\mathcal{A}$ wins the above privacy security game $\mathsf{Game}_{\mathtt{private}}$ with only negligible probability. That is,*

$$\mathsf{Pr}[\mathcal{A} \textit{ wins } \mathsf{Game}_{\mathtt{private}}] \leq negl(\lambda), \tag{15}$$

where λ is the security parameter.

5.2 Security Proof

For ease of exposition, we clarify two related but distinct concepts: valid proof and genuine proof. (1) A proof is *genuine*, if it is the same as the one generated by an honest (deterministic) prover on the same query. (2) A proof is *valid*, if it is accepted by the honest verifier. In our scheme, for each query, there exists only one genuine proof, and there exist many valid proofs. To be secure, our scheme has to ensure that it is computationally hard to compute a valid but not genuine proof.

Definition 5 (m-Bilinear Strong Diffie-Hellman (m-BSDH) Assumption) *Let $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ be a bilinear map where $\mathbb{G}$ and $\mathbb{G}_T$ are both multiplicative cyclic*

groups of prime order p. Let g be a randomly chosen generator of group $\mathbb{G}$. *Let* $\varsigma \in_R \mathbb{Z}_p^*$ *be chosen at random. Given as input a* $(m+1)$*-tuple* $T = (g, g^{\varsigma}, g^{\varsigma^2} \ldots, g^{\varsigma^m}) \in \mathbb{G}^{m+1}$, *for any PPT adversary* $\mathcal{A}$, *the following probability is negligible*

$$\mathsf{Pr}\left[d = e(g,g)^{1/(\varsigma+c)} \text{ where } (c,d) = \mathcal{A}(T)\right] \leq negl(\log p). \tag{16}$$

Theorem 1 *Suppose m-BSDH Assumption hold, and* PRF *is a secure pseudorandom function. The* DPOS *scheme constructed in Sec 4 is sound, according to Definition 3.*

Game 1 The first game is the same as soundness security game $\mathtt{Game}_{\mathtt{sound}}$, except that the pseudorandom function PRF_{s_0} outputs true randomness. Precisely, for each given seed s_0, the function PRF is evaluated in the following way:

1. The challenger keeps a table to store all previous encountered input-output pairs $(v, \mathsf{PRF}_{s_0}(v))$.
2. Given an input v, the challenger lookups the table for v, if there exists an entry (v,u), then return u as output. Otherwise, choose u at random from the range of PRF, insert $(v, \mathsf{PRF}_{s_0}(v) := u)$ into the table and return u as output.

Game 2 The second game is the same as **Game 1**, except that the pseudorandom function PRF_{s_1} with seed s_1 outputs true randomness. The details are similar as in **Game 1**.

Lemma 2 *Suppose m-BSDH Assumption holds. In* **Game 2**, *any PPT adversary is unable to find two* distinct *valid tuples* $T_0 \neq T_1$ *and the last four elements of tuple* T_0 *and* T_1 *are equal, where* $T_0 = (z, \phi_\alpha;\ \bar{\sigma}, \bar{t}, \psi_\alpha, \psi_\beta)$ *and* $T_1 = (z', \phi'_\alpha;\ \bar{\sigma}, \bar{t}, \psi_\alpha, \psi_\beta)$. *Precisely, for any PPT adversary* $\mathcal{A}$ *in* **Game 2**, *there exists a PPT algorithm* $\mathcal{B}$, *such that*

$$\mathsf{Pr}\left[\begin{matrix} T_0 \neq T_1 \text{ are both valid and } T_0[2,3,4,5] = T_1[2,3,4,5], \\ \text{where } (T_0,T_1) = \mathcal{A}^{\mathbf{Game\ 2}} \end{matrix}\right] \tag{17}$$

$$\leq \mathsf{Pr}\left[\mathcal{B} \text{ solves } m\text{-BSDH } \textit{Problem}\right] \tag{18}$$

(Proof of Lemma 2 is given in Appendix B.2)

Lemma 3 *Suppose m-BSDH Assumption holds. In* **Game 2**, *any PPT adversary is unable to find two* distinct *valid tuples* $T_0 \neq T_1$ *such that the last four elements of* T_0 *and* T_1 *are not equal, where* $T_0 = (z, \phi_\alpha;\ \bar{\sigma}, \bar{t}, \psi_\alpha, \psi_\beta)$ *and* $T_1 = (z', \phi'_\alpha;\ \bar{\sigma}', \bar{t}', \psi'_\alpha, \psi'_\beta)$ *and* $(\bar{\sigma}, \bar{t}, \psi_\alpha, \psi_\beta) \neq (\bar{\sigma}', \bar{t}', \psi'_\alpha, \psi'_\beta)$. *Precisely, for any PPT adversary* $\mathcal{A}$ *in* **Game 2**, *there exists a PPT algorithm* $\mathcal{D}$, *such that*

$$\mathsf{Pr}\left[\begin{matrix} T_0 \neq T_1 \text{ are both valid and } T_0[2,3,4,5] \neq T_1[2,3,4,5], \\ \text{where } (T_0,T_1) = \mathcal{A}^{\mathbf{Game\ 2}} \end{matrix}\right] \tag{19}$$

$$\leq 2\mathsf{Pr}\left[\mathcal{D} \text{ solves } m\text{-BSDH } \textit{Problem}\right] \tag{20}$$

(Proof of Lemma 3 is given in Appendix B.3).

Proof (Sketch proof of Theorem 1). Since PRF is a secure pseudorandom function, the soundness security game $\mathtt{Game}_{\mathtt{sound}}$ and **Game 1** are computationally indistinguishable. So are **Game 1** and **Game 2**. Therefore, Lemma 2 and Lemma 3 also hold in $\mathtt{Game}_{\mathtt{sound}}$ with negligible difference in success probability.

Using proof of contradiction, one can show that: For any adversary $\mathcal{A}$ that wins $\mathtt{Game}_{sound}$, there exists noticeable fraction of possible weights $W := \{w_i \in \mathbb{Z}_p :$

$i \in \mathbf{C}\}$ over the domain of weights W, such that for each weight W, there exists noticeable fraction of points $\xi \in \mathbb{Z}_p$ over the domain $\mathbb{Z}_p$, the adversary $\mathcal{A}$ can provide the correct response to the challenge $(\{(i, w_i) : i \in \mathbf{C}\}, \xi)$ where $\{w_i : i \in \mathbf{C}\} = W$.

Therefore, from sufficient number of points $z = \mathsf{Poly}_{\vec{\vec{F}}}(\xi)$ along the same polynomial $\mathsf{Poly}_{\vec{\vec{F}}}(\cdot)$, the knowledge extractor can find the coefficient vector $\vec{\vec{F}}$ by solving linear equation system over $\mathbb{Z}_p$. From $\mathtt{transcript}_{\mathcal{A}}$, the knowledge extractor can recover the blinding randomness $\vec{y}$, and thus recover the weighted sum of file blocks: $\sum_{i \in \mathbf{C}} w_i \vec{F}_i$. Furthermore, from sufficient number of weighted sum w.r.t different weights W's, the knowledge extractor can recover each file block $\vec{F}_i, i \in \mathbf{C}$, by solving a linear equation system over $\mathbb{Z}_p$. Consequently, the challenger of $\mathtt{Game}_{\mathtt{sound}}$ wins the game, as we desire. □

5.3 Privacy Preserving

Definition 6 (Discrete Log Assumption) *Let $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ be a bilinear map where $\mathbb{G}$ and $\mathbb{G}_T$ are both multiplicative cyclic groups of prime order p. Let g be a randomly chosen generator $g \in \mathbb{G}$ and $h \in_R \mathbb{G}$ be a randomly chosen group element. For any PPT adversary $\mathcal{A}$, the probability that $\mathcal{A}$ can output x such that $g^x = h$ is negligible. That is,*

$$\mathsf{Pr}\left[g^x = h \text{ where } x = \mathcal{A}(g, h)\right] \leq negl(\log p). \tag{21}$$

Theorem 4 *Suppose Discrete Log (DL) Assumption holds in group $\mathbb{G}$ and PRF is a secure pseudorandom function. The DPOS scheme constructed in Sec 4 is privacy-preserving according to Definition 4.*

Game 4 This game is identical to the privacy security game $\mathtt{Game}_{\mathtt{private}}$, except that the pseudorandom values $\{\mathsf{PRF}_{s_0}(i)\}$ with seed s_0 are replaced by true random values, as in **Game 1**. *Note: We emphasize that the values $\mathsf{PRF}_{s_1}(i)$ with seed s_1 is* not *replaced, and it is still the actual "pseudorandom" function output. In the privacy security game, the adversary is the dishonest verifier, who will obtain the seed s_1 as a part of the verification secret key vsk.*

Game 5 This game is identical to **Game 4**, except that adversary $\mathcal{A}$ (i.e the dishonest verifier/auditor) is provided with extra information $(g^{\langle \vec{\alpha}, \vec{F}_i \rangle}, g^{\rho \langle \vec{\beta}, \vec{F}_i \rangle}, g^{\sigma_i}, g^{t_i})$ for each i and for each file.

Lemma 5 *Suppose the Discrete Log Assumption holds in group $\mathbb{G}$. In* **Game 5**, *any PPT adversary $\mathcal{A}$ (i.e. the dishonest verifier/auditor) is unable to output file sector $\vec{F}_i[j]$ for any $i \in [0, n-1], j \in [0, m-1]$. Precisely, for any PPT adversary $\mathcal{A}$, there exists PPT algorithm $\mathcal{E}$, such that*

$$\mathsf{Pr}\left[\mathcal{A} \textit{ wins } \textbf{Game 5}\right] \leq \mathsf{Pr}\left[\mathcal{E} \textit{ solves } \textbf{DL } \textit{problem}\right]. \tag{22}$$

(Proof of Lemma 5 is given in Appendix C)

Proof (of Theorem 4). Since PRF is a secure pseudorandom function, we have $\mathsf{Pr}\left[\mathcal{A} \text{ wins } \mathtt{Game}_{\mathtt{private}}\right] \leq \mathsf{Pr}\left[\mathcal{A}_2 \text{ wins } \textbf{Game 4}\right] + negl(\lambda)$ for any PPT algorithm $\mathcal{A}_2$. Since **Game 5** is identical to **Game 4**, except that the adversary will obtain extra information, we have $\mathsf{Pr}\left[\mathcal{A}_2 \text{ wins } \textbf{Game 4}\right] \leq \mathsf{Pr}\left[\mathcal{A}_3 \text{ wins } \textbf{Game 5}\right]$ for any PPT algorithm $\mathcal{A}_3$.

Combining the above two inequalities and Lemma 5, Theorem 4 is proved. □

6 Conclusion

We proposed a novel POS scheme which is lightweight and privacy preserving. On one side, the proposed scheme is as efficient as private key POS scheme, especially very efficient in authentication tag generation. On the other side, the proposed scheme supports third party auditor and can revoke an auditor at any time, close to the functionality of publicly verifiable POS scheme. Compared to existing publicly verifiable POS scheme, our scheme improves the authentication tag generation speed by more than 100 times. Our scheme also prevents data leakage to the auditor during the auditing process. How to enable dynamic operations (e.g. inserting/deleting a data block) in our scheme is in future work.

References

1. Alwen, J., Dodis, Y., Wichs, D.: Leakage-Resilient Public-Key Cryptography in the Bounded-Retrieval Model. In: CRYPTO'09. pp. 36–54 (2009)
2. Aniket Kate, Gregory M. Zaverucha, I.G.: Constant-Size Commitments to Polynomials and Their Applications. In: ASIACRYPT'10. pp. 177–194
3. Ateniese, G., Burns, R., Curtmola, R., Herring, J., Khan, O., Kissner, L., Peterson, Z., Song, D.: Remote data checking using provable data possession. ACM Tran. on Info. and Sys. Sec., TISSEC 2011 14(1), 12:1–12:34 (2011)
4. Ateniese, G., Burns, R., Curtmola, R., Herring, J., Kissner, L., Peterson, Z., Song, D.: Provable data possession at untrusted stores. In: ACM CCS'07. pp. 598–609. ACM (2007)
5. Ateniese, G., Kamara, S., Katz, J.: Proofs of storage from homomorphic identification protocols. In: ASIACRYPT'09. LNCS, vol. 5912, pp. 319–333. Springer (2009)
6. Ateniese, G., Pietro, R.D., Mancini, L.V., Tsudik, G.: Scalable and efficient provable data possession. In: SecureComm'08. pp. 9:1–9:10. ACM (2008)
7. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the weil pairing. Journal of Cryptology 17(4), 297–319 (2004)
8. Bowers, K.D., Juels, A., Oprea, A.: HAIL: A high-availability and integrity layer for cloud storage. In: ACM CCS'09. pp. 187–198. ACM (2009)
9. Bowers, K.D., Juels, A., Oprea, A.: Proofs of retrievability: Theory and implementation. In: CCSW'09. pp. 43–54. ACM (2009)
10. Cash, D., Küpçü, A., Wichs, D.: Dynamic proofs of retrievability via oblivious RAM. In: EUROCRYPT'13. LNCS, vol. 7881, pp. 279–295. Springer (2013)
11. Chang, E.C., Xu, J.: Remote integrity check with dishonest storage server. In: ESORICS'08. LNCS, vol. 5283, pp. 223–237. Springer (2008)
12. Curtmola, R., Khan, O., Burns, R., Ateniese, G.: MR-PDP: Multiple-replica provable data possession. In: ICDCS'08. pp. 411–420. IEEE (2008)
13. Deswarte, Y., Quisquater, J.J., Saïdane, A.: Remote integrity checking: How to trust files stored on untrusted servers. In: Integrity and Internal Control in Information Systems VI. LNCS, vol. 140, pp. 1–11. Springer (2004)
14. Dodis, Y., Vadhan, S., Wichs, D.: Proofs of Retrievability via Hardness Amplification. In: Proceedings of TCC'09, LNCS, vol. 5444, pp. 109–127. Springer (2009)
15. Erway, C., Küpçü, A., Papamanthou, C., Tamassia, R.: Dynamic provable data possession. In: ACM CCS'09. pp. 213–222. ACM (2009)
16. Goldreich, O.: A Sample of Samplers - A Computational Perspective on Sampling (survey). Electronic Colloquium on Computational Complexity (ECCC) 4(20) (1997)
17. Hao, Z., Zhong, S., Yu, N.: A privacy-preserving remote data integrity checking protocol with data dynamics and public verifiability. TKDE'11 23(9), 1432–1437 (2011)
18. Juels, A., Burton S. Kaliski, J.: PORs: Proofs of retrievability for large files. In: ACM CCS'07. pp. 584–597. ACM (2007)
19. Naor, M., Rothblum, G.N.: The complexity of online memory checking. Journal of the ACM 56(1) (2009)
20. Okamoto, T.: Provably secure and practical identification schemes and corresponding signature schemes. In: CRYPTO'92. pp. 31–53
21. Reed, I.S., Solomon, G.: Polynomial codes over certain finite fields. Journal of the Society for Industrial and Applied Mathematics 8(2), 300–304 (1960)
22. Schwarz, T.J.E., Miller, E.L.: Store, forget, and check: Using algebraic signatures to check remotely administered storage. In: ICDCS'06. IEEE (2006)

23. Sebé, F., Domingo-Ferrer, J., Martínez-Ballesté, A., Deswarte, Y., Quisquater, J.J.: Efficient remote data possession checking in critical information infrastructures. TKDE'08 20(8), 1034–1038 (2008)
24. Shacham, H., Waters, B.: Compact proofs of retrievability. In: ASIACRYPT'08. LNCS, vol. 5350, pp. 90–107. Springer (2008)
25. Shacham, H., Waters, B.: Compact proofs of retrievability. Journal of Cryptology 26(3), 442–483 (2013)
26. Shah, M.A., Baker, M., Mogul, J.C., Swaminathan, R.: Auditing to keep online storage services honest. In: HotOS'07. USENIX Association (2007)
27. Shah, M.A., Swaminathan, R., Baker, M.: Privacy-preserving audit and extraction of digital contents. Cryptology ePrint Archive, Report 2008/186 (2008), http://eprint.iacr.org/2008/186
28. Shi, E., Stefanov, E., Papamanthou, C.: Practical dynamic proofs of retrievability. In: ACM CCS'13. pp. 325–336. ACM (2013)
29. Wang, B., Li, B., Li, H.: Oruta: Privacy-preserving public auditing for shared data in the cloud. In: IEEE Cloud 2012. pp. 295–302. IEEE (2012)
30. Wang, B., Li, B., Li, H.: Public auditing for shared data with efficient user revocation in the cloud. In: INFOCOM'13. pp. 2904–2912. IEEE (2013)
31. Wang, C., Chow, S.S., Wang, Q., Ren, K., Lou, W.: Privacy-preserving public auditing for secure cloud storage. IEEE Tran. on Computers 62(2), 362–375 (2013)
32. Wang, C., Ren, K., Lou, W., Li, J.: Toward publicly auditable secure cloud data storage services. IEEE Network Magazine 24(4), 19–24 (2010)
33. Wang, C., Wang, Q., Ren, K., Cao, N., Lou, W.: Towards secure and dependable storate services in cloud computing. IEEE Transactions on Services Computing 5(2), 220–232 (2012)
34. Wang, C., Wang, Q., Ren, K., Lou, W.: Ensuring data storage security in cloud computing. In: Proceedings of IWQoS'09. pp. 1–9. IEEE (2009)
35. Wang, C., Wang, Q., Ren, K., Lou, W.: Privacy-preserving public auditing for data storage security in cloud computing. In: INFOCOM'10. pp. 525–533. IEEE (2010)
36. Wang, Q., Wang, C., Li, J., Ren, K., Lou, W.: Enabling public verifiability and data dynamics for storage security in cloud computing. In: ESORICS'09. LNCS, vol. 5789, pp. 355–370. Springer (2009)
37. Wang, Q., Wang, C., Ren, K., Lou, W., Li, J.: Enabling public auditability and data dynamics for storage security in cloud computing. TPDS'11 22(5), 847–859 (2011)
38. Xu, J., Chang, E.C.: Towards efficient proofs of retrievability. In: ASIACCS'12. pp. 79–90. ACM (2012)
39. Yang, K., Jia, X.: Data storage auditing service in cloud computing: challenges, methods and opportunities. World Wide Web 15(4), 409–428 (2012)
40. Yang, K., Jia, X.: An efficient and secure dynamic auditing protocol for data storage in cloud computing. TPDS'13 24(9), 1717–1726 (2013)
41. Yuan, J., Yu, S.: Proofs of retrievability with public verifiability and constant communication cost in cloud. In: Proceedings of the 2013 International Workshop on Security in Cloud Computing, Cloud Computing 2013. pp. 19–26. ACM (2013)
42. Zeng, K.: Publicly verifiable remote data integrity. In: ICICS'08. LNCS, vol. 5308, pp. 419–434. Springer (2008)
43. Zhu, Y., Hu, H., Ahn, G.J., Yu, M.: Cooperative provable data possession for integrity verification in multicloud storage. TPDS'12 23(12), 2231–2244 (2012)
44. Zhu, Y., Wang, H., Hu, Z., Ahn, G.J., Hu, H., Yau, S.S.: Dynamic audit services for integrity verification of outsourced storages in clouds. In: Proceedings of SAC'11. pp. 1550–1557. ACM (2011)

A Proof for Completeness

Substituting Eq (10) into Eq (9), we have left hand side of Eq (9):

$$LHS^{\gamma^{-1}} = \frac{e(g,g)^{\alpha \mathsf{Poly}_{\vec{F}}(\alpha)}}{e(g,Y_\alpha)\cdot e(g, g^{r\sum_{i\in\mathbf{C}} w_i\sigma_i})} = \frac{e(g,g)^{\alpha \mathsf{Poly}_{\vec{F}}(\alpha)}}{e(g, g^{\alpha \mathsf{Poly}_{\vec{y}}(\alpha)})\cdot e(g, g^{r\sum_{i\in\mathbf{C}} w_i\sigma_i})} \tag{23}$$

$$= \frac{e(g,g)^{\langle \vec{\alpha},\ \vec{F}\rangle}}{e(g, g^{\langle \vec{\alpha},\ \vec{y}\rangle})\cdot e\left(g, g^{r\sum_{i\in\mathbf{C}} w_i(\langle \vec{\alpha},\ \vec{F_i}\rangle + \mathsf{PRF}_{s_0}(i))}\right)} \tag{24}$$

$$= \frac{e(g,g)^{\left\langle \vec{\alpha},\ r\sum_{i\in\mathbf{C}} w_i\vec{F_i}+\vec{y}\right\rangle}}{e(g,g)^{\langle \vec{\alpha},\ \vec{y}\rangle}\cdot e\,(g,g)^{r\left(\left\langle \vec{\alpha},\ \sum_{i\in\mathbf{C}} w_i\vec{F_i}\right\rangle + \sum_{i\in\mathbf{C}} w_i\mathsf{PRF}_{s_0}(i)\right)}} \tag{25}$$

$$= \frac{1}{e(g,g)^{r\sum_{i\in\mathbf{C}} w_i\mathsf{PRF}_{s_0}(i)}}. \tag{26}$$

The right hand side (RHS) of Eq (9) can be shown in a similar way, we leave details to full paper.

B Proof for Soundness

B.1 Simulate Game 2 using the input of m-BSDH Problem

Let bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ and tuple $T = (g, g^{\varsigma}, g^{\varsigma^2}, \ldots, g^{\varsigma^m}) \in \mathbb{G}^m$ be as stated in the m-BSDH Assumption. With information T, we can simulate **Game 2** as below. Recall that the adversary $\mathcal{A}$ in this game is the dishonest prover (i.e. the cloud storage server).
Setup Choose at random a bit $\iota \in \{0, 1\}$. If $\iota = 1$, let $\alpha = \varsigma$, and choose at random $\beta \in_R \mathbb{Z}_p^*$. If $\iota = 0$, let $\beta = \varsigma$, and choose at random $\alpha \in_R \mathbb{Z}_p^*$.

Choose two random group elements $\gamma, \rho \in_R \mathbb{Z}_p^*$. For each $j \in [1, m]$, we can find values of g_j, h_j: $g_j = g^{\alpha^j} \in \mathbb{G}; h_j = \left(g^{\beta^j}\right)^{\rho} \in \mathbb{G}$. Let $g_0 = g$ and $h_0 = g^{\rho}$. The secret key is $sk = (\alpha, \beta, s_0)$, the public key is $pk = (g_0, g_1, \ldots, g_m)$, the verification secret key is $vsk = (\rho, \gamma, s_1)$, and the verification public key is $vpk = (h_0, h_1, \ldots, h_m)$, where the two random seeds (s_0, s_1) for pseudorandom function $\mathcal{PRF}$ will be determined later. Notice that, the simulator does not have information of (s_0, s_1), and does not know either α or β: if $\iota = 1$, $\alpha = \varsigma$ is unknown and β is known; if $\iota = 0$, $\beta = \varsigma$ is unknown and α is known. Send (pk, vpk) to the adversary.
Learning

- StoreQuery(F): Given a data file F chosen by the adversary $\mathcal{A}$. The simulator simulates the algorithm Tag as below: For each $i \in [0, n-1]$, choose the authentication tags $\sigma_i, t_i \in_R \mathbb{Z}_p$ at random, which will implicitly define the values of the file-specific randomness $\mathsf{PRF}_{s_0}(i)$ and $\mathsf{PRF}_{s_1}(i)$. The simulator is able to compute: $g^{\langle \vec{\alpha},\ \vec{F}_i \rangle} = \prod_{j=1}^{m} g_j^{\vec{F}_i[j-1]}$; $g^{\rho\langle \vec{\beta},\ \vec{F}_i \rangle} = \prod_{j=1}^{m} h_j^{\vec{F}_i[j-1]}$; $g^{\mathsf{PRF}_{s_0}(i)} = \frac{g^{\sigma_i}}{g^{\langle \vec{\alpha},\ \vec{F}_i \rangle}}$; $g^{\mathsf{PRF}_{s_1}(i)} = \frac{g^{t_i}}{g^{\rho\langle \vec{\beta},\ \vec{F}_i \rangle} \cdot \left(g^{\mathsf{PRF}_{s_0}(i)}\right)^{\gamma}}$.
  Send file F and authentication tags $\{(i, \sigma_i, t_i)\}$ to the adversary $\mathcal{A}$.
- VerifyQuery: The simulator has full information of pk, vpk, and γ. Although the simulator does not know s_1, it knows values $g^{\mathsf{PRF}_{s_1}(i)}$ for each $i \in [0, n-1]$. The simulator can execute the verifier algorithm V in the proposed DPOS scheme exactly, except that when computing the right hand side of Eq (9), the term $g^{-r \sum_{i \in \mathbf{C}} w_i \mathsf{PRF}_{s_1}(i)}$ is computed as $\prod_{i \in \mathbf{C}} \left(g^{\mathsf{PRF}_{s_1}(i)}\right)^{-r w_i}$. So the simulator can find the exact the same decision bit $b \in \{\mathtt{Accept}, \mathtt{Reject}\}$ as in a real game.
- RevokeVKQuery: Choose at random $\gamma' \in_R \mathbb{Z}_p^*$. Update h_j exactly as in algorithm UpdVK: $h_j' := h_j^{\gamma'}$, $j \in [1, m]$. Update ρ as $\rho' := \gamma' \cdot \rho$. For each $i \in [0, n-1]$, choose the new authentication tag $t_i' \in_R \mathbb{Z}_p$ at random, which will implicitly define the new version of file-specific randomness $\mathsf{PRF}_{s_1'}(i)$ as $g^{\mathsf{PRF}_{s_1'}(i)} := g^{t_i'} \cdot \left(\frac{g^{t_i}}{g^{\mathsf{PRF}_{s_1}(i)}}\right)^{-\gamma'}$. So the simulator knows the value of $g^{\mathsf{PRF}_{s_1'}(i)}$ for each $i \in [0, n-1]$.

B.2 Proof for Lemma 2

Proof (of Lemma 2). Our hypothesis is: Adversary $\mathcal{A}$ can output such (T_0, T_1) as stated in Lemma 2. Let bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ and tuple $T = (g, g^{\varsigma}, g^{\varsigma^2}, \ldots, g^{\varsigma^m}) \in \mathbb{G}^m$ be as stated in the m-BSDH Assumption. We will construct a PPT algorithm $\mathcal{B}$, which will simulate **Game 2** to interact with adversary $\mathcal{A}$, and then compute (c, d) from $\mathcal{A}$'s output (T_0, T_1), such that $d = e(g, g)^{1/(\varsigma + c)}$.

The algorithm $\mathcal{B}$ simulates the **Game 2** to interact with the adversary $\mathcal{A}$, from information T, as in Appendix B.1. By the hypothesis, both T_0 and T_1 are valid, i.e. satisfy

Eq (8) and Eq (9):

$$e(\psi_\alpha, g) = e(\phi_\alpha,\ g^\alpha/g^\xi) \cdot e(g,g)^z \tag{27}$$

$$e(\psi_\alpha, g) = e(\phi'_\alpha,\ g^\alpha/g^\xi) \cdot e(g,g)^{z'} \tag{28}$$

Combining Eq (27) and Eq (28), we have $e(\phi_\alpha/\phi'_\alpha,\ g^\alpha/g^\xi) = e(g,g)^{z'-z}$.
Case 1: $z' = z \mod p$. If $\alpha = \xi$, then find any value $c \neq -\alpha \mod p$, compute $d = e(g,g)^{1/(\alpha+c)} = e(g,g)^{1/(\xi+c)}$. Output (c,d) as solution to m-BSDH problem. If $\alpha \neq \xi$, then $\phi_\alpha = \phi'_\alpha$. As a result, $T_0 = T_1$, contradicting with the hypothesis that $T_0 \neq T_1$.
Case 2: $z' \neq z \mod p$. The inverse $1/(z'-z) \mod p$ exists. Let $c = -\xi \mod p$ and compute d as $d = e(\phi_\alpha/\phi'_\alpha,\ g)^{1/(z'-z) \mod p} \in \mathbb{G}_T$. Output (c,d) as the solution to the m-BSDH problem. One can verify that

$$d^{\alpha+c} = \left(e(\phi_\alpha/\phi'_\alpha,\ g)^{1/(z'-z) \mod p}\right)^{\alpha-\xi} = e(\phi_\alpha/\phi'_\alpha,\ g^{\alpha-\xi})^{1/(z'-z) \mod p} \tag{29}$$

$$= \left(e(g,g)^{z'-z}\right)^{1/(z'-z) \mod p} = e(g,g). \tag{30}$$

B.3 Proof for Lemma 3

Proof (of Lemma 3). Our hypothesis is: Adversary $\mathcal{A}$ can output such (T_0, T_1) as stated in Lemma 3. Let bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ and tuple $T = (g, g^\varsigma, g^{\varsigma^2}, \ldots, g^{\varsigma^m}) \in \mathbb{G}^m$ be as stated in the m-BSDH Assumption. We will construct a PPT algorithm $\mathcal{D}$, which will simulate **Game 2** to interact with adversary $\mathcal{A}$, and then compute (c,d) from $\mathcal{A}$'s output (T_0, T_1), such that $d = e(g,g)^{1/(\varsigma+c)}$.

The algorithm $\mathcal{D}$ simulates the **Game 2** to interact with the adversary $\mathcal{A}$, from information T, as in Appendix B.1. By the hypothesis, both T_0 and T_1 are valid, i.e. satisfy Eq (8) and Eq (9):

$$\left(\frac{e(\psi_\alpha,\ g^\alpha)}{e\left(g,\ Y_\alpha \cdot (g^{\bar\sigma}/Y_\sigma)^{r_\sigma^{-1}\cdot r}\right)}\right)^\gamma = \frac{e(\psi_\beta, g^\beta)}{e\left(g,\ Y_\beta \cdot (g^{\bar t}/Y_t)^{r_t^{-1}\cdot r} \cdot g^{-r\sum_{i\in\mathbf{C}} w_i \mathsf{PRF}_{s_1}(i)}\right)} \tag{31}$$

$$\left(\frac{e(\psi'_\alpha,\ g^\alpha)}{e\left(g,\ Y_\alpha \cdot (g^{\bar\sigma'}/Y_\sigma)^{r_\sigma^{-1}\cdot r}\right)}\right)^\gamma = \frac{e(\psi'_\beta, g^\beta)}{e\left(g,\ Y_\beta \cdot (g^{\bar t'}/Y_t)^{r_t^{-1}\cdot r} \cdot g^{-r\sum_{i\in\mathbf{C}} w_i \mathsf{PRF}_{s_1}(i)}\right)} \tag{32}$$

Divide Eq (31) with Eq (32), we have

$$\left(\frac{e(\frac{\psi_\alpha}{\psi'_\alpha},\ g^\alpha)}{e(g,g)^{r_\sigma^{-1}\cdot r\cdot \Delta\sigma}}\right)^\gamma = \left(\frac{e(\frac{\psi_\beta}{\psi'_\beta},\ g^\beta)}{e(g,g)^{r_t^{-1}\cdot r\cdot \Delta t}}\right) \in \mathbb{G}_T \tag{33}$$

where $\Delta\sigma := \bar\sigma' - \bar\sigma$ and $\Delta t := \bar t' - \bar t$.

For ease of exposition, let us represent the above Eq (33) as $A^\gamma = B$, where the meaning of variable A and B can be explained straightforwardly by looking at Eq (33). The adversary $\mathcal{A}$ (i.e. the dishonest prover/cloud storage server) has sufficient information to compute values of A and B by itself. So the adversary $\mathcal{A}$ is able to compute values A and B, such that $A^\gamma = B$.

Notice that, in our scheme, among all data that the adversary (dishonest cloud storage server) owns (i.e data blocks, authentication tags $\{\sigma_i, t_i\}$, and public keys (pk, vpk)), the secret value γ only appears in the computation of t_i, where γ is perfectly protected by $\mathsf{PRF}_{s_1}(i)$ (which is *true* randomness in **Game 2**) from the adversary $\mathcal{A}$ (dishonest prover).

Once a verification public/private key pair is revoked, γ will be re-randomized as $\gamma'\gamma$, where γ' is a newly chosen uniform random variable hidden from $\mathcal{A}$. herefore, γ is semantically secure against $\mathcal{A}$, and $\mathcal{A}$ is unable do brute-force search attack to find values of γ. Therefore, the adversary $\mathcal{A}$ is unable to compute a pair (A, B) such that $A^\gamma = B$ and $A \neq 1$. As a result, it has to be the case that $A = B = 1 = e(g,g)^0 \in \mathbb{G}_T$.

We have $A = \left(\frac{e(\frac{\psi_\alpha}{\psi'_\alpha},\ g^\alpha)}{e(g,g)^{r_\sigma^{-1}\cdot r\cdot \Delta\sigma}}\right) = \left(\frac{e(\frac{\psi_\beta}{\psi'_\beta},\ g^\beta)}{e(g,g)^{r_t^{-1}\cdot r\cdot \Delta t}}\right) = B = 1 \quad \in \mathbb{G}_T$

Recall that $(\psi_\alpha, \psi_\beta, \bar{\sigma}, \bar{t}) \neq (\psi'_\alpha, \psi'_\beta, \bar{\sigma}', \bar{t}')$ are distinct, by our hypothesis. We define three mutually exclusive events as below: (1) $\mathbf{E}_1$: $\bar{\sigma} \neq \bar{\sigma}' \mod p$, i.e. $\Delta\sigma \neq 0 \mod p$. (2) $\mathbf{E}_2$: $\neg\mathbf{E}_1 \wedge (\bar{t} \neq \bar{t}' \mod p)$. (3) $\mathbf{E}_3$: $\neg\mathbf{E}_1 \wedge \neg\mathbf{E}_2$.

Case 1: $\mathbf{E}_1$. Let $c = 0 \in \mathbb{Z}_p$ and compute $d \in \mathbb{G}_T$ as below

$$d = e(\frac{\psi_\alpha}{\psi'_\alpha},\ g)^{r_\sigma\cdot(r\cdot\Delta\sigma)^{-1} \mod p} \in \mathbb{G}_T. \tag{34}$$

In case $\varsigma = \alpha$, the solution to the m-BSDH problem is (c, d). One can verify that

$$e(g,\ g) = \left(e(\frac{\psi_\alpha}{\psi'_\alpha}, g^\alpha)\right)^{r_\sigma\cdot(r\cdot\Delta\sigma)^{-1} \mod p} \tag{35}$$

$$e(g,g)^{1/(\alpha+c)} = \left(e(\frac{\psi_\alpha}{\psi'_\alpha}, g^{\alpha/(\alpha+c)})\right)^{r_\sigma\cdot(r\cdot\Delta\sigma)^{-1} \mod p} = d \ \in \mathbb{G}_T \tag{36}$$

Case 2: $\mathbf{E}_2$ Similar as above, the algorithm $\mathcal{D}$ can break m-BSDH Assumption w.r.t. $\varsigma = \beta$.

Case 3: $\mathbf{E}_3$. Under the hypothesis that $(\psi_\alpha, \psi_\beta, \bar{\sigma}, \bar{t}) \neq (\psi'_\alpha, \psi'_\beta, \bar{\sigma}', \bar{t}')$, we have $\mathsf{Pr}[\mathbf{E}_3] = 0$. By the definition of event $\mathbf{E}_3$, $\Delta\sigma = \Delta t = 0 \mod p$. From $A = B = 1$, we have $e(\frac{\psi_\alpha}{\psi'_\alpha},\ g^\alpha) = e(g,g)^0 = e(\frac{\psi_\beta}{\psi'_\beta},\ g^\beta) \in \mathbb{G}_T$. Since $\alpha, \beta \in \mathbb{Z}_p^*$ and $\alpha \neq 0 \neq \beta$, we have $\psi_\alpha = \psi'_\alpha$ and $\psi_\beta = \psi'_\beta$. As a result, we have $(\psi_\alpha, \psi_\beta, \bar{\sigma}, \bar{t}) = (\psi'_\alpha, \psi'_\beta, \bar{\sigma}', \bar{t}')$, which **contradicts** with our hypothesis that $(\psi_\alpha, \psi_\beta, \bar{\sigma}, \bar{t}) \neq (\psi'_\alpha, \psi'_\beta, \bar{\sigma}', \bar{t}')$ are distinct.

Recall that in the simulation in Appendix B.1, if the random bit $\iota = 1$, then $\varsigma = \alpha$ and if $\iota = 0$, then $\varsigma = \beta$. It is easy to see that, the random bit ι is perfectly protected from the adversary $\mathcal{A}$ in the simulated game. Therefore,

$$\mathsf{Pr}\left[(c,d) \text{ solves } m\text{-BSDH problem}\right] = \mathsf{Pr}\left[\iota = 1 \wedge \mathbf{E}_1\right] + \mathsf{Pr}\left[\iota = 0 \wedge \mathbf{E}_2\right] \tag{37}$$

$$= \frac{1}{2}\mathsf{Pr}\left[\mathbf{E}_1 \vee \mathbf{E}_2\right] = \frac{1}{2}\mathsf{Pr}\left[\mathcal{A} \text{ wins Lem 3 }\right] \tag{38}$$

□

C Proof for Privacy-Preserving

Proof (of Lemma 5). Given input $(g, g^x) \in \mathbb{G}^2$, where $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is a bilinear map of prime order p, and g is a generator of group $\mathbb{G}$. The goal of DL problem is to output the secret exponent $x \in \mathbb{Z}_p$. We construct algorithm $\mathcal{E}$ as below.

Algorithm $\mathcal{E}$ simulates the challenger in **Game 5** to interact with the adversary $\mathcal{A}$.

Setup. Simulate key generation algorithm KeyGen as below: Choose $\alpha, \beta, \gamma, \rho, s_1$ and compute $g_j, h_j, j \in [0, m]$ and keys (pk, vpk, vsk) in the same way as in algorithm KeyGen. Let $sk = (\alpha, \beta, s_0)$, where PRF seed s_0 will be determined later.

Simulate the algorithm Tag for the target input file F^* as below: Choose at random $x_{i,j} \in_R \mathbb{Z}_p$ for each $i \in [0, n-1]$ and $j \in [0, m-1]$. Set the (unknown) target file as $F^* = (\vec{\boldsymbol{F}}_0^*, \ldots, \vec{\boldsymbol{F}}_{n-1}^*)$, where (unknown) data sector $\vec{\boldsymbol{F}}_i^*[j] := x + x_{i,j} \mod p$. Algorithm $\mathcal{E}$ can compute $g^{\vec{\boldsymbol{F}}_i^*[j]} = g^x \cdot g^{x_{i,j}}$ for the unknown data sector $\vec{\boldsymbol{F}}_i^*[j]$. If the adversary $\mathcal{A}$ can

output the value of some file sector $\vec{\boldsymbol{F}}_i^*[j]$, then $\mathcal{E}$ can find the solution $x = \vec{\boldsymbol{F}}_i^*[j] - x_{i,j}$ to the DL problem.

For each $i \in [0, n-1]$, choose at random the authentication tag $\sigma_i \in_R \mathbb{Z}_p$, which will implicitly define the value of $\mathsf{PRF}_{s_0}(i)$ and authentication tag t_i:

$$g^{\mathsf{PRF}_{s_0}(i)} := g^{\sigma_i} / \prod_{j=1}^{m} g_j^{\vec{\boldsymbol{F}}_i^*[j-1]}; \quad g^{t_i} := g^{\gamma\sigma_i} \cdot \prod_{j\in[1,m]} \left(g^{\vec{\boldsymbol{F}}_i[j-1]}\right)^{\rho\beta_j - \gamma\alpha_j} \cdot g^{\mathsf{PRF}_{s_1}(i)} \tag{39}$$

Notice that the above computation of g^{t_i} utilizes this equation: $t_i = \gamma\sigma_i + \left\langle \rho\vec{\boldsymbol{\beta}} - \gamma\vec{\boldsymbol{\alpha}},\ \vec{\boldsymbol{F}}_i \right\rangle + \mathsf{PRF}_{s_1}(i) \mod p$.

Compute and send extra information $\{g^{\langle\vec{\boldsymbol{\alpha}},\ \vec{\boldsymbol{F}}_i\rangle}, g^{\rho\langle\vec{\boldsymbol{\beta}},\ \vec{\boldsymbol{F}}_i\rangle}, g^{\sigma_i}, g^{t_i} : i \in [0, n-1]\}$ to the adversary $\mathcal{A}$.

Learning.

Store-Query: For any file F chosen by the adversary $\mathcal{A}$, simulate algorithm Tag for input file F as in the setup phase, except that the unknown file F^* is replaced by known file F.

Verify-Query: For any r, r_σ, r_t, ξ chosen by the adversary $\mathcal{A}$, $\mathcal{A}$ is able to compute the response of the prover by itself, with the identical distribution: Choose $\vec{\vec{\boldsymbol{F}}} \in_R \mathbb{Z}_p^m, \bar{\sigma}, \bar{t} \in_R \mathbb{Z}_p$ at random and compute $(Y_\alpha, Y_\beta, Y_\sigma, Y_t)$ as below:

$$Y_\alpha = \prod_{j\in[1,m]} g_j^{\vec{\vec{\boldsymbol{F}}}[j-1]} \Big/ \prod_{i\in\mathbf{C}} \left(g^{\langle\vec{\boldsymbol{\alpha}},\ \vec{\boldsymbol{F}}_i\rangle}\right)^{rw_i}; \quad Y_\sigma = g^{\bar{\sigma}} \Big/ \prod_{i\in\mathbf{C}} (g^{\sigma_i})^{r_\sigma w_i}; \tag{40}$$

$$Y_\beta = \prod_{j\in[1,m]} h_j^{\vec{\vec{\boldsymbol{F}}}[j-1]} \Big/ \prod_{i\in\mathbf{C}} \left(g^{\rho\langle\vec{\boldsymbol{\beta}},\ \vec{\boldsymbol{F}}_i\rangle}\right)^{rw_i}; \quad Y_t = g^{\bar{t}} \Big/ \prod_{i\in\mathbf{C}} \left(g^{t_i}\right)^{r_t w_i}. \tag{41}$$

The dishonest verifier $\mathcal{A}$ is able to compute $z, \psi_\alpha, \psi_\beta, \phi_\alpha$ from $\vec{\vec{\boldsymbol{F}}}$ and public keys (pk, vpk), in the same way as by an honest prover algorithm P. One can verify that the distribution of $(Y_\alpha, Y_\beta, Y_\sigma, Y_t;\ z, \phi_\alpha, \bar{\sigma}, \bar{t}, \psi_\alpha, \psi_\beta)$ computed by $\mathcal{A}$, is identical to the distribution of an honest prover's response. The only difference is that, in real case, the honest prover is able to commit the values of $(Y_\alpha, Y_\beta, Y_\sigma, Y_t)$, before seeing the value (r, r_σ, r_t) chosen by the verifier (here it is $\mathcal{A}$); in the simulated case here, $\mathcal{A}$ has to choose values (r, r_σ, r_t) first and then compute $(Y_\alpha, Y_\beta, Y_\sigma, Y_t)$. But such ordering does not provide more information to the adversary $\mathcal{A}$. We remark that $\mathcal{A}$'s simulation in the Verify-Query borrows ideas from the proof of the Okomato-Identity scheme.

RevokeVK-Query: Follow the algorithm UpdVK exactly, except that the new authentication tag t_i' is implicitly defined by value $g^{t_i'} := \left(g^{t_i} / g^{\mathsf{PRF}_{s_1}(i)}\right)^{\gamma'} \cdot g^{\mathsf{PRF}_{s_1'}(i)} \in \mathbb{G}$.

Guess The adversary $\mathcal{A}$ outputs (i, j, b).

The algorithm $\mathcal{E}$ outputs $x := b - x_{i,j} \mod p$ as solution to DL problem. Therefore,

$$\mathsf{Pr}\left[\mathcal{A} \text{ wins } \mathbf{Game\ 5}\right] = \mathsf{Pr}\left[b = \vec{\boldsymbol{F}}_i^*[j]\right] = \mathsf{Pr}\left[g^x = h \text{ where } x = \mathcal{E}(g,h)\right]. \tag{42}$$

□