Towards Efficient Proofs of Retrievability in Cloud Storage

Jia Xu and Ee-Chien Chang

National University of Singapore
Department of Computer Science
{xujia,changec}@comp.nus.edu.sg

Abstract. Proofs of Retrievability ($\mathcal{POR}$) is a cryptographic method for remotely auditing the integrity of files stored in the cloud, without keeping a copy of the original files in local storage. In a $\mathcal{POR}$ scheme, a user Alice backups her data file together with some authentication data to a potentially dishonest cloud storage server Bob. Later, Alice can periodically and remotely verify the integrity of her data stored with Bob using the authentication data, without retrieving back the data file during a verification. Besides security, performances in communication, storage overhead and computaton are major considerations. Shacham and Waters [1] gave a fast scheme with $\mathcal{O}(s)$ communication bits and a factor of $1/s$ file size expansion. Although Ateniese *et al.* [2] achieves constant communication requirement with the same $1/s$ storage overhead, it requires intensive computation in the setup and verification. In this paper, we incorporate a recent construction of constant size polynomial commitment scheme into Shacham and Waters [1] scheme. The resulting scheme requires constant communication bits (particularly, 720 bits if elliptic curve is used or 3312 bits if a modulo group is used) per verification and a factor of $1/s$ file size expansion, and its computation in the setup and verification is significantly reduced compared to Ateniese *et al.* [2]. Essentially, Ateniese *et al.* [2] requires one group multiplication per each bit of the data file in the setup, while the proposed scheme requires one group multiplication per each chunk of data bits (160 bits per chunk if elliptic curve is used or 1024 bits per chunk if modulo group is used). The experiment results show that our proposed scheme is indeed efficient and practical. Our security proof is based on Strong Diffie-Hellman Assumption.

Keywords: Cloud Storage, Proofs of Retrievability, Remote Data Integrity Check, Homomorphic Authentication Tag, Polynomial Commitment, Provable Data Possession

1 Introduction

Backuping data in a cloud storage, for example Amazon Cloud Drive, Microsoft Skydrive, or Dropbox, is gaining popularity recently. We are considering scenarios where users may have concerns of the integrity of their data stored in the cloud storage. Such prudent users may not be simply satisfied with the cloud storage server's promise on maintaining the data integrity. Instead, they desire a technical way to verify that whether the cloud storage server is keeping his promise and following the service level agreement (SLA). That is, these users want to base their data integrity on the *incapability* of cloud storage server to break SLA without being caught.

Cloud storage users have many reasons to not trust in the cloud storage server, for example, the cloud storage server may have incentive to cut cost by shifting data to a cheaper but slower storage, or the cloud storage server may intend to cover up data loss events. Such threat to integrity of data stored in cloud is indeed realistic. Several events about massive loss of Gmail [3] and Hotmail [4] have been reported. There are also plenty of data loss cases that are claimed by individuals but not confirmed officially by the cloud server, e.g. data loss cases in Dropbox [5].

Proofs of Retrievability ($\mathcal{POR}$) model proposed by Juels and Kaliski [6] is among the first few attempts to formulize the notion of "remotely and reliably verifying the data integrity without retrieving the data file". A $\mathcal{POR}$ scheme consists of setup phase and a sequence of verification phases. In the setup phase, a data owner Alice preprocesses her data file using her private key to generate

some authentication information. Then Alice sends the data file together with authentication information to the cloud storage server Bob, and removes them from her local storage. Consequently, in the end of setup phase, Alice only has her private key in her local storage, and Bob has both the data file and the corresponding authentication information. In each subsequent verification phase, Alice generates a random challenge query and Bob is supposed to produce a short response or proof upon the received challenge query, based on Alice's data file and the corresponding authentication information. In the end of a verification phase, Alice will verify Bob's response using her private key and decide to *accept* or *reject* this response.

The performance of a $\mathcal{POR}$ scheme is determined by a few factors: communication bits (i.e. the bit lengths of a challenge and a response) exchanged between Alice and Bob per verification, storage overhead on Alice/Bob's side, computation time on Alice/Bob's side in a verification, and computation time for setup on Alice side.

Several solutions for remote integrity check are proposed. Ateniese *et al.* [2] proposed a solution with $\mathcal{O}(1)$ communication and $1/s$ storage overhead, by generalizing the RSA-based scheme [7], where s is the ratio of the size of a data block to the size of an authentication tag. The drawback of this scheme is that it needs to carry out a large number of exponentiations and is thus computationally intensive in both setup phase and verification phase. Shacham and Waters [1] gave a scheme which is much more efficient in computation (only addition/multiplication and pseudorandom function are required) and has the same $1/s$ storage overhead. However, this scheme requires $\mathcal{O}(s)$ communication bits per verification. Both Ateniese *et al.* [2] and Shacham and Waters [1] adopt some sorts of homomorphic linear authenticator [8] and random sampling to achieve efficiency. It is possible to incorporate the error erasure code into a remote data integrity check scheme, in order to reduce the number of data blocks accessed during one verification, in a similar way and achieving similar tradeoff as in the sublinear online memory checker [9].

1.1 Our result

In Shacham and Waters [1]'s $\mathcal{POR}$ scheme, the size of a proof is dominated by s group elements. We manage to aggregate these s group elements into two group elements, leading to a reduction in proof size from $\mathcal{O}(s)$ to $\mathcal{O}(1)$, by exploiting an intriguing property of polynomial, which is recently used by Kate [10] to construct a constant size polynomial commitment.

Our contributions can be summarized as below.

- We propose a new efficient and secure $\mathcal{POR}$ scheme in Section 4. In this scheme, a data block consists of s group elements and a subset of ℓ blocks are accessed in each verification. The storage overhead is $1/s$ of the data file size, and communication cost is $\mathcal{O}(1)$ bits per verification, and the computation cost is $\mathcal{O}(s)$ group exponentiations on the server side (prover) and $\mathcal{O}(\ell)$ group addition/multiplication/PRF on the client (verifier) side.
- We prove that the proposed $\mathcal{POR}$ scheme is secure in Theorem 1 under a variant of Strong Diffie-Hellman Assumption, which is weaker than the original Strong Diffie-Hellman Assumption.
- The empirical study in Section 5 shows that our scheme is practical under reasonable setting.

In a typical setting where an elliptic curve group of size $\lambda = 160$ is used and the system parameter $s = 100$, during each verification, 720 communication bits are exchanged between the data owner and the cloud storage server where 240 bits for challenge and 480 bits for response, and 100 elliptic curve exponentiations are required to generate a response on the cloud storage server side. The small number of communication bits is also desirable in situations where the challenge and response could be piggybacked into packets generated by other services provided the server.

1.2 Related work

Recently, a lot of works [6, 2, 11, 1, 12, 13, 14, 15, 8, 16, 17] have devoted to the study of remote data integrity check. Juels *et al.* [6] presented a strong security model, Ateniese *et al.* [2] gave an efficient scheme which is secure under a weaker $\mathcal{PDP}$ model. Efficient methods using some sorts of homomorphic authentication tag are proposed in [11, 1, 8]. Dynamic-$\mathcal{PDP}$ [13] extends to dynamic setting and public verifiability is exploited in [16] and the privacy issue in public verification is studied in [17]. Recently, Kate *et al.* [10] proposed an efficient commitment scheme for polynomial and Benabbas *et al.* [18] proposed a secure delegation scheme for polynomial. Both schems can be extended to support $\mathcal{POR}$ easily but with limitations. We will elaborate more on Kate *et al.* [10]'s polynomial commitment scheme in Section 3.

The most efficient variant scheme E-PDP in Ateniese *et al.* [2] is suffering the attack by Shacham and Waters [1]. In Ateniese *et al.* [2], the main construction requires the prover to compute the product $\prod_{(i,a_i)\in \mathtt{Chal}} \mathtt{T}_i^{a_i}$ for all tags $\mathtt{T}_i$ selected by the challenge `Chal`. The authors proposed an efficient variant scheme, named E-PDP, by setting all coefficients a_i in the challenge `Chal` as 1, so that only multiplication is involved and expensive exponentiation is avoided. Shacham and Waters [1] presented an attack on E-PDP, such that the adversary can answer correctly a non-negligible fraction of queries, but there exists no extractor that can recover any data block.

1.3 Organization

The rest of this paper is organized as below: Section 2 reviews the security formulation of Proofs of Retrievability. We describe background knowledge on Kate's polynomial commitment scheme in Section 3 and present our main scheme in Section 4. We analyze the performance of the proposed scheme, conduct experiments and report the empirical results in Section 5. After that, we conclude this paper in Section 6.

Section 4 is self-contained and readers may jump to Section 4 directly to read our main construction.

2 Formulation

2.1 Summary of Notations

At the first, we summarize the key notations used in this paper in Table 1.

2.2 System Model

We restate the $\mathcal{POR}$ [6, 1] model as below, with slight modifications on notations. We adopt the 1-round prove-verify version in Juels [6] for simplicity.

Definition 1 ($\mathcal{POR}$ [6, 1]) *A* Proofs Of Retrievability *($\mathcal{POR}$) scheme consists of four algorithms* (KeyGen, DEncode, Prove, Verify)*:*

- $\mathsf{KeyGen}(1^\lambda) \rightarrow (pk, sk)$*: Given security parameter λ, the randomized key generating algorithm outputs a public-private key pair (pk, sk).*
- $\mathsf{DEncode}(sk, \mathbf{M}) \rightarrow (\mathsf{id}_\mathbf{M}, \hat{\mathbf{M}})$*: Given the private key sk and a data file $\mathbf{M}$, the encoding algorithm* DEncode *produces a unique identifier $\mathsf{id}_\mathbf{M}$ and the encoded file $\hat{\mathbf{M}}$.*
- $\mathsf{Prove}(pk, \mathsf{id}_\mathbf{M}, \hat{\mathbf{M}}, C) \rightarrow \psi$*: Given the public key pk, an identifier $\mathsf{id}_\mathbf{M}$, an encoded file $\hat{\mathbf{M}}$, and a challenge query C, the prover algorithm* Prove *produces a proof ψ.*
- $\mathsf{Verify}(sk, \mathsf{id}_\mathbf{M}, C, \psi) \rightarrow$ `accept` *or* `reject`*: Given the private key sk, an identifier $\mathsf{id}_\mathbf{M}$, a challenge query C, and a proof ψ, the deterministic verifying algorithm* Verify *will output either* accept *or* reject.

Table 1: Summary of Key Notations in this paper

Notation	Semantics
$x \xleftarrow{\$} S$	Uniformly randomly choose x from the finite set S.
λ	Group size
s	The number of group elements in a data block. Typically, an authentication tag consists of one group element. So s is also the ratio of the size of a data block to the size of an authentication tag.
ℓ	The number of data blocks accessed during a verification.
n	The number of data blocks in a data file. Thus the size of a data file is $ns\lambda$ bits.
$\vec{\boldsymbol{m}}$	A vector of form $(m_0, m_1, m_2, \ldots, m_{d-1})$, where d is the dimension of vector $\vec{\boldsymbol{m}}$.
$f_{\vec{\boldsymbol{m}}}(x)$	A polynomial $m_0 + m_1 x + m_2 x^2 + \ldots + m_{d-1} x^{d-1}$ of degree $d-1$ with vector $\vec{\boldsymbol{m}}$ as coefficient, where d is the dimension of vector $\vec{\boldsymbol{m}}$.
PRF	Pseudorandom function
$\mathcal{POR}$	Proofs of Retrievability
$\mathcal{PDP}$	Provable Data Possession
EPOR	Efficient Provable Data Possession; it is the name of the scheme proposed in this paper

Completeness. A $\mathcal{POR}$ scheme (KeyGen, DEncode, Prove, Verify) is *complete*, if an honest prover (who ensures the integrity of his storage and executes the procedure Prove to compute a proof) will always be accepted by the verifier. More precisely, for any key pair (pk, sk) generated by KeyGen, and any data file $\mathbf{M}$, any challenge query C, if $\psi \leftarrow \mathsf{Prove}(pk, \mathsf{id}_{\mathbf{M}}, \hat{\mathbf{M}}, C)$, then $\mathsf{Verify}(sk, \mathsf{id}_{\mathbf{M}}, C, \psi)$ outputs accept with probability 1, where $(\mathsf{id}_{\mathbf{M}}, \hat{\mathbf{M}}) \leftarrow \mathsf{DEncode}(sk, \mathbf{M})$.

2.3 Security Model

2.3.1 Trust Model and Scope of Topic In a $\mathcal{POR}$ system, only the data owner is trusted and the cloud storage server is treated as untrusted and potentially malicious.

We clarify that, the following topics are out of the scope of this paper: (1) Support of dynamic operations; (2) Denial of Service Attack; (3) Frame attack. This is not a limitation of our work, since all of these can be achieved using existing methods.

2.3.2 $\mathcal{POR}$ Security Game We rewrite the $\mathcal{POR}$ security game in Juels *et al* [6] and Shacham *et al.* [1] in a standard way. The security game between a *probabilistic polynomial time* (PPT) adversary $\mathcal{A}$ and a PPT challenger $\mathcal{C}$ w.r.t. a $\mathcal{POR}$ scheme $\mathcal{E}$ = (KeyGen, DEncode, Prove, Verify) is as below.

Setup: The challenger $\mathcal{C}$ runs the key generating algorithm KeyGen to obtain public-private key pair (pk, sk), and gives pk to the adversary $\mathcal{A}$.

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

- Store query ($\mathbf{M}$): Given a data file $\mathbf{M}$ chosen by $\mathcal{A}$, the challenger $\mathcal{C}$ responses by running data encoding algorithm $(\mathsf{id}, \hat{\mathbf{M}}) \leftarrow \mathsf{DEncode}(sk, \mathbf{M})$ and sending the encoded data file $\hat{\mathbf{M}}$ together with its identifier id to $\mathcal{A}$.
- Verification query (id): Given a file identifier id chosen by $\mathcal{A}$, if id is the (partial) output of some previous store query that $\mathcal{A}$ has made, then the challenger $\mathcal{C}$ initiates a $\mathcal{POR}$ verification with $\mathcal{A}$ w.r.t. the data file $\mathbf{M}$ associated to the identifier id in this way:
  - $\mathcal{C}$ chooses a random challenge `Chal`;
  - $\mathcal{A}$ produces a proof ψ w.r.t. the challenge `Chal`;
    Note: adversary $\mathcal{A}$ may generate the proof in an arbitrary method rather than applying the algorithm Prove.
  - $\mathcal{C}$ verifies the proof ψ by running algorithm $\mathsf{Verify}(sk, \mathsf{id}, \texttt{Chal}, \psi)$. Denote the output as b.

At the end $\mathcal{C}$ sends the decision bit $b \in \{\mathsf{accept}, \mathsf{reject}\}$ to $\mathcal{A}$ as feedback. Otherwise, if id is not the (partial) output of any previous store query that $\mathcal{A}$ has made, $\mathcal{C}$ does nothing.

Commit: Adversary $\mathcal{A}$ chooses a file identifier id^* among all file identifiers she obtains from $\mathcal{C}$ by making store queries in **Learning** phase, and commit id^* to $\mathcal{C}$. Let $\mathbf{M}^*$ denote the data file associated to identifier id^*.
Retrieve: The challenger $\mathcal{C}$ initiates ζ number of $\mathcal{POR}$ verifications with $\mathcal{A}$ w.r.t. the data file $\mathbf{M}^*$, where $\mathcal{C}$ plays the role of verifier and $\mathcal{A}$ plays the role of prover, as in the **Learning** phase. At the end of each verification, $\mathcal{C}$ provides the decision bits (accept or reject) to $\mathcal{A}$ as feedback. From messages collected in these ζ interactions with $\mathcal{A}$, $\mathcal{C}$ extracts a data file $\mathbf{M}'$ using some PPT extractor algorithm. The adversary $\mathcal{A}$ wins this game, if and only if $\mathbf{M}' \neq \mathbf{M}^*$.

The adversary $\mathcal{A}$ is ϵ-*admissible* [1], if the probability that $\mathcal{A}$ convinces $\mathcal{C}$ to accept in a verification in the **Retrieve** phase, is at least ϵ. We denote the above game as $\mathsf{Game}_{\mathcal{A}}^{\mathcal{E}}(\zeta)$.

Definition 2 ([6, 1]) *A $\mathcal{POR}$ scheme $\mathcal{E}$ is sound, if for any PPT ϵ-admissible adversary $\mathcal{A}$ with non-negligible ϵ, there exists a polynomial ζ, such that the advantage $\mathsf{Adv}_{\mathcal{A}}^{\mathcal{E}}(\zeta)$ defined as below is negligible.*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathcal{E}}(\zeta) \stackrel{\text{def}}{=} \mathsf{Pr}\left[\mathcal{A} \textit{ wins } \mathsf{Game}_{\mathcal{A}}^{\mathcal{E}}(\zeta)\right] \tag{1}$$

2.3.3 Clarification of Security Model There should be no confusion between the security formulation and the real world application of a $\mathcal{POR}$ scheme. We remark that the security games $\mathsf{Game}_{\mathcal{A}}^{\mathcal{E}}$, especially the $\mathsf{Retrieve}$ phase, are only for security formulation, and applications of a $\mathcal{POR}$ scheme do not necessarily follow the description of the security game exactly. For example, in real world applications, the data owner will be the one who chooses the data file, instead of the cloud storage server, and the data owner can retrieve her data file by simply requesting the cloud storage server to send it back.

The $\mathsf{Retrieve}$ phase in the security games just ensures that, in theory, user's file *can* be recovered from multiple verifications with the cloud storage server efficiently (using some PPT extractor algorithm), as long as the cloud storage server can pass a non-negligible fraction of challenge queries. Essentially, a secure $\mathcal{POR}$ scheme provides a mechanism, in which the data owner will be guaranteed that her data file can be efficiently recovered from the server's storage at the moment that a verification is accepted, without *actually* downloading the file from the server. Furthermore, this guarantee is based on the assumption that the cloud storage server is not able to solve some cryptographic hard problems[1], without trusting in the cloud storage server.

2.4 Assumption

Let p and $q = 2p + 1$ be prime. The subgroup $\mathbb{G}$ of quadratic residues in $\mathbb{Z}_q^*$ has order p. A *weaker* variant of Strong Diffie-Hellman Assumption over the group $\mathbb{G}$ is described as below.

Definition 3 (A weaker variant of s-Strong Diffie-Hellman (s-SDH) Assumption [19, 20]) *Let p and $q = 2p + 1$ be prime, and $\mathbb{G}$ be the subgroup of quadratic residues in $\mathbb{Z}_q^*$. Let g be a random generator of $\mathbb{G}$. Let $\alpha \stackrel{\$}{\leftarrow} \mathbb{Z}_p$ be chosen at random. Given as input a tuple $(p, q, T = (g, g^{\alpha}, g^{\alpha^2}, \ldots, g^{\alpha^{s-1}}))$, for any PPT adversary $\mathcal{A}$, for a random $c \stackrel{\$}{\leftarrow} \mathbb{Z}_p \setminus \{-\alpha\}$, the probability $\mathsf{Pr}\left[\mathcal{A}(p, q, T, c) = g^{1/(\alpha+c)}\right]$ is negligible.*

[1] For information-theoretical secure $\mathcal{POR}$ schemes (e.g. [14]), there will be no such assumption.

Recall that in a standard Strong Diffie-Hellman Assumption, the value c is chosen by the adversary $\mathcal{A}$, while in the above variant, the value c is randomly chosen. Therefore, the standard Strong Diffie-Hellman Assumption implies the variant in Definition 3. We remark that when our scheme is alternatively instantiated using elliptic curve, the elliptic curve version of SDH Assumption is required.

3 Background on Constant Size Polynomial Commitment Scheme

Kate *et al.* [10] proposed a constant size commitment scheme for polynomial. Their scheme exploits a simple and elegant algebraic property of polynomials: *For any polynomial $f(x) \in \mathbb{Z}_p[x]$ and for any scalar input $r \in \mathbb{Z}_p$, the polynomial $x - r$ divides the polynomial $f(x) - f(r)$.*

3.1 Brief Description of Kate's Polynomial Commitment Scheme

Let us denote with $f_{\vec{\boldsymbol{m}}}(x) \in \mathbb{Z}_p[x]$ the polynomial with coefficient vector $\vec{\boldsymbol{m}} = (m_0, \ldots, m_{s-1}) \in (\mathbb{Z}_p)^s$, that is, $f_{\vec{\boldsymbol{m}}}(x) \equiv \sum_{j=0}^{s-1} m_j x^j$. Let $\mathbb{G}$ and $\mathbb{G}_T$ be two multiplicative group of prime order p and $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ be a bilinear map.

We brief Kate's commitment scheme [10] as below: In the setup, a trust party chooses a public key $pk := (g, g^{\alpha}, \ldots, g^{\alpha^{s-1}}) \in \mathbb{G}^s$, where g is a generator of group $\mathbb{G}$ and $\alpha \in \mathbb{Z}_p$ is chosen at random and kept secret. In order to commit a polynomial $f_{\vec{\boldsymbol{m}}}(x) := \sum_{j=0}^{s-1} m_j x^j$ with coefficient vector $\vec{\boldsymbol{m}} = (m_0, \ldots, m_{s-1}) \in (\mathbb{Z}_p)^s$, a committer can compute a commitment $\mathcal{C}$ using the public key pk, that is, $\mathcal{C} := \prod_{j=0}^{s-1} \left(g^{\alpha^j}\right)^{m_j} = g^{f_{\vec{\boldsymbol{m}}}(\alpha)} \in \mathbb{G}$, and then publish $\mathcal{C}$. Later, for any scalar $r \in \mathbb{Z}_p$, the committer can compute $y := f_{\vec{\boldsymbol{m}}}(r) \in \mathbb{Z}_p$ and generate a *short* proof ψ (Kate [10] calls it witness) to convince a verifier that y is indeed the correct evaluation of $f_{\vec{\boldsymbol{m}}}(r)$, without revealing the polynomial $f_{\vec{\boldsymbol{m}}}(x)$. The proof (or witness) ψ is generated as below:

- Divide the polynomial $f_{\vec{\boldsymbol{m}}}(x) - f_{\vec{\boldsymbol{m}}}(r)$ with $(x - r)$ using polynomial long division, and denote the coefficient vector of the resulting quotient polynomial as $\vec{\boldsymbol{w}} = (w_0, w_1, \ldots, w_{s-2})$, that is, $f_{\vec{\boldsymbol{w}}}(x) \equiv \frac{f_{\vec{\boldsymbol{m}}}(x) - f_{\vec{\boldsymbol{m}}}(r)}{x - r}$.
- Then compute $\psi := g^{f_{\vec{\boldsymbol{w}}}(\alpha)}$ using the public key pk in the same way as computing $g^{f_{\vec{\boldsymbol{m}}}(\alpha)}$, i.e. $\psi := \prod_{j=0}^{s-2} \left(g^{\alpha^j}\right)^{w_j} = g^{f_{\vec{\boldsymbol{w}}}(\alpha)} \in \mathbb{G}$.

After receiving (r, y, ψ) from the committer, a verifier can verify whether $y \stackrel{?}{=} f_{\vec{\boldsymbol{m}}}(r)$ with the proof ψ and public key $pk = (g, g^{\alpha}, \ldots, g^{\alpha^{s-1}})$ and the public commitment $\mathcal{C}$ of the unknown polynomial $f_{\vec{\boldsymbol{m}}}(x)$, using a bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$:

$$e(g, \mathcal{C}) / e(g, g)^y \quad \stackrel{?}{=} \quad e(\psi, g^{\alpha} / g^r). \tag{2}$$

Note that the left hand side of above equation (2) is $e(g, \mathcal{C}) / e(g, g)^y = e(g, g)^{f_{\vec{\boldsymbol{m}}}(\alpha) - y}$, and the right hand side is $e(\psi, g^{\alpha} / g^r) = e(g^{f_{\vec{\boldsymbol{w}}}(\alpha)}, g^{\alpha - r}) = e(g, g)^{(\alpha - r) f_{\vec{\boldsymbol{w}}}(\alpha)}$.

In summary, the commitment scheme proposed by Kate *et al.* [10] allows the owner of a polynomial $f(x)$ to generate a constant size proof for the correctness of the polynomial evaluation $f(r)$ at *any* particular point $x = r$.

3.2 Proofs of Retrievability Scheme implied by Kate's Polynomial Commitment

We realize that Kate's Polynomial Commitment[2] Scheme [10] (Setup, Commit, CreatWitness, VerifyWitness) immediately implies a $\mathcal{POR}$ scheme (KeyGen, DEncode, Prove, Verify).

[2] In fact, Kate's Polynomial Commitment scheme [10] contains two more algorithms Open and VerifyPoly. Since these two algorithms are not required in our black-box construction of a $\mathcal{POR}$ scheme, we save them.

KeyGen The data owner generates the public key $pk := (g, g^{\alpha}, \ldots, g^{\alpha^{s-1}})$ and private key $sk := \alpha$ by running the algorithm Setup of Kate's Polynomial Commitment Scheme [10].
DEncode The data file is represented as a vector $\vec{\boldsymbol{m}} = (m_0, m_1., \ldots, m_{s-1}) \in (\mathbb{Z}_p)^s$ of dimension s. The data owner computes the commitment $\mathcal{C} := g^{f_{\vec{\boldsymbol{m}}}(\alpha)}$ by running algorithm Commit, sends the file $\vec{\boldsymbol{m}}$ to the cloud storage server and keeps the commitment $\mathcal{C}$ in local storage.
Prove After receiving a random input r from the data owner, the cloud storage server evaluates the polynomial $f_{\vec{\boldsymbol{m}}}(x)$ at point $x = r$ to obtain $y := f_{\vec{\boldsymbol{m}}}(r)$, and generates a witness $\psi := g^{\frac{f_{\vec{\boldsymbol{m}}}(x) - f_{\vec{\boldsymbol{m}}}(r)}{x-r}}$ by running CreatWitness. Then sends response (y, ψ) to the data owner.
Verify The data owner verifies whether ψ is a valid proof w.r.t. $(r, y, \mathcal{C})$ by running algorithm VerifyEval.

3.3 Discussion

The security of the above $\mathcal{POR}$ scheme can be reduced to the security of Kate's commitment scheme. However, the implied $\mathcal{POR}$ scheme suffers from several limitations:

1. The commitment generated using Kate's [10] scheme is not a MAC (message authentication code) or signature, since anyone with the public key can produce a valid commitment for any polynomial (with degree less than s, which is the size of the public key). Consequently, the commitments have to be stored in a trusted storage. Note in $\mathcal{POR}$, the cloud storage is not trusted.
2. The scheme requires to access every bit of the data during each verification, since it treats the whole data file as one block, and does not support random sampling as in previous solutions [6, 1, 2].
3. The second limitation described above may be resolved by exploiting the homomorphism of Kate's polynomial commitment scheme, but with penalty. Kate's polynomial commitment scheme is homomorphic. That is, if $\mathcal{C}_1 = g^{f_{\vec{\boldsymbol{m}}_1}(\alpha)}$ is the commitment of a polynomial $f_{\vec{\boldsymbol{m}}_1}(x)$ and $\mathcal{C}_2 = g^{f_{\vec{\boldsymbol{m}}_2}(\alpha)}$ is the commitment of a polynomial $f_{\vec{\boldsymbol{m}}_2}(x)$, then $C_1 \times C_2 = g^{f_{\vec{\boldsymbol{m}}_1}(\alpha) + f_{\vec{\boldsymbol{m}}_2}(\alpha)} = g^{f_{\vec{\boldsymbol{m}}_1 + \vec{\boldsymbol{m}}_2}(\alpha)}$ is a valid commitment of polynomial $f_{\vec{\boldsymbol{m}}_1 + \vec{\boldsymbol{m}}_2}(x)$. So in a $\mathcal{POR}$ scheme, we may treat the data as a sequence of n blocks, and apply the Kate's scheme on each of them to obtain n commitments. The verification can still be conducted due to the homomorphism. The drawback is that, due to the first limitation described above, all of these n commitments have to be kept in the data owner's local storage. However, in a $\mathcal{POR}$ scheme, the data owner's storage cost is desired to be $O(1)$ during verification and indeed all previous known solutions achieve $O(1)$ storage cost for the data owner. Besides this, generating n commitments can be computationally costly.

4 EPOR: Efficient Proofs of Retrievability Scheme

In this section, we construct an efficient $\mathcal{POR}$ scheme with private verification and name this scheme as EPOR. Our construction integrates Kate's polynomial commitment scheme [10] and Shacham and Waters' $\mathcal{POR}$ scheme [1] (the one with private verification) in a seamless way. We emphasize that EPOR can be instantiated using elliptic curve, although in the following description, EPOR is constructed over a modulo group.

Recall that: (1) the notation $f_{\vec{\boldsymbol{m}}}(x)$ denotes the polynomial with coefficient vector $\vec{\boldsymbol{m}} = (m_0, \ldots, m_{s-1})$, that is, $f_{\vec{\boldsymbol{m}}}(x) \equiv \sum_{j=0}^{s-1} m_j x^j$; (2) our scheme described below exploits an algebraic property of polynomials: for any polynomial $f(x)$ and for any scalar input r, the polynomial $x - r$ divides the polynomial $f(x) - f(r)$.

4.1 Construction

KeyGen(1^λ) $\rightarrow$ (pk, sk)

Choose at random a λ bits safe prime q such that $p = (q-1)/2$ is also prime. Choose at random an element g from $\mathbb{Z}_q^*$ such that the multiplicative order of g is p. Choose at random two elements τ, α from $\mathbb{Z}_p^*$: $\tau, \alpha \xleftarrow{\$} \mathbb{Z}_p^*$, Choose at random a PRF key, denoted as seed. The public key is $pk := (p, \{g^{\alpha^j} \mod q\}_{j=0}^{s-1})$ and the private key is $sk := (p, \mathsf{seed}, \alpha, \tau)$.

DEncode(sk, M) $\rightarrow$ (id, $\hat{\mathbf{M}}$)

Given the file $\mathbf{M}$, first apply the error erasure code on it; then split the error erasure encoded file into n vectors (or blocks), such that each vector consists of s elements from $\mathbb{Z}_p$: $\{\vec{\boldsymbol{m}}_i = (m_{i,0}, \ldots, m_{i,s-1}) \in \mathbb{Z}_p^s\}_{0 \le i \le n-1}$. Choose a unique identifier id from domain $\{0,1\}^\lambda$. For each $\vec{\boldsymbol{m}}_i$, $0 \le i \le n-1$, compute an authentication tag t_i as below

$$t_i := \mathsf{PRF}_{\mathsf{seed}}(\mathsf{id}, i) \ + \ \tau f_{\vec{\boldsymbol{m}}_i}(\alpha) = \mathsf{PRF}_{\mathsf{seed}}(\mathsf{id}, i) \ + \ \tau \sum_{j=0}^{s-1} m_{i,j}\alpha^j \mod p.$$

The final encoded file $\hat{\mathbf{M}}$ consists of $\{\vec{\boldsymbol{m}}_i = (m_{i,0}, \ldots, m_{i,s-1})\}$, $0 \le i \le n-1$, together with authentication tags $\{t_i\}$, $0 \le i \le n-1$.

Prove(pk, id, $\hat{\mathbf{M}}$, (r, C)) $\rightarrow$ (y, ψ, σ)

The challenge is (r, C), where $r \xleftarrow{\$} \mathbb{Z}_p^*$ is a random nonce chosen from $\mathbb{Z}_p^*$, and C is a set $\{(i, \nu_i)\}$ of ℓ index-weight pairs (i, ν_i)'s, where each index $i \in [0, n-1]$, each weight $\nu_i \in B \subseteq [1, 2^\lambda]$, and i's are distinct (*Note: The set B is a system parameter as in [1]* and will be used in the proof. See Appendix B.1). Compute

$$\mu_j := \sum_{(i,\nu_i)\in C} \nu_i m_{i,j} \mod p \qquad \text{for } 0 \le j \le s-1, \tag{3}$$

$$\sigma := \sum_{(i,\nu_i)\in C} \nu_i t_i \mod p. \tag{4}$$

Let vector $\vec{\boldsymbol{\mu}} := (\mu_0, \ldots, \mu_{s-1})$ (*Note:* $\vec{\boldsymbol{\mu}} = \sum_{(i,\nu_i)\in C} \nu_i \vec{\boldsymbol{m}}_i$). Evaluate polynomial $f_{\vec{\boldsymbol{\mu}}}(x)$ at point $x = r$ to obtain $y := f_{\vec{\boldsymbol{\mu}}}(r) \mod p$. Divide the polynomial $f_{\vec{\boldsymbol{\mu}}}(x) - f_{\vec{\boldsymbol{\mu}}}(r)$ with $(x - r)$ using polynomial division, and denote the coefficients vector of the resulting quotient polynomial as $\vec{\boldsymbol{w}} = (w_0, \ldots, w_{s-2})$, that is, $f_{\vec{\boldsymbol{w}}}(x) \equiv \frac{f_{\vec{\boldsymbol{\mu}}}(x) - f_{\vec{\boldsymbol{\mu}}}(r)}{x-r}$. Compute ψ with the public key $pk = (p, \{g^{\alpha^j} \mod q\}_{j=0}^{s-1})$ as below

$$\psi := \prod_{j=0}^{s-2} \left(g^{\alpha^j}\right)^{w_j} = g^{f_{\vec{\boldsymbol{w}}}(\alpha)} \mod q. \tag{5}$$

Output (y, ψ, σ).

Verify(sk, id, (r, C), (y, ψ, σ)) $\rightarrow$ accept or reject

Parse C as a set $\{(i, \nu_i)\}$ of ℓ index-weight pairs (i, ν_i)'s, where each index $i \in [0, n-1]$, each weight $\nu_i \in B \subseteq [1, 2^\lambda]$, and i's are distinct. Verify the following equality Eq (6) with the private key $sk = (p, \mathsf{seed}, \alpha, \tau)$. If it holds, then output accept; otherwise, output reject.

$$\psi^{\alpha - r} \ \stackrel{?}{=} \ g^{\tau^{-1}\left(\sigma - \sum_{(i,\nu_i)\in C} \nu_i \mathsf{PRF}_{\mathsf{seed}}(\mathsf{id}, i)\right) \ - \ y} \mod q \tag{6}$$

4.2 Discussion

- The challenge $C = \{(i, \nu_i) : i \in [n], \nu_i \in B \subseteq [1, 2^\lambda]\}$ can be represented compactly by two short random seeds using a pseudorandom function. Alternatively, Dodis *et al.* [14]'s PoR Code can be applied.
- As mentioned in Section 3, Kate's commitment scheme [10] can be used as a secure $\mathcal{POR}$ scheme. Compared to the $\mathcal{POR}$ scheme implied by Kate's commitment scheme, our construction resolves the limitations described in Section 3.3:
  - The above scheme EPOR turns Kate's commitment into an authentication tag (i.e. a message authentication code). As a result, all such authentication tags can be stored in the untrusted cloud storage, and the cloud storage server is unable to forge new authentication tags.
  - Scheme EPOR splits the data file into a sequence of blocks and supports random sampling during verification as in previous solutions [1, 2], that is, in each verification, only a small number of data blocks are selected and accessed for integrity check. Consequently, the computation cost during verification is greatly reduced. Note that due to the application of error erasure code, such random sampling does not sacrifice the error detection probability of a $\mathcal{POR}$ scheme, compared with the case that every bit is accessed during verification.
  - During the data encoding (the algorithm DEncode), only group addition/multiplication and PRF (pseudorandom function) evaluation are required, and the more expensive exponentiations in generating Kate's commitments are avoided. During verification, the data owner only keeps the secret key in local storage which has size in $\mathcal{O}(1)$.
- Compared to Shacham and Waters [1]' scheme, the algorithm Prove in EPOR is able to aggregate s number of weighted sums $\mu_0, \mu_1, \ldots, \mu_{s-1}$ into two numbers y and ϕ using the idea in Kate's polynomial commitment scheme, where $y = f_{\vec{\mu}}(r) = \sum_{j=0}^{s-1} \mu_j r^j \in \mathbb{Z}_p$ (r is a random nonce chosen by the data owner) and $\psi = g^{\frac{f_{\vec{\mu}}(\alpha) - f_{\vec{\mu}}(r)}{\alpha - r}} \in \mathbb{Z}_q^*$. In this way, EPOR requires only $\mathcal{O}(1)$ communication bits per verification. In comparison, the Shacham and Waters [1] scheme requires $\mathcal{O}(s)$ communication bits per verification, since $(\mu_0, \mu_1, \ldots, \mu_{s-1})$ are sent back directly as the response.

4.3 Security

Theorem 1 *The proposed scheme* EPOR *is a complete and sound $\mathcal{POR}$ scheme as defined in Section 2, if the SDH Assumption in Definition 3 holds and* PRF *is cryptographic secure pseudorandom function.*

The full proof is given in the Appendix.

5 Performance Analysis and Experiment

In this section, we analyze the performance of our proposed scheme EPOR in communication, storage, computation and false accept rate, and report the empirical results based on our prototype implementation. We also compare our scheme with existing works by Shacham and Waters [1] and Ateniese *et al.* [2]. We remark that, although our implementation adopts the modulo group of size 1024 due to our time constraint, our scheme can use elliptic curve with much smaller group size 160.

5.1 Performance Analysis

5.1.1 Communication During a verification, the communication cost is the size of a challenge plus the size of its corresponding response (or proof). In our scheme EPOR, the challenge consists a set C of index-weight pairs and a random group element r. As mentioned previously, the set C can be represented compactly with two 80 bits PRF seeds. The group element r is used to retrieve a polynomial function value $f(r)$ for some polynomial $f(x)$ determined by a linear combination of the data blocks specified in the set C. In the security analysis, the goal of r is to retrieve multiple function values $f(r_i)$'s for different inputs r_i, and then recover the polynomial $f(x)$ by solving a linear equation system. For this reason, we can simply choose r from a smaller range $[0, 2^{80} - 1]$ without any sacrificing in the security. As a result, the challenge size is $80 \times 3 = 240$ bits.

In our scheme EPOR, a response, i.e. the proof, consists of three group elements y, σ, ψ, which are derived from the challenge, the data blocks and authentication tags. So the size of a response is 3λ bits. Therefore, the communication cost per verification is $3\lambda + 240$ bits.

5.1.2 Storage During verification, the data owner only keeps the private key in her local storage. The size of private key is $3\lambda + 80$ bits.

The storage overhead (due to authentication tags) on the cloud storage server side is $1/s$ of the data file size, where the system parameter s is the block size and equals to the ratio of the size of a data block to the size of an authentication tag. The public key is also kept in the cloud storage and its size is $(s+1)\lambda$ bits. Note that in our scheme, there is only one public key per user, without regarding to the number of files the user stores in the cloud storage server.

5.1.3 Computation Our scheme is very efficient in setup. Key generation requires s number of group exponentiations. Suppose a $ns\lambda$ bits data file consists of n data blocks, each block has s group elements and each group element has λ bits. The data preprocess (i.e. the DEncode algorithm) requires only ns number of group multiplications and additions, together with n PRF evaluations. Note that the PRF is simulated with an AES stream cipher.

During a verification, the computation complexity on the cloud storage server side is dominated by the computation of ψ in Equation (5) in the algorithm Prove on page 8. This dominant step takes $(s - 1)$ number of group exponentiations, and is the bottleneck of efficiency of our scheme when the block size s becomes large.

5.1.4 False Positive Rate Recall that, error erasure code is applied at the beginning of the algorithm DEncode. Suppose a rate-ρ Reed-Solomon code is adopted, that is, any ρ fraction of data blocks in the encoded file can recover the original file, and the ratio of the size of encoded file to the original is $1/\rho$. If an encoded file is corrupted such that it is unable to recover the original using the erasure decoding, then more than $1 - \rho$ fraction of data blocks are corrupted. In this case, a randomly chosen data block is not corrupted with probability smaller than ρ, and the probability[3] that ℓ independently randomly chosen data blocks will not hit any corrupted data block is smaller than ρ^ℓ, independent on the file size. Our scheme guarantees that if a corrupted data block is hit in a verification, then the data owner will accept with only negligible probability. So the false accept rate is smaller than ρ^ℓ, if ℓ independently random blocks are accessed in a verification.

We list out the false accept rate w.r.t. various challenge size and various erasure code rate in Table 2. The choices of value of challenge size ℓ is 100, 300, 500, or 700; the choices of erasure encode

[3] Note that this argument is based on the case of choosing indices of data blocks at random *with* replacement. The other case where choosing indices at random *without* replacement will have a larger error detection rate.

rate ρ is 0.99 or 0.98. Note that the the storage overhead due to erasure encoding is $1/0.99 - 1 \approx 0.0101$ of original file size, if $\rho = 0.99$; $1/0.98 - 1 \approx 0.0204$, if $\rho = 0.98$.

Table 2: The False Accept Rate Versus Challenge Size and Erasure code rate. Recall that the challenge size ℓ represents the number of data blocks accessed in a verification.

Challenge Size	**False Accept Rate ρ^ℓ** with $\rho = 0.99$	**False Accept Rate ρ^ℓ** with $\rho = 0.98$
$\ell = 100$	0.366032341	0.132619556
$\ell = 300$	0.049040894	0.002332506
$\ell = 500$	0.006570483	0.000041024
$\ell = 700$	0.000880311	0.000000722

5.1.5 Recommended System Parameters We recommend the following system parameter for our proposed scheme EPOR: The error erasure rate is 0.98, block size s is around 160, the challenge size is around 500. In this setting, the false accept rate is 4.1024×10^{-5}, the number of communicaiton bits required in a verification is 720 for elliptic curve group or 3312 for modulo group, the storage overhead is about 2% due to erasure encoding and $1/160 = 0.625\%$ due to authentication tags. Our experiment will confirm that the query latency is within 1 second.

5.1.6 Comparison We give a comparison on the performances of our scheme with Shacham and Waters [1] and Ateniese *et al.* [2] in Table 3 with an example. A more detailed and generic comparison is given in Table 4 on page 12. Note that in our proposed scheme EPOR, the practical choice of value s is bounded by the computation on the server side, which is similar to the case of Ateniese [2]. In contrast, in SW [1], the largest practical value of s is limited by the communication requirement.

Table 3: Comparison with an example among the $\mathcal{PDP}$ scheme by Ateniese *et al.* [2], the $\mathcal{POR}$ scheme by Shacham and Waters [1], and the $\mathcal{POR}$ scheme named EPOR proposed in this paper. After erasure encoding, the file size is 1GB, block size is $s = 100$, and storage overhead due to authentication tags is about 10MB for all schemes. For all schemes listed below, we assume that, during a verification, the challenge $C = \{(i, \nu_i)\}$ are represented by two 80 bits PRF seeds. System parameter ℓ represents the size of set C. All computation times are represented by the corresponding dominant factor. `exp` and `mul` denote the group exponentiation and group multiplication respectively in the corresponding group. Note that one 1024 bits modular exponentiation or one 160 bits elliptic curve exponentiation takes roughly 5 millisecond in a standard PC.

Scheme	**Group Size**	**Communication bits**	**Computation (Data Preprocess)**	**Computation (Prove)**
Ateniese [2]	$\lambda = 1024$	$2\lambda + 320 = 2368$	2^{23} `exp`. over $\mathbb{Z}_N^*$	$(100 + \ell)$ `exp`. over $\mathbb{Z}_N^*$
Shacham and Water [1]	$\lambda = 80$	$(s+1)\lambda + 160 = 8240$	2^{27} `mul`. over $\mathbb{Z}_p$	100ℓ `mul`. over $\mathbb{Z}_p$
EPOR (ECC)	$\lambda = 160$	$3\lambda + 240 = 720$	2^{26} `mul`. over $\mathbb{Z}_p$	100 `exp`. over Elliptic Curve
EPOR ($\mathbb{Z}_p$)	$\lambda = 1024$	$3\lambda + 240 = 3312$	2^{23} `mul`. over $\mathbb{Z}_p$	100 `exp`. over $\mathbb{Z}_p^*$

Table 4: Performance Comparison. All schemes support private verification only In each scheme, a challenge set $C = \{(i, \nu_i)\}$ contains ℓ index-coefficient pairs and is represented compactly by two 80 bits PRF seeds. In the table, `exp`, `mul` and `add` represent exponentiation, multiplication and addition in the corresponding groups/fields; notation $|\mathsf{F}|$ denotes the file size in bits. Recall that the notations λ, s, ℓ, n are as described in Table 1 in Section 2. *Note: (1) Ateniese* et al. *[2,21] withdraws the claim of public verifiability in order to fix a flaw in their proof. (2) In Ateniese's $\mathcal{PDP}$ scheme, exponentiation with a large integer of size $s\lambda$ is required. We represent such exponentiation as a number of s normal group exponentiation* `exp`*, where the exponent is λ bits long. Similar for the RSA based scheme.*

Scheme	Finite Field Size (bits)	Communication (bits)	Storage Overhead	Computation (Prover)	Computation (Verifier)	Computation (Data Preprocess)
RSA-based scheme [7]	$\lambda = 1024$	2λ	Zero	$\lvert\mathsf{F}\rvert/\lambda$ exp.	1 exp.	1 exp.
PolyCommit [10] (Elliptic Curve)	$\lambda = 160$	3λ	Zero	$\lvert\mathsf{F}\rvert/\lambda$ exp. + $2\lvert\mathsf{F}\rvert/\lambda$ (mul. + add.)	2 pairing	$\lvert\mathsf{F}\rvert/\lambda$ (mul. + add.) + 1 exp.
PolyDelgegation [18]	$\lambda = 160$	$2\lambda + 240$	$\lvert\mathsf{F}\rvert$	$\lvert\mathsf{F}\rvert/\lambda$ (exp. + mul. + add.)	2 exp.	$\lvert\mathsf{F}\rvert/\lambda$ (exp. + mul. + add.)
Ateniese [2]	$\lambda = 1024$	$2\lambda + 320$	$\lvert\mathsf{F}\rvert/s$	$(\ell + s)$ exp. + 2ℓ mult. + ℓ add + 1 hash + 2ℓ PRF	ℓ (exp. + mult.) + 1 hash	ns exp. + n hash
S.W. [1](Private Verification)	$\lambda = 80$	$(s+1)\lambda + 160$	$\lvert\mathsf{F}\rvert/s$	$s\ell$ (add + mult) + 2ℓ PRF	$(\ell + s)$ (add + mult) + 3ℓ PRF	$\lvert\mathsf{F}\rvert/\lambda$ (mul. + add.) + $\lvert\mathsf{F}\rvert/(\lambda s)$ PRF
EPOR (Elliptic Curve)	$\lambda = 160$	$3\lambda + 240$	$\lvert\mathsf{F}\rvert/s$	$(s-1)$ exp. + $(s\ell + s + \ell)$ (add + mul) + 2ℓ PRF	2 exp. + ℓ (add + mult) + 3ℓ PRF	$\lvert\mathsf{F}\rvert/\lambda$ (mul. + add.) + $\lvert\mathsf{F}\rvert/(\lambda s)$ PRF
EPOR ($\mathbb{Z}_p$)	$\lambda = 1024$	$3\lambda + 240$	$\lvert\mathsf{F}\rvert/s$	$(s-1)$ exp. + $(s\ell + s + \ell)$ (add + mul) + 2ℓ PRF	2 exp. + ℓ (add + mult) + 3ℓ PRF	$\lvert\mathsf{F}\rvert/\lambda$ (mul. + add.) + $\lvert\mathsf{F}\rvert/(\lambda s)$ PRF

We also compare the proposed scheme EPOR with Shacham and Waters' scheme [1] in communication and storage overhead. For a 1GB data file, we plot the number of communication bits (i.e. the size of a challenge and a proof) against the storage overhead for both schemes in Figure 1.

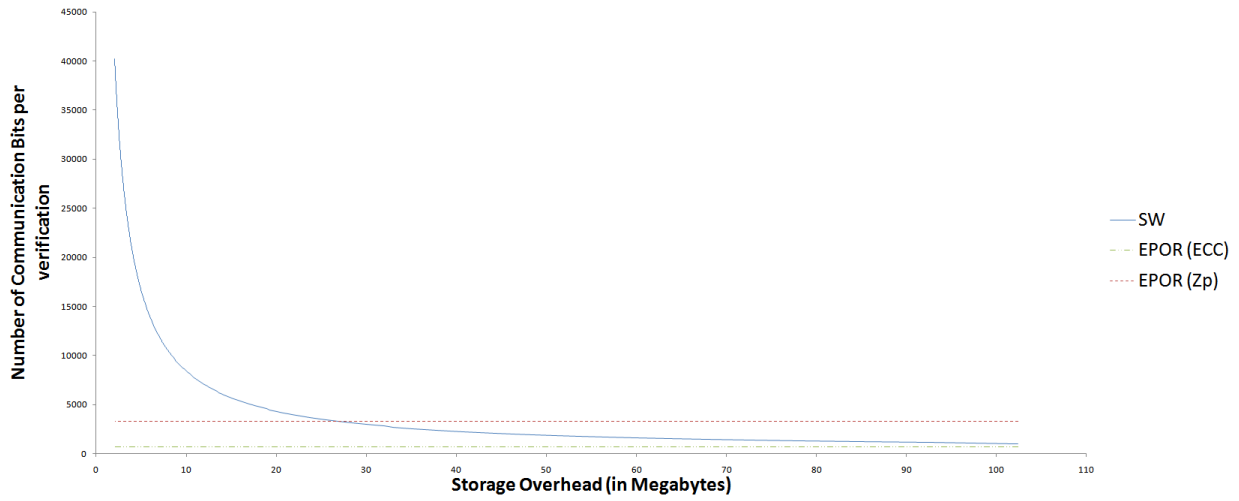


Fig. 1: Comparison on communication (in bits) and storage overhead (in megabytes) w.r.t. a 1GB data file. **SW** denotes the $\mathcal{POR}$ scheme with private verification by Shacham and Waters [1]; EPOR (ECC) denotes our proposed scheme instantiated over elliptic curve group; EPOR ($\mathbb{Z}_p$) denotes our proposed scheme instantiated over group $\mathbb{Z}_p$. Note: In this comparison, the size of public key is counted as a part of storage overhead for EPOR.

5.2 Experiment

The goal of this experiment is to measure the actual running time of the four algorithms KeyGen, DEnc, Prove, Verify in the proposed EPOR scheme, with disk IO time included and networking communication time excluded. Note that the reported query latency includes of running time of Prove and Verify and disk IO time.

5.2.1 Experiment Environment and Setting We have implemented a prototype of our EPOR scheme in C programming language. The large integer arithmetic is computed using GNU MP [22] library version 5.0.1. The pseudorandom function PRF are simulated with AES symmetric cipher provided in OpenSSL [23] library version 1.0.0d. The disk IO is handled by C library function `mmap`. We observe a low memory consumption for all experiments conducted. Our implementation is not optimized and further performance improvements of our scheme can be expected.

Our test machine is a laptop computer, which is equipped with a 2.5GHz Intel Core 2 Duo CPU (model T9300), a 3GB PC2700-800MHZ RAM and a 7200RPM hard disk. The test machine runs 32 bits version of Gentoo Linux OS with kernel 2.6.36. The file system is EXT4 with 4KB page size.

Our test data files are of size 16MB, 32MB, 64MB, 128MB, 256MB and 512MB, respectively (We assume these are the file sizes after error erasure encoding). The choices of values of various system parameters, i.e. group size λ, block size s and challenge size ℓ, are listed in Table 5.

Table 5: The choices of values of various system parameters in our experiment

System Parameter	Semantics	Choices of Values
λ	Group Size	1024
s	Block size, i.e. the number of group elements in a data block	40, 80, 160, 320, 640, or 960
ℓ	Challenge size, i.e. the number of data blocks accessed in a verification	100, 300, 500 or 700.

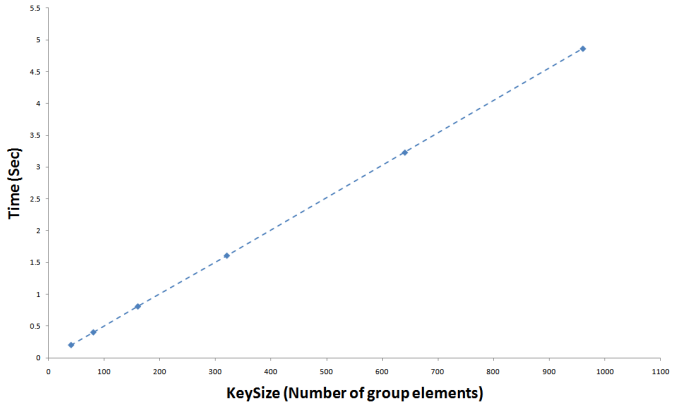
Our experiments are conducted in this way:

- Key generation: For each choice of block size s, we generate a key pair with size s using the key generating program KeyGen. The generated public key consists of s group elements.
- Data preprocess: For each test file, for each choice of value of block size s, we run the data encoding program DEncode to generate a set of authentication tags.
- Verification: For each test file, for each choice of value of block size s, for each choice of value of challenge size ℓ, we run the Prove and Verify programs to simulate the interaction between the data owner and cloud storage server.

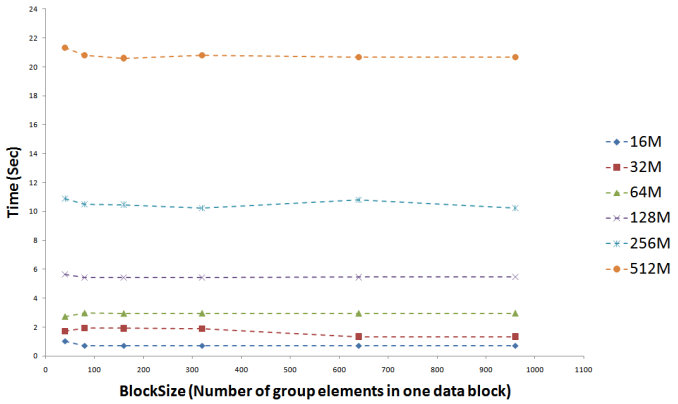
Every single experiment case is repeated for 10 times and the reported timing data are the averages. We remark that experiment trials are run in sequence without parallelization.

5.2.2 Experiment Results The experiment results are showed in Figure 2. All experiment results are averaged over 10 trials. Since all experiment results vary little across different trials, we do not report the variances or confidence intervals.

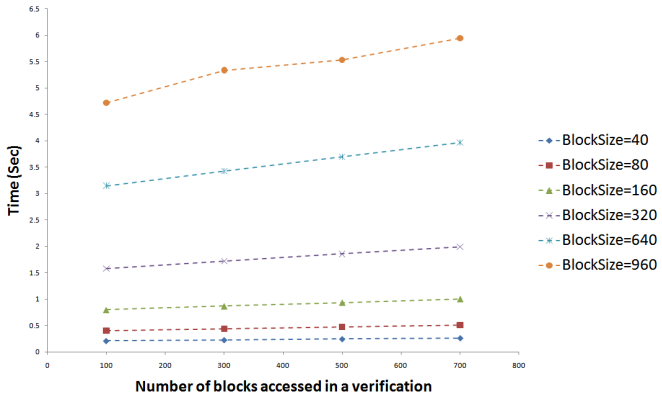
Our experiment result in Figure 2(a) indicates that the key generating time is proportional to the key size, i.e. the number of group elements in a key. The experiment result in Figure 2(b) indicates that the data preprocess time (particularly, DEncode) is proportional to the data file size and almost independent on the block size s. The experiment also shows that the query latency is



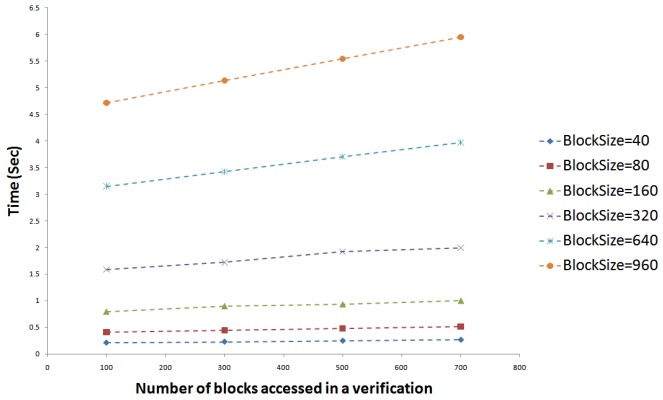
(a) Time to generate a key VS the key size



(b) Data preprocess time VS the block size. Each line is labeled with the size (in megabytes) of corresponding data file.



(c) Query latency VS the challenge size for a 128MB data file. Each line is labeled with the corresponding block size.



(d) Query latency for a 512MB data file.

Fig. 2: The subfigure (c) and (d) represent the results of the same experiment w.r.t. different data files, where (c) for a 128MB data file and (d) for a 512MB data file. The key size is the number of group elements in a key; the block size is the number of group elements in one data block; the challenge size is the number of data blocks accessed during one verification. All time measurements include disk IO time, but do not include network communication time. Our test data files are of size 16MB, 32MB, 64MB, 128MB, 256MB and 512MB, respectively. Various block sizes include 40, 80, 160, 320, 640, and 960. Various challenge sizes include 100, 300, 500 and 700. Our test machine is a laptop computer, equipped with a 2.5GHz Intel Core 2 Duo CPU (model T9300), a 3GB PC2700-800MHZ RAM and a 7200RPM hard disk. Our program utilizes only one CPU core.

proportional to the block size s, almost independent on the file size, and grows very slowly with the challenge size ℓ, suggesting that the computation of exponentiations becomes the bottleneck when s is so large. All of these results agree with our analysis.

6 Conclusion

We proposed an efficient and secure $\mathcal{POR}$ scheme. Our scheme requires only a constant number of communication bits (particularly 720 bits when elliptic curve is used) per verification and $1/s$ storage overhead, where s can be as large as hundreds. The small number of communication bits in a verification makes it possible to piggyback the challenge and/or response of our scheme into other communication packets between the data owner and the cloud storage server if any.

References

1. Shacham, H., Waters, B.: Compact Proofs of Retrievability. In: ASIACRYPT. (2008) 90–107
2. Ateniese, G., Burns, R., Curtmola, R., Herring, J., Kissner, L., Peterson, Z., Song, D.: Provable data possession at untrusted stores. In: CCS '07: ACM conference on Computer and communications security. (2007) 598–609
3. Google: Gmail back soon for everyone `http://gmailblog.blogspot.com/2011/02/gmail-back-soon-for-everyone.html`.
4. Microsoft: Hotmail email access issue now resolved `http://windowsteamblog.com/windows_live/b/windowslive/archive/2011/01/03/hotmail-email-access-issue-now-resolved.aspx`.
5. Dropbox: Dropbox forums on data loss topic `http://forums.dropbox.com/tags.php?tag=data-loss`.
6. Juels, A., Kaliski, Jr., B.S.: Pors: proofs of retrievability for large files. In: CCS '07: ACM conference on Computer and communications security. (2007) 584–597
7. Deswarte, Y., Quisquater, J.J., Saïdane, A.: Remote Integrity Checking: How to Trust Files Stored on Untrusted Servers . In: Proceeding of the Conference on Integrity and Internal Control in Information Systems. (2003) 1–11
8. Ateniese, G., Kamara, S., Katz, J.: Proofs of Storage from Homomorphic Identification Protocols. In: ASIACRYPT '09: International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology. (2009) 319–333
9. Naor, M., Rothblum, G.N.: The complexity of online memory checking. J. ACM **56** (2009) 2:1–2:46
10. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-Size Commitments to Polynomials and Their Applications. In: ASIACRYPT. (2010) 177–194
11. Chang, E.C., Xu, J.: Remote Integrity Check with Dishonest Storage Server. In: ESORICS '08: European Symposium on Research in Computer Security: Computer Security. (2008) 223–237
12. Curtmola, R., Khan, O., Burns, R., Ateniese, G.: MR-PDP: Multiple-Replica Provable Data Possession. In: ICDCS '08: International Conference on Distributed Computing Systems. (2008) 411–420
13. Erway, C., Küpçü, A., Papamanthou, C., Tamassia, R.: Dynamic provable data possession. In: CCS '09: ACM conference on Computer and communications security. (2009) 213–222
14. Dodis, Y., Vadhan, S., Wichs, D.: Proofs of Retrievability via Hardness Amplification. In: Proceedings of the 6th Theory of Cryptography Conference on Theory of Cryptography. TCC '09 (2009) 109–127
15. Bowers, K.D., Juels, A., Oprea, A.: HAIL: a high-availability and integrity layer for cloud storage. In: CCS '09: ACM conference on Computer and communications security. (2009) 187–198
16. 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: European conference on Research in computer security. (2009) 355–370
17. Wang, C., Wang, Q., Ren, K., Lou, W.: Privacy-Preserving Public Auditing for Data Storage Security in Cloud Computing. In: IEEE INFOCOM. (2010) 525–533
18. Benabbas, S., Gennaro, R., Vahlis, Y.: Verifiable Delegation of Computation over Large Datasets. In: CRYPTO. (2011) 110
19. Boneh, D., Boyen, X.: Short Signatures Without Random Oracles. In: Advances in Cryptology—EUROCRYPT 2004. (2004) 56–73
20. Boneh, D., Boyen, X.: Short signatures without random oracles and the sdh assumption in bilinear groups. J. Cryptol. **21** (2008) 149–177
21. Ateniese, G., Burns, R., Curtmola, R., Herring, J., Kissner, L., Peterson, Z., Song, D.: Provable Data Possession at Untrusted Stores. Cryptology ePrint Archive, Report 2007/202 (2007) `http://eprint.iacr.org/`.
22. GMP: The GNU Multiple Precision Arithmetic Library `http://www.gmplib.org/`.
23. OpenSSL: OpenSSL Project `http://www.openssl.org/`.
24. Gennaro, R., Gentry, C., Parno, B.: Non-interactive Verifiable Computing: Outsourcing Computation to Untrusted Workers. In: CRYPTO '10: Annual International Cryptology Conference on Advances in Cryptology. (2010) 465–482
25. Chung, K.M., Kalai, Y., Vadhan, S.P.: Improved Delegation of Computation Using Fully Homomorphic Encryption. In: CRYPTO '10: Annual International Cryptology Conference on Advances in Cryptology. (2010) 483–501

A Concept of *Valid Proof* versus *Correct proof*

For the sake of presentation, we clarify two distinct concepts *valid proof* and *correct proof*.

Valid Proof : A proof is valid, if it is accepted by the verifier, i.e. the verifier algorithm outputs accept when taking the proof as input.

Correct Proof : A proof is correct, if it is the same as the one generated by an honest prover on the same query, i.e. the correct proof is the output of the Prove algorithm.

If a $\mathcal{POR}$ scheme is correct, then all correct proofs are valid proofs, but may not vice versa.

B Shacham and Waters' Proof Framework

Shacham and Waters [1] provided a modular proof framework for their proposed $\mathcal{POR}$ schemes. Informally, the proof framework consists of three parts as below:

- **Systems unforgeability:** If a proof generated by an adversarial prover is *valid* (i.e. the proof is accepted by the verifier), then the proof has to be *correct* (i.e. the proof is the same as the one generated by an honest prover w.r.t. the same challenge), except a negligible probability; this part is proved using cryptographic techniques.
- **Extractability:** Given an adversarial prover that can provide correct proofs for a non-negligible fraction of challenge queries, an extractor can collect sufficient number of correct proofs for different queries and recover a large fraction of data file; this part is proved using combinatorial techniques.
- **Retrievability:** Using error erasure decoding algorithm (e.g. Reed-Solomon codes), the original data file can be recovered from a small portion of correct data; this part is proved using coding-theoretical techniques.

Interestingly, the full proofs for all schemes proposed by Shacham and Waters [1] only differ in the first part, and share the second and third parts. Furthermore, we find that proof for our schemes inherit this property, i.e. our proofs also share the second (with minor modifications) and the third parts proof with Shacham and Waters [1].

Next, we quote the results for part two and part three proof in Shacham and Waters [1] as below.

B.1 Theorem for Part-Two Proof

Theorem 2 (Shacham and Waters [1], Theorem 4.3) *Let $\mathcal{A}^{\mathsf{Retrieve}}$ denote the adversary $\mathcal{A}$ in the* Retrieve *phase of the security game $\mathsf{Game}^{\mathcal{E}}_{\mathcal{A}}$, and F be the n-block file chosen by $\mathcal{A}$ in the* Commit *phase. Suppose $\mathcal{A}^{\mathsf{Retrieve}}$ on file F is well-behaved: i.e. any valid proof generated by $\mathcal{A}^{\mathsf{Retrieve}}$ has to be correct (with probability 1), and is ϵ-admissible: i.e. convincingly answers an ϵ fraction of verification queries. Let $\omega := 1/\#B + (\rho n)^{\ell}/(n-\ell+1)^{\ell}$. Then, provided that $\epsilon - \omega$ is positive and non-negligible, it is possible to recover a ρ fraction of the encoded file blocks in $\mathcal{O}(n/(\epsilon-\omega))$ interactions with $\mathcal{A}^{\mathsf{Retrieve}}$ and in $\mathcal{O}(n^2 s + (n+\epsilon n^3)/(\epsilon-\omega))$ time overall.*

Here, B denotes the domain of the coefficients in a verification challenge query and $\#B$ denotes the size of B.

B.2 Theorem for Part-Three Proof

Theorem 3 (Shacham and Waters [1], Theorem 4.8) *Given a ρ fraction of the n blocks of an encoded file F^*, it is possible to recover the entire original file F with all but negligible probability.*

C Scheme EPOR is secure

C.1 The proposed $\mathcal{POR}$ scheme EPOR is complete

Proof. Suppose $\psi = g^{f_{\vec{w}}(\alpha)} \mod p$, $\sigma = \sum_{(i,\nu_i)\in C} \nu_i t_i \mod p$, $y = f_{\mu}(r) \mod p$. Then the LHS (left hand side) of Equation (6) is

$$LHS = \left(g^{f_{\vec{w}}(\alpha)}\right)^{\alpha - r} = g^{f_{\vec{w}}(\alpha)\times(\alpha-r)} = g^{\frac{f_{\vec{\mu}}(\alpha)-f_{\vec{\mu}}(r)}{\alpha-r}\times(\alpha-r)} = g^{f_{\vec{\mu}}(\alpha)-f_{\vec{\mu}}(r)} \mod q. \tag{7}$$

The RHS (right hand side) of Equation (6) is

$$RHS = g^{\tau^{-1}\left(\sum_{(i,\nu_i)\in C}\nu_i t_i \;-\; \sum_{(i,\nu_i)\in C}\nu_i \mathsf{PRF}_{\mathsf{seed}}(\mathsf{id},i)\right) \;-\; y} \tag{8}$$

$$= g^{\tau^{-1}\left(\sum_{(i,\nu_i)\in C}\nu_i (t_i - \mathsf{PRF}_{\mathsf{seed}}(\mathsf{id},i))\right) \;-\; y} \tag{9}$$

$$= g^{\tau^{-1}\left(\sum_{(i,\nu_i)\in C}\;\nu_i \;\cdot\; \tau f_{\vec{\boldsymbol{m}}_i}(\alpha)\right) \;-\; y} \tag{10}$$

$$= g^{\left(f_{\sum_{(i,\nu_i)\in C}\nu_i\vec{\boldsymbol{m}}_i}\;(\alpha)\right) \;-\; y} \tag{11}$$

$$= g^{f_{\vec{\mu}}(\alpha) \;-\; y} \qquad (\text{ Since } \vec{\boldsymbol{\mu}} = \sum_{(i,\nu_i)\in C}\nu_i\vec{\boldsymbol{m}}_i) \tag{12}$$

$$= g^{f_{\vec{\mu}}(\alpha) \;-\; f_{\vec{\mu}}(r)} = LSH \mod q. \tag{13}$$

Note that we obtain Equation (11) from Equation (10), i.e. the equality between $\sum_{(i,\nu_i)\in C}\nu_i f_{\vec{\boldsymbol{m}}_i}(\alpha)$ and $f_{\vec{\boldsymbol{a}}}(\alpha)$ with vector $\vec{\boldsymbol{a}} = \sum_{(i,\nu_i)\in C}\nu_i\vec{\boldsymbol{m}}_i$, through some straightforward algebra manipulation over polynomials. □

C.2 Part One Proof of Scheme EPOR

Here we provide the part one proof for our scheme EPOR. Informally, our part one proof shows: *The response* (y, ψ, σ) *to a challenge query in* EPOR *is unforgeable, in the sense that no PPT adversary can find a valid but not correct proof with non-negligible probability.* First, in Lemma 4, we prove the unforgeability in the no-feedback setting where all accept/reject decisions are kept private from the adversary. Next, in Lemma 5, we prove the unforgeability in the feedback setting where all accept/reject decisions are provided to the adversary.

It is worthy to point out that, we achieve security in feedback setting by lifting the security in no-feedback setting, in contrast with verifiable cloud computing [24,25] that achieve security only in no-feedback setting (for a much larger class of delegated functions). We emphasize that this is possible because our scheme EPOR has an essential difference with [24,25]: Informally, in EPOR, we prove that if the response (y, ψ, σ) is accepted by the verifier, then all of y, ψ, σ are correct (i.e. equal to the corresponding value computed by an honest cloud server); while in the case of [24, 25], their security proof only ensures that the computation result y is correct and cannot ensure whether the proof (ψ, σ) is exactly the one generated by an honest cloud server. In [24, 25], indeed anyone with the public key of the fully homomorphic encryption can output many different proofs (for the correct computation result) such that the verifier will accept all of them.

We emphasize that the below proof has to deal with an special case: when $\alpha - r$ is even, $(-\psi)^{\alpha-r} = \psi^{\alpha-r}$. That is, with non-negligible probability, $(y, -\psi, \sigma)$ is also a valid proof which can pass the Verify algorithm. Our following proof will deal with special case explicitly. An alternative approach is that, in algorithm KeyGen we choose prime q with an additional requirement $q = 3 \mod 4$ and in algorithm Verify we add one more test to check whether ψ is a quadratic residue modulo q. Note that g is a quadratic residue, so is $\psi = g^{f_{\vec{w}}(\alpha)}$. For prime $q = 3 \mod 4$, the negation of a quadratic residue modulo q is always a non-residue. In this way, the verifier will always reject $(y, -\psi, \sigma)$.

Lemma 4 *Suppose that all accept/reject decisions are completely hidden from the adversary. Let* $\mathcal{A}^{\mathsf{Retrieve}}$ *denote the adversary* $\mathcal{A}$ *in the* Retrieve *phase of the security game* $\mathsf{Game}^{\mathcal{E}}_{\mathcal{A}}$*,* (pk, sk) *be the key pair generated by the challenger in the setup phase of game* $\mathsf{Game}^{\mathcal{E}}_{\mathcal{A}}$*,* id *be the identifier chosen by adversary* $\mathcal{A}$ *in the* Commit *phase of game* $\mathsf{Game}^{\mathcal{E}}_{\mathcal{A}}$*,* $\mathbf{M}$ *be the data file associated to* id *and* $(\mathsf{id}, \hat{\mathbf{M}})$ *be the output that* $\mathcal{A}$ *obtains from the challenger upon store query (*$\mathbf{M}$*). For any PPT adversary* $\mathcal{A}$*,*

$$\mathsf{Adv}^{\mathcal{E},\mathsf{forge,no\text{-}fb}}_{\mathcal{A}}(\lambda) = \mathsf{Pr}\left[\begin{array}{l} C := \{(i, \nu_i)\} \xleftarrow{\$} \left([1,n] \times [1, 2^{\lambda}]\right)^{\ell} \text{ with } i \text{ distinct}; \\ (y, \psi, \sigma) \leftarrow \mathsf{Prove}(pk, \mathsf{id}, \hat{\mathbf{M}}, C); \\ (y_0, \psi_0, \sigma_0) \leftarrow \mathcal{A}^{\mathsf{Retrieve}}(pk, \mathsf{id}, C) : \\ \qquad \mathsf{Verify}(sk, \mathsf{id}, C, (y_0, \psi_0, \sigma_0)) = \mathsf{accept} \ \wedge \ (y_0, \psi_0, \sigma_0) \neq (y, \psi, \sigma) \ \wedge \ (y_0, \psi_0, \sigma_0) \neq (y, -\psi, \sigma) \end{array}\right]$$
$$\leq \mathsf{Adv}^{s\text{-}SDH}_{\mathcal{A}} + \frac{1}{p} + N_{PRF} \cdot \mathsf{Adv}^{PRF}_{\mathcal{A}},$$

where N_{PRF} *denotes the number of distinct evaluation of PRF required. The probability is over all random coins and the choice of the challenge query.*

Proof (of Lemma 4).
Game 1. The first game is just the one specified in Lemma 4, i.e. a modified version of security game $\mathsf{Game}^{\mathcal{E}}_{\mathcal{A}}$, such that the adversary $\mathcal{A}$ wins if and only if $\mathcal{A}$ outputs a valid but incorrect $\mathcal{POR}$ proof for a randomly chosen challenge query. We have $\mathsf{Adv}^{\mathcal{E},\mathsf{forge,no\text{-}fb}}_{\mathcal{A}} = \mathsf{Pr}[\mathcal{A} \text{ wins } \mathbf{Game\ 1}]$.
Game 2. The second game is the same as Game 1, except that in the scheme $\mathcal{E}$, the PRF function $\mathsf{PRF}_{\mathsf{seed}}(\cdot)$ is evaluated in the following way:

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

Game 3 The third game is the same as **Game 2**, except that adversary $\mathcal{A}$ wins if and only if $\mathsf{Verify}(sk, \mathsf{id}, C, (y_0, \psi_0, \sigma_0)) = \mathsf{accept}$ and $(y_0, \psi_0, \sigma_0) \neq (y, \psi, \sigma)$ and $(y_0, \psi_0, \sigma_0) \neq (y, -\psi, \sigma)$ and $\sigma_0 = \sigma$.
Game 4 The fourth game is the same as **Game 2**, except that adversary $\mathcal{A}$ wins if and only if $\mathsf{Verify}(sk, \mathsf{id}, C, (y_0, \psi_0, \sigma_0)) = \mathsf{accept}$ and $(y_0, \psi_0, \sigma_0) \neq (y, \psi, \sigma)$ and $(y_0, \psi_0, \sigma_0) \neq (y, -\psi, \sigma)$ and $\sigma_0 \neq \sigma$.

It is straightforward that

$$\mathsf{Pr}[\mathcal{A} \text{ wins } \mathbf{Game\ 2}] = \mathsf{Pr}[\mathcal{A} \text{ wins } \mathbf{Game\ 3}] + \mathsf{Pr}[\mathcal{A} \text{ wins } \mathbf{Game\ 4}].$$

Claim C1 *If there is a non-negligible difference in the adversary's success probability between* **Game 1** *and* **Game 2***, then we can use the adversary to break the security of the PRF. That is,*

$$|\mathsf{Pr}[\mathcal{A} \textit{ wins } \mathbf{Game\ 1}] - \mathsf{Pr}[\mathcal{A} \textit{ wins } \mathbf{Game\ 2}]| \leq N_{PRF} \cdot \mathsf{Adv}^{PRF}_{\mathcal{A}},$$

where N_{PRF} is the number of distinct evaluations of PRF required to answer all store queries made by $\mathcal{A}$ and $\mathsf{Adv}^{\mathcal{A}}_{PRF}$ denotes the probability that $\mathcal{A}$ distinguish the output of PRF from true randomness.

The above Claim C1 can be proved using a standard hybrid argument. Here we save the details.

Claim C2 *If adversary $\mathcal{A}$ wins* **Game 3** *with non-negligible probability, then $\mathcal{A}$ can break the s-SDH assumption. That is,* $\mathsf{Pr}[\mathcal{A}$ *wins* **Game 3** $] \leq \mathsf{Pr}[\mathcal{A}$ *solves s-SDH problem*$] = \mathsf{Adv}^{s\text{-}SDH}_{\mathcal{A}}$.

Proof (of Claim C2). We construct an adversary $\mathcal{B}$, based on the adversary $\mathcal{A}$, to solve the s-SDH problem.

Given an input $(p, q, \{g^{\alpha^j}\}_{j=0}^{s-1})$, the adversary $\mathcal{B}$ can simulate the **Game 3** as below:

KeyGen : Choose τ, seed in the same way as in the algorithm KeyGen. Let $pk := (p, q, \{g^{\alpha^j}\}_{j=0}^{s-1})$ and $sk := (\mathsf{seed}, \alpha, \tau, \tau^{-1})$, where α is unknown to the adversary $\mathcal{B}$.

DEncode : For each file block $\vec{\boldsymbol{m}}_i$, generate the authentication tag t_i by randomly choosing an element from group $\mathbb{Z}_p$: $t_i \xleftarrow{\$} \mathbb{Z}_p$.

Note: There exists some unknown s_i's, such that $t_i = s_i + \tau f_{\vec{\boldsymbol{m}}_i}(\alpha) \mod p$ for each i.

Verification : In each verification, choose $r \xleftarrow{\$} \mathbb{Z}_p$ and $C = \{(i, \nu_i)\}$ at random.

The above game simulated by adversary $\mathcal{B}$ is identical to the **Game 3** to the view of $\mathcal{A}$. In **Game 3**, the adversary $\mathcal{A}$ is given the public key $pk = (p, q, \{g^{\alpha^j} \mod q\}_{j=0}^{s-1})$ and authentication tags t_i's for any data blocks (vectors) chosen by $\mathcal{A}$:

$$t_i = \mathsf{PRF}_{\mathsf{seed}}(\mathsf{id}, i) + \tau f_{\vec{\boldsymbol{m}}_i}(\alpha) = \mathsf{PRF}_{\mathsf{seed}}(\mathsf{id}, i) + \tau \sum_{j=0}^{s-1} m_{i,j}\alpha^j \mod p.$$

Since in **Game 3**, the value of $\mathsf{PRF}_{\mathsf{seed}}(\mathsf{id}, i)$ are *truly* uniformly random over $\mathbb{Z}_p$, the authentication tags t_i's reveal *no* information to adversary $\mathcal{A}$ about the secret α at all (even if $\mathcal{A}$ was computationally unbounded).

Let (y, ψ, σ) and (y_0, ψ_0, σ_0) be as specified in Lemma 4. Adversary $\mathcal{B}$ computes and outputs (c, w) to the SDH problem as below:

- If $y = y_0$: find any $c \neq -r \in \mathbb{Z}_p$ and set $w := g^{1/(r+c)} \mod q$;
- If $y \neq y_0$: set $c = -r$ and $w := \left(\frac{\psi}{\psi_0}\right)^{1/(y_0 - y)} \mod q$.

Now We want to show that: ***if $\mathcal{A}$ wins the* Game 3*, i.e.*** $\mathsf{Verify}(sk, \mathsf{id}, C, (y_0, \psi_0, \sigma_0)) = \mathsf{accept}$ ***and*** $(y_0, \psi_0, \sigma_0) \neq (y, \psi, \sigma)$ ***and*** $(y_0, \psi_0, \sigma_0) \neq (y, -\psi, \sigma)$ ***and*** $\sigma_0 = \sigma$***, then*** $g^{\frac{1}{\alpha + c}} = w$.

Since both (y, ψ, σ) and (y_0, ψ_0, σ_0) are accepted by the verifier w.r.t. the same challenge (r, C), the two tuples satisfy the Equation (6):

$$\psi^{\alpha - r} \stackrel{?}{=} g^{\tau^{-1}\left(\sigma - \sum_{(i,\nu_i)\in C} \nu_i \mathsf{PRF}_{\mathsf{seed}}(\mathsf{id}, i)\right) - y} \mod q \tag{14}$$

$$\psi_0{}^{\alpha - r} \stackrel{?}{=} g^{\tau^{-1}\left(\sigma_0 - \sum_{(i,\nu_i)\in C} \nu_i \mathsf{PRF}_{\mathsf{seed}}(\mathsf{id}, i)\right) - y_0} \mod q \tag{15}$$

Dividing Equation (14) with Equation (15), we obtain

$$\left(\frac{\psi}{\psi_0}\right)^{\alpha - r} = g^{\tau^{-1}(\sigma - \sigma_0) \; + \; y_0 - y} = g^{\; y_0 - y} \mod q \qquad (\text{ Since } \sigma_0 = \sigma) \tag{16}$$

If $y = y_0$, then $\psi \neq \psi_0$ and $-\psi \neq \psi_0$, since $(y_0, \psi_0, \sigma_0) \neq (y, \psi, \sigma)$ and $(y_0, \psi_0, \sigma_0) \neq (y, -\psi, \sigma)$ and $\sigma_0 = \sigma$ (Recall that we are assumming that $\mathcal{A}$ wins the **Game 3**). In this case, Equation (16) implies $\alpha = r$. Thus adversary $\mathcal{B}$'s output $(c, w = g^{1/(r+c)} = g^{1/(\alpha+c)})$ is a valid solution to the s-SDH problem.

If $y \neq y_0$, substituting $\frac{\psi}{\psi_0}$ with $w^{y_0 - y} \mod q$ into the above equation, we have

$$w^{(y_0 - y)(\alpha - r)} = g^{y_0 - y} \mod q \tag{17}$$

$$w = g^{1/(\alpha - r))} \mod q \tag{18}$$

Thus adversary $\mathcal{B}$'s output $(c = -r, w)$ is a valid solution to the s-SDH problem. We emphasize that the above argument allows the value c to be chosen at random. □

Claim C3 *For computationally unbounded adversary $\mathcal{A}$,* $\mathsf{Pr}[\mathcal{A}$ *wins* **Game 4**$] \leq \frac{1}{p}$.

Proof (of Claim C3). Let (y, ψ, σ) and (y_0, ψ_0, σ_0) be as specified in Lemma 4.

Suppose an computationally unbounded adversary $\mathcal{A}$ wins the Game 4, i.e. $\mathsf{Verify}(sk, \mathsf{id}, C, (y_0, \psi_0, \sigma_0)) = \mathsf{accept}$ and $(y_0, \psi_0, \sigma_0) \neq (y, \psi, \sigma)$ and $(y_0, \psi_0, \sigma_0) \neq (y, -\psi, \sigma)$ and $\sigma_0 \neq \sigma$.

Similar as the proof of Claim C2, we have

$$\left(\frac{\psi}{\psi_0}\right)^{\alpha - r} = g^{\tau^{-1}(\sigma - \sigma_0) \ + \ y_0 - y} \mod q \tag{19}$$

The computationally unbounded adversary $\mathcal{A}$ can find α from the public key by solving discrete log, and eventually find τ from the above equation. However, the secret τ is only involved in the authentication tag t_i's and is protected by the $\mathsf{PRF}_{\mathsf{seed}}(\cdot)$ which generates *truly* uniformly random numbers over $\mathbb{Z}_p$ (Note that in **Game 4**, the output of $\mathsf{PRF}_{\mathsf{seed}}(\cdot)$ is chosen at random and kept in a lookup table as in **Game 2**). Thus, the probability that $\mathcal{A}$ finds τ has to be $1/p$. Consequently, the adversary $\mathcal{A}$ wins the **Game 4** with probability

$$\mathsf{Pr}\left[\mathcal{A} \text{ wins } \textbf{Game 4}\right] \leq \mathsf{Pr}\left[\mathcal{A} \text{ finds } \tau\right] = \frac{1}{p}.$$

□

In summary, we have showed that

$$\begin{aligned} \mathsf{Adv}_{\mathcal{A}}^{\mathcal{E},\mathsf{forge},\mathsf{no\text{-}fb}} &= \mathsf{Pr}[\mathcal{A} \text{ wins } \textbf{Game 1}] \leq \mathsf{Pr}[\mathcal{A} \text{ wins } \textbf{Game 2}] + N_{PRF} \cdot \mathsf{Adv}_{\mathcal{A}}^{PRF} && (20)\\ &= (\mathsf{Pr}[\mathcal{A} \text{ wins } \textbf{Game 3}] + \mathsf{Pr}[\mathcal{A} \text{ wins } \textbf{Game 4}]) + N_{PRF} \cdot \mathsf{Adv}_{\mathcal{A}}^{PRF} && (21)\\ &\leq \mathsf{Adv}_{\mathcal{A}}^{s\text{-}SDH} + \frac{1}{p} + N_{PRF} \cdot \mathsf{Adv}_{\mathcal{A}}^{PRF}. && (22) \end{aligned}$$

Therefore, Lemma 4 is proved. □

Lemma 5 *If Assumption 3 (s-SDH Assumption) holds and PRF is secure, then* $\mathsf{Adv}_{\mathcal{A}}^{\mathcal{E},\mathsf{forge}}$ *is negligible, where* $\mathsf{Adv}_{\mathcal{A}}^{\mathcal{E},\mathsf{forge}}$ *is defined in the same way as* $\mathsf{Adv}_{\mathcal{A}}^{\mathcal{E},\mathsf{forge},\mathsf{no\text{-}fb}}$ *in* **Lemma 4** *except that the accept/reject decisions are provided to the adversary* $\mathcal{A}$ *at the end of each verification. Let* $\epsilon = \mathsf{Adv}_{\mathcal{A}}^{\mathcal{E},\mathsf{forge},\mathsf{no\text{-}fb}}$*. We have*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathcal{E},\mathsf{forge}} \leq \epsilon \times (1-\epsilon)^{N_{ver}} \ + \ \left(1 - (1-\epsilon)^{N_{ver}}\right) \approx \epsilon + N_{ver} \cdot \epsilon, \tag{23}$$

where N_{ver} *is the number of verification queries made by the adversary* $\mathcal{A}$*.*

Proof (of Lemma 5).

Game 1 The first game is the same as in **Game 1** in the proof of Lemma 4. It stands for forgeability game in no-feedback model.

Game 2.k For each integer $k \geq 0$, **Game 2.k** is the same as **Game 1**, except that the adversary $\mathcal{A}$ adaptively makes k verification queries before launching forgery attack and the accept/reject decisions are provided to the adversary $\mathcal{A}$ at the end of each of these k verifications.

We describe two different verification strategies for the simulator of the $\mathcal{POR}$ game.

- **Simulated verifier**: The simulator keeps a local copy of data and tags and plays an honest cloud storage server. For each proof (y_0, ψ_0, σ_0) received from the adversary $\mathcal{A}$, the simulator computes the corresponding correct proof (y, ψ, σ) from the simulator's local copy of data and tags. If $(y_0, \psi_0, \sigma_0) \in \{(y, \psi, \sigma), (y, -\psi, \sigma)\}$, then outputs accept; otherwise outputs reject.
- **Imaginary verifier**: An imaginary verification oracle $\mathcal{O}^{\mathsf{Verify}(sk;\cdot)}$ which somehow has the knowledge of the private key.

Note that (1) the simulated verifier accepts only correct proof while the imaginary verifier oracle accepts all valid proofs which include correct proofs; (2) the simulated verifier provides absolutely *no* new information to the adversary $\mathcal{A}$, since $\mathcal{A}$ itself can simulate such verifier by keeping another intact copy of the data of tags from the beginning.

Let us code accept with the bit '0' and code reject with the bit '1', and denote with $a_i \in \{0,1\}$ be the decision bit output by the imaginary verification oracle $\mathcal{O}^{\mathsf{Verify}(sk;\cdot)}$ for the i-th verification query made by the adversary $\mathcal{A}$; $b_i \in \{0,1\}$ be the decision bit output by the simulated verifier. Furthermore, let $A_k := a_1 a_2 \ldots a_k \in \{0,1\}^k$ and $B_k := b_1 b_2 \ldots b_k \in \{0,1\}^k$.

Claim C4 $\mathsf{Pr}[\mathcal{A}$ *wins* **Game 1**$] = \mathsf{Pr}[\mathcal{A}$ *wins* **Game 2.0** $]$

Claim C5 $\mathsf{Pr}[A_1 = B_1] = 1 - \epsilon$*, where* $\epsilon = \mathsf{Adv}_{\mathcal{A}}^{\mathcal{E},\mathsf{forge},\mathsf{no\text{-}fb}}$*.*

$\mathsf{Pr}[A_1 \neq B_1]$ is just the probability that the adversary $\mathcal{A}$ can forge a valid but not correct proof, without making any previous verification query.

Claim C6 *Let* $\epsilon = \mathsf{Adv}_{\mathcal{A}}^{\mathcal{E},\mathsf{forge,no\text{-}fb}}$. *For any integer* $k \geq 1$, *if* $\mathsf{Pr}[A_k = B_k] = (1-\epsilon)^k$, , *then* $\mathsf{Pr}[A_{k+1} = B_{k+1}] = (1-\epsilon)^{k+1}$.

This claim can be proved using the fact that $\mathsf{Pr}[a_{k+1} = b_{k+1} | A_k = B_k] = 1 - \epsilon$, which is because that $A_k = B_k$ indicates that the adversary $\mathcal{A}$ has obtained *no* information from the accept/reject feedbacks in the k verification queries.

Claim C7 $\mathsf{Pr}[\mathcal{A}$ *wins* **Game 2.k**$] \leq \epsilon \cdot (1-\epsilon)^k + \left(1 - (1-\epsilon)^k\right)$.

Claim C7 can be easily proved using the result $\mathsf{Pr}[A_k = B_k] = (1-\epsilon)^k$. Hence Lemma 5 is proved. □

C.3 Part Two Proof of Scheme EPOR

For any $C = \{(i, \nu_i)\}$, we choose s random nonces $r_0, r_1, \ldots, r_{s-1}$ and query the cloud storage sever with challenge (r_u, C), $0 \leq u \leq s-1$ in sequence. Denote the received response from the server as (y_u, ψ_u, σ_u). Suppose all responses are accepted. Then we can find the polynomial $f_{\vec{\boldsymbol{\mu}}_i}(x)$ which passes all points (r_u, y_u). The coefficients $\vec{\boldsymbol{\mu}}_i$ of polynomial $f_{\vec{\boldsymbol{\mu}}_i}(x)$ is a linear combination of data blocks $\vec{\boldsymbol{m}}_i$'s,

$$\vec{\boldsymbol{\mu}}_i = \sum_{(i,\nu_i) \in C} \nu_i \vec{\boldsymbol{m}}_i$$

which is identical to the output of a server in the private verification scheme in Shacham and Waters' [1]. Thus the client obtains the same information about the original data file after s successful verifications as she can obtain with Shacham and Waters' [1] scheme in one successful verification (*Note the communication cost per verification is* $\mathcal{O}(s)$ *in Shacham and Waters' [1], while* $\mathcal{O}(1)$ *in our case*). The rest of part two proof of Scheme EPOR follows **Theorem 4.3** in Shacham and Waters' [1].

C.4 Part Three Proof of Scheme EPOR

The part three proof of Scheme EPOR is identical to the one in Shacham and Waters [1] for the private verification scheme.

D An Efficient Variant version of Ateniese's PDP scheme

Ateniese [2]'s $\mathcal{PDP}$ scheme requires the data owner to compute a group exponentiation for each data block to generate an authentication tag in the setup. Now we present an efficient variant version of their scheme, which removes the demand of expensive exponentiation operations in the setup.

KeyGen Choose a RSA modulus $n = pq$ such that both p and q are safe primes. Choose at random $g \xleftarrow{\$} \mathbb{Z}_n^*$ such that g has a multiplicative order $(p-1)(q-1)/2$ Let $\phi(n) = (p-1)(q-1)$. Choose a random $\alpha \xleftarrow{\$} \mathbb{Z}_{\phi(n)}$. Choose a PRF seed s. Let $sk := (p, q, \alpha, s)$ and $pk = (n, g)$.

Tagging For each data block m_i, the data owner computes an authentication tag t_i:

$$t_i := \alpha m_i + \mathsf{PRF}(s, i) \mod \phi(n).$$

Challenging The verifier finds a random $d \xleftarrow{\$} \mathbb{Z}_n$, and computes $g_d := g^d \mod n$. Chooses a challenge $C := \{(i, \nu_i)\}$ at random, such that i's are index, and $\nu \in \mathbb{Z}_n$. Sends C and g_d to the server.

Proving The server finds all selected blocks m_i's and tags t_i's, and compute (π_1, π_2) as below

$$\pi_1 := \sum_{(i,\nu_i)\in C} \nu_i m_i; \tag{24}$$

$$\pi_2 := \sum_{(i,\nu_i)\in C} \nu_i t_i. \tag{25}$$

Note: The above two equations are computed over integer domain.

The server computes and sends $(g_d^{\pi_1} \mod n)$ and $(g^{\pi_2} \mod n)$ to the verifier.

Verifying The verifier checks whether the following equality holds:

$$(g_d^{\pi_1})^{\alpha} \stackrel{?}{=} \left(\frac{g^{\pi_2}}{g^{\sum_{(i,\nu_i)\in C} \nu_i \mathsf{PRF}(s,i)}}\right)^d \mod n$$

The security of the above scheme can be proved using the Knowledge of Exponent Assumption (KEA).

E EPOR with public verifiability

In order to construct a variant version of EPOR to support public key verification, we require an homomorphic signature scheme: Given valid message-signature pairs (M_i, σ_i) and weights ν_i's, one can efficiently find a valid signature for $\prod_i M_i^{\nu_i}$ with the public key only.

E.1 An Aggregatable Signature Scheme

SS.KeyGen(1^λ)

Choose a bilinear map $(p, \mathbb{G}, \mathbb{G}_T, g, e)$, such that p is a λ bits prime, both groups $\mathbb{G}$ and $\mathbb{G}_T$ have order p, g is a generator of group $\mathbb{G}$, and $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is a blinear map. Choose ζ, β, γ from $\mathbb{Z}_p^*$ at random. The public key is $spk := (g^{\frac{1}{\zeta}}, g, g^\beta, g^\gamma)$ and private key is $ssk := (\zeta, \beta, \gamma)$. Make the blinear map public.

SS.Sign(ssk, M)

Let $h_1, h_2 : \mathbb{G} \to \mathbb{G}$ be two random oracles. Choose y from $\mathbb{Z}_p$ at random. The signature for M is computed as below

$$(y, \sigma_0, \sigma_1) := \left(y,\ \left(h_1(M)M^\beta\right)^\zeta,\ \left(h_2(M)\,(Mg^y)^\gamma\right)^\zeta\right) \in \mathbb{Z} \times \mathbb{G} \times \mathbb{G}. \tag{26}$$

SS.Combine($spk, C, \{\nu_i : i \in C\}, \{(M_i; y_i, \sigma_{i,0}, \sigma_{i,1})\}_{i\in C}$)

Compute the aggregated signature for aggregated message $M := \prod_{i\in C} M_i^{\nu_i}$ as below

$$(y_M, \sigma_{M,0}, \sigma_{M,1}) := \left(\sum_{i\in C} \nu_i y_i \mod p,\ \prod_{i\in C} \sigma_{i,0}^{\nu_i},\ \prod_{i\in C} \sigma_{i,1}^{\nu_i}\right). \tag{27}$$

SS.Verify($spk, M, (y_M, \sigma_{M,0}, \sigma_{M,1}), C, \{\nu_i : i \in C\}$)

If the following two equalities hold, then output accept; otherwise output reject.

$$e(\sigma_{M,0}, g^{\frac{1}{\zeta}}) \stackrel{?}{=} e(g,\ \prod_{i\in C} h_1(M_i)^{\nu_i})\ e(g^\beta,\ M) \tag{28}$$

$$e(\sigma_{M,1}, g^{\frac{1}{\zeta}}) \stackrel{?}{=} e(g,\ \prod_{i\in C} h_2(M_i)^{\nu_i})\ e(g^\gamma,\ M \cdot g^{y_M}) \tag{29}$$

Lemma 6 *The signature scheme* $\mathsf{SS} = (\mathsf{SS.KeyGen}, \mathsf{SS.Sign}, \mathsf{SS.Combine}, \mathsf{SS.Verify})$ *constructed as above is known-plaintext unforgeable in the random oracle model, if Diffie-Hellman Inversion Assumption holds.*

Proof. Given $(w, w^{\zeta}, w^{\zeta^2})$, the goal is to find w^{ζ^3}.

The simulator simulates the signature scheme as below:

- KeyGen: Choose s_0, s_1 from $\mathbb{Z}_p^*$ at random. Set $\beta = s_0\zeta$ and $\gamma = s_1\zeta$. The public key is $pk = (g^{\frac{1}{\zeta}} = w, g = w^{\zeta}, g^{\beta} = \left(w^{\zeta^2}\right)^{s_0}, g^{\gamma} = \left(w^{\zeta^2}\right)^{s_1}$ and the private key is $sk = (\zeta, \beta, \gamma)$.
  Note: The simulator knows the values of public key and does not know the values of ζ, β *or* γ.
- Sign: The simulator chooses m_i from $\mathbb{Z}_p$ and sets the message $M_i := g^{m_i}$. The simulator programs the random oracle h_1 and h_2 as follows:
  - $h_1(M_i)$: Choose z_i at random from $\mathbb{Z}_p$ and set $h_1(M_i)$ as
  $$h_1(M_i) := \frac{g^{z_i}}{M_i^{\beta}} = \frac{g^{z_i}}{(g^{\beta})^{m_i}}.$$
  - $h_2(M_i)$: Choose u_i at random from $\mathbb{Z}_p$ and set $h_2(M_i)$ as
  $$y_i \xleftarrow{\$} \mathbb{Z}_p; \quad h_2(M_i) := \frac{g^{u_i}}{M_i^{\gamma} g^{y_i\gamma}} = \frac{g^{u_i}}{(g^{\gamma})^{m_i+y_i}}.$$

  The signature is $(y_i, \sigma_{i,0}, \sigma_{i,1})$, where $\sigma_{i,0}, \sigma_{i,1}$ are computed as below
  $$\sigma_{i,0} := \left(h_1(M_i) M_i^{\beta}\right)^{\zeta} = (g^{z_i})^{\zeta} = \left(g^{\zeta}\right)^{z_i} \tag{30}$$
  $$\sigma_{i,1} := \left(h_2(M_i)\,(M_i g^{y_i})^{\gamma}\right)^{\zeta} = (g^{u_i})^{\zeta} = \left(g^{\zeta}\right)^{u_i}. \tag{31}$$

Let $(y_M, \sigma_{M,0}, \sigma_{M,1})$ denote the correct signature for an aggregated message $M = \prod_{i\in C} M_i$ w.r.t. $\{\nu_i : i \in C\}$. Suppose an adversary can find a different valid signature $(\hat{y}_M, \hat{\sigma}_{M,0}, \hat{\sigma}_{M,1})$ for $\hat{M}$.

We have
$$e(\sigma_{M,0}, g^{\frac{1}{\zeta}}) = e(g, \prod_{i\in C} h_1(M_i)^{\nu_i})\; e(g^{\beta},\; M) \tag{32}$$
$$e(\sigma_{M,1}, g^{\frac{1}{\zeta}}) = e(g, \prod_{i\in C} h_2(M_i)^{\nu_i})\; e(g^{\gamma},\; M \cdot g^{y_M}) \tag{33}$$
$$e(\hat{\sigma}_{M,0}, g^{\frac{1}{\zeta}}) = e(g, \prod_{i\in C} h_1(M_i)^{\nu_i})\; e(g^{\beta},\; \hat{M}) \tag{34}$$
$$e(\hat{\sigma}_{M,1}, g^{\frac{1}{\zeta}}) = e(g, \prod_{i\in C} h_2(M_i)^{\nu_i})\; e(g^{\gamma},\; \hat{M} \cdot g^{\hat{y}_M}) \tag{35}$$

$$\frac{\text{Eq 34}}{\text{Eq 32}}: \quad e(\frac{\hat{\sigma}_{M,0}}{\sigma_{M,0}}, g^{\frac{1}{\zeta}}) = e(g^{\beta},\; \frac{\hat{M}}{M}) \tag{36}$$
$$\frac{\text{Eq 35}}{\text{Eq 33}}: \quad e(\frac{\hat{\sigma}_{M,1}}{\sigma_{M,1}}, g^{\frac{1}{\zeta}}) = e(g^{\gamma},\; \frac{\hat{M}}{M} \cdot g^{\hat{y}_M - y_M}) \tag{37}$$

Since $\gamma = s\beta = s_0^{-1}s_1\beta$, we have
$$e(\left(\frac{\hat{\sigma}_{M,1}}{\sigma_{M,1}}\right)^{s^{-1}}, g^{\frac{1}{\zeta}}) = e(g^{\beta},\; \frac{\hat{M}}{M} \cdot g^{y_M}) \tag{38}$$
$$e(\left(\frac{\hat{\sigma}_{M,1}}{\sigma_{M,1}}\right)^{s^{-1}} \frac{\sigma_{M,0}}{\hat{\sigma}_{M,0}},\; g^{\frac{1}{\zeta}}) = e(g^{\beta}, g^{\hat{y}_M - y_M}) \tag{39}$$

Therefore, $g^{\zeta\beta}$ can be computed
$$\left(\left(\frac{\hat{\sigma}_{M,1}}{\sigma_{M,1}}\right)^{s^{-1}} \frac{\sigma_{M,0}}{\hat{\sigma}_{M,0}}\right)^{\frac{1}{\hat{y}_M - y_M}} = g^{\zeta\beta}.$$

Problem: How about $\hat{y}_M - y_M = 0$*?*

As a result, the simulator can compute $w^{\zeta^3} = \left(g^{\beta}\right)^{s_0^{-1}\zeta} = \left(g^{\zeta\beta}\right)^{s_0^{-1}}$.

E.2 Construction of EPOR with public verifiability

$\mathsf{KeyGen}(1^\lambda) \rightarrow (pk, sk)$

Run $\mathsf{SS.KeyGen}(1^\lambda)$ to generate signing key pair (spk, ssk). Let the $(p, \mathbb{G}, \mathbb{G}_T, g, e)$ be the bilinear map groups chosen by $\mathsf{SS.KeyGen}$, where p is a λ bits prime, both groups $\mathbb{G}$ and $\mathbb{G}_T$ have order p, g is a generator of group $\mathbb{G}$, and $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ is a blinear map. Choose at random two elements τ, α from $\mathbb{Z}_p^*$: $\tau, \alpha \xleftarrow{\$} \mathbb{Z}_p^*$, Choose at random a PRF key, denoted as seed. The public key is $pk := (spk, p, g^\tau, \{g_j := g^{\alpha^j} \in \mathbb{G}\}_{j=0}^{s-1})$ and the private key is $sk := (ssk, p, \mathsf{seed}, \alpha, \tau)$.

$\mathsf{DEncode}(sk, \mathrm{M}) \rightarrow (\mathsf{id}, \hat{\mathrm{M}})$

Given the file $\mathbf{M}$, first apply the error erasure code on it; then split the error erasure encoded file into n vectors (or blocks), such that each vector consists of s elements from $\mathbb{Z}_p$: $\{\vec{\boldsymbol{m}}_i = (m_{i,0}, \ldots, m_{i,s-1}) \in \mathbb{Z}_p^s\}_{0 \le i \le n-1}$. Choose a unique identifier id from domain $\{0,1\}^\lambda$. For each $\vec{\boldsymbol{m}}_i$, $0 \le i \le n-1$, compute an authentication tag t_i as below

$$t_i := \mathsf{SS.Sign}(ssk, g^{f_{\vec{\boldsymbol{m}}_i}(\alpha)})$$

The final encoded file $\hat{\mathbf{M}}$ consists of $\{\vec{\boldsymbol{m}}_i = (m_{i,0}, \ldots, m_{i,s-1})\}$, $0 \le i \le n-1$, together with authentication tags $\{t_i\}$, $0 \le i \le n-1$.

$\mathsf{Prove}(pk, \mathsf{id}, \hat{\mathrm{M}}, (r, C)) \rightarrow (y, \psi, \sigma)$

The challenge is (r, C), where $r \xleftarrow{\$} \mathbb{Z}_p^*$ is a random nonce chosen from $\mathbb{Z}_p^*$, and C is a set $\{(i, \nu_i)\}$ of ℓ index-weight pairs (i, ν_i)'s, where each index $i \in [0, n-1]$, each weight $\nu_i \in B \subseteq [1, 2^\lambda]$, and i's are distinct (*Note: The set B is a system parameter as in [1]* and will be used in the proof. See Appendix B.1). Compute

$$\mu_j := \sum_{(i,\nu_i) \in C} \nu_i m_{i,j} \mod p \qquad \text{for } 0 \le j \le s-1, \tag{40}$$

$$\sigma := \prod_{(i,\nu_i) \in C} t_i^{\nu_i} \in \mathbb{G}. \tag{41}$$

Let vector $\vec{\boldsymbol{\mu}} := (\mu_0, \ldots, \mu_{s-1})$ (*Note:* $\vec{\boldsymbol{\mu}} = \sum_{(i,\nu_i) \in C} \nu_i \vec{\boldsymbol{m}}_i$). Evaluate polynomial $f_{\vec{\boldsymbol{\mu}}}(x)$ at point $x = r$ to obtain $y := f_{\vec{\boldsymbol{\mu}}}(r) \mod p$. Divide the polynomial $f_{\vec{\boldsymbol{\mu}}}(x) - f_{\vec{\boldsymbol{\mu}}}(r)$ with $(x - r)$ using polynomial division, and denote the coefficients vector of the resulting quotient polynomial as $\vec{\boldsymbol{w}} = (w_0, \ldots, w_{s-2})$, that is, $f_{\vec{\boldsymbol{w}}}(x) \equiv \frac{f_{\vec{\boldsymbol{\mu}}}(x) - f_{\vec{\boldsymbol{\mu}}}(r)}{x - r}$. Compute ψ and ψ_2 with the public key $pk = (p, g^\tau, \{g^{\alpha^j} \mod q\}_{j=0}^{s-1})$ as below

$$\psi := \prod_{j=0}^{s-2} \left(g^{\alpha^j}\right)^{w_j} = g^{f_{\vec{\boldsymbol{w}}}(\alpha)} \in \mathbb{G}. \tag{42}$$

$$\psi_2 := \prod_{j=0}^{s-1} \left(g^{\alpha^j}\right)^{\mu_j} = g^{f_{\vec{\boldsymbol{\mu}}}(\alpha)} \in \mathbb{G}. \tag{43}$$

$$\varsigma := \mathsf{SS.Combine}(spk, C, \{g^{f_{\vec{\boldsymbol{m}}_i}(\alpha)}, t_i\}_{(i,\nu_i) \in C}) \tag{44}$$

Output $(y, \psi, \sigma, \psi_2, \varsigma)$.

$\mathsf{Verify}(sk, \mathsf{id}, (r, C), (y, \psi, \sigma, \psi_2)) \rightarrow$ accept or reject

Parse C as a set $\{(i, \nu_i)\}$ of ℓ index-weight pairs (i, ν_i)'s, where each index $i \in [0, n-1]$, each weight $\nu_i \in B \subseteq [1, 2^\lambda]$, and i's are distinct. Verify the following two equalities with the private key $sk = (p, \mathsf{seed}, \alpha, \tau)$. If it holds, then output accept; otherwise, output reject.

$$\mathsf{SS.Verify}(spk, \psi_2, \varsigma, C) \stackrel{?}{=} \mathsf{accept} \tag{45}$$

$$e(\psi,\ g^\alpha / g^r) e(g,\ g)^y \stackrel{?}{=} e(\psi_2, g) \tag{46}$$

Note that: (1) In [1], if we aggregate $\mu_1, \ldots, \mu_s$ as $\prod_{j=1}^s u_j^{\mu_j}$ in the scheme with public verification, the resulting scheme is not a secure $\mathcal{POR}$. (2) The above scheme implies an efficient delegation scheme for polynomial evaluation, where polynomial commitments $g^{f_{\vec{\boldsymbol{m}}}(\alpha)}$'s can be put in the untrusted server and their integrity can be protected with our new aggregatable signature scheme constructed in Appendix E.1.