Practically Efficient Proof of Retrievability in Cloud Storage

Jia XU
jiaxu2001@gmail.com

National University of Singapore
Department of Computer Science

Abstract. Cloud storage, among other cloud computing services, is becomming more and more prevalent in the IT industry. In a secure cloud storage application, a user Alice outsources (backups) her data file together with some authentication data to a potentially untrusted Cloud Storage Server Bob. Later, Alice wants to periodically and remotely verify the integrity of her data stored with Bob using the authentication data, without keeping a local copy of the data file or retrieving back the data file during the verification. We propose two secure and efficient methods that allow Bob to prove to Alice that he indeed keeps her data file intactly:

- Our first method is the first provable-secure Proof of Retrievability scheme constructed over integer domain, and has the same complexity as Shacham and Waters [1].
- Our second method reduces the communication complexity of Shacham and Waters [1] from $\mathcal{O}(s)$ to $\mathcal{O}(1)$, where keeping other aspects of complexity unchanged, using our new construction of a functional encryption scheme.

Keywords: Cloud Storage, Proof of Retrievability, Remote Data Integrity Check, Homomorphic Authentication Tag, Functional Encryption

1 Introduction

In a cloud storage service, many users (individuals or companies) store (or backup) their data files on a remote cloud storage server through Internet. These data may or may not be retrieved, depending on the usrs' future demand, which is hard to predict in advance in many cases. There is a chance that some portion of the data is corrupted in the cloud storage server, but the owner of the data is not aware of it and will be suffered from various loss (e.g. economic loss) due to the corrupted data. The threat to data itegrity in cloud storage is indeed realistic and several events about massive loss of Gmail and Hotmail have been reported. It is desired to allow the data owner to check the integrity of their data stored in cloud storage server remotely, in an efficient and reliable manner, without simply trusting in the cloud storage server. The separation between the guarantee of data integrity and trust in the potential dishonest cloud storage server, may conciliate users who are afraid to adopt cloud computing service due to security concern.

A data owner, say Alice, has many reasons to not trust the messages from the cloud storage server Bob, which includes but not limited to: (1) Bob has incentive to save his storage and computation cost by exploiting the weakness of the authentication mechanism; (2) Bob may collude with the business competitor of the Alice and intend to fool Alice with altered data which may provide the competitor advantage over Alice; (3) Bob may be compromised due to internal attacker (e.g. angry unsatisfied employee) or outside attacker (e.g. third party attacker tempering the communication channel); (4) Occasional harddisk failure and CPU/RAM malfunction and so on.

The above concerns and efficiency requirment rule out the following straightforward approaches: (1) The data owner Alice computes and keeps a hash (e.g. SHA256) value of her data file before outsourcing the data to cloud storage server. Later the data owner can ask the cloud storage server to compute and send back the hash value. This method is not robust for multilple verifications, since the potential untrusted cloud storage server could keep the hash value and pass all subsequent verifications without keeping the user's data any more, so that cloud storage server can save storage cost and computation cost. (2) The data owner keeps a local copy of her data after outsourcing. Then in each verification, the data owner can choose a random key k and asks the cloud storage server for the MAC value of her data w.r.t. the key k. Data owner Alice can verify the returned MAC value with her local copy of data. This method requires linear storage w.r.t. the outsourced data on Alice's side, and violate or weaken the benefits of cloud storage service. (3) In each verification, Alice can retrieve the data back, compute a hash value localy, and compare it with the pre-computed hash value that is kept in Alice's local storage. This method requires linear communication bandwidth w.r.t. the data size per verification.

Our Controbution Our contributions can be summarized as below.

- We propose the first secure $\mathcal{POR}$ scheme over integer domain:
  - We design a new short homomorphic verification tag over integer domain, where the bit-length of a tag is smaller than the input. We prove that the proposed tag function is a secure MAC (Message Authentication Code).
  - We design a new $\mathcal{POR}$ scheme based on the short homomorphic verification tag. The proposed $\mathcal{POR}$ scheme is efficient. Precisely, our scheme requires 2 IP packets per vericiation, constant storage overhead on verifier side and 2% storage overhead on prover side, and the computation on both sides requires only addition, multiplication and stream cipher operation.
  - We prove that the proposed $\mathcal{POR}$ scheme is secure under Riemann hypothesis in standard model.
- We improve Shacham and Waters [1]'s scheme (the one with private verification) by reducing the communication complexity from $\mathcal{O}(s)$ to $\mathcal{O}(1)$:
  - We propose a new functional encryption scheme. In our functional encryption scheme w.r.t. the one-way keyed function family $\{f_\rho\}$, a decryption key k_ρ w.r.t. function key ρ can be generated from a secret key. With the decryption key k_ρ and a ciphertext of a message Msg as input, the decryption algorithm will output a function value $f_\rho(\mathsf{Msg}) = \Omega^{\rho \mathsf{Msg}}$ for some constant Ω.
  - We design a new $\mathcal{POR}$ scheme by incorporating the above functional encryption scheme into Shacham and Waters [1]'s scheme. The new method improves the communication from $\mathcal{O}(s)$ to $\mathcal{O}(1)$.
  - We prove that the new method is secure under a slightly weak model of $\mathcal{POR}$: If all accept or reject decisons are *completely* hidden from the cloud storage server, the new scheme is a secure $\mathcal{POR}$.

To some extent, our scheme combines advantages of both Shacham and Waters [1] and Ateniese *et al.* [2]. Shacham and Waters [1] is provably secure under the strongest model of Juels [3], and is efficient in computation where only addition, multiplication and stream cipher operation are involved. However, Shacham and Waters [1] requires linear (w.r.t. data file size) storage overhead on Bob's side. Ateniese *et al.* [2] requires small storage overhead (precisely a small fraction of the data file size) in the bright aspect, but expensive exponention [1] with large exponent in computation. Its security is proved under a weaker $\mathcal{PDP}$ model. Our scheme requires only addition, multiplication

[1] Note that Ateniese *et al.* [2] gave an efficient variant scheme E-PDP that avoids expensive exponention operations. However, this variant scheme E-PDP is broken by Shacham and Waters [1]. The detailed discussion is given in Section 1.1.

and stream cipher operation are involved, and small storage overhead, and it is proably secure under $\mathcal{POR}$ model. Furthermore, all of the three schemes require two IP packets for communication per verification.

1.1 Related work

Recently, a lot of works [3,2,4,1,5,6,7,8,9,10,11] have devoted to the study of remote data integrity check. Juels *et al.* [3] presented a strong security model, Ateniese *et al.* [2] gave an efficient scheme which is secure under a weaker $\mathcal{PDP}$ model. [4,1,9] proposed efficient methods using some sorts of homomorphic authentication tag. [6] extends to dynamic setting and [10] exploits public verifiability and [11] studied the privacy issue in public verification.

It is worthy to point out that, the most efficient variant scheme E-PDP by 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 $\prod_{(i,a_i)\in Chal} \mathtt{T}_i^{a_i}$ 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 exponention 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.

2 Formulation

2.1 System Model

We restate the $\mathcal{POR}$ [3,1] model as below, with slight modifications on notations. We adopt the 1-round verify-prove version in Juels [3] for simplicity.

Definition 1 ($\mathcal{POR}$ [3,1]) *A* Proof Of Retrievability *($\mathcal{POR}$) scheme consists of four algorithms* (KeyGen*,* DEnc*,* Prove*,* Verify)*:*

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

Completeness. A $\mathcal{POR}$ scheme (KeyGen, DEnc, Prove, Verify) is *complete*, if an honest prover (who ensures the integrity of his storage and follow 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 $\mathbf{X}$, any challenge query C, if $\psi \leftarrow \mathsf{Prove}(pk, \mathsf{id}_\mathbf{X}, \hat{\mathbf{X}}, C)$, then $\mathsf{Verify}(sk, \mathsf{id}_\mathbf{X}, C, \psi)$ outputs accept with probability 1, where $(\mathsf{id}_\mathbf{X}, \hat{\mathbf{X}}) \leftarrow \mathsf{DEnc}(sk, \mathbf{X})$.

2.2 Security Model

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

For simplicity, the following topics are out of the scope of this paper (We emphasize that this is not a limitation of our model):

1. Eavesdrop and tempering: the communication channel between the data owner (verifier) and cloud storage server (prover) is assumed to be secure, since (1) any adversarial behavior over the communication channel (e.g. monitoring or tempering the channel) can be performed on the server side with the same or better effect; (2) we can secure the communication channel using SSL/TLS; (3) we can prevent the server from being framed by third party adversaries over the communication channel, by requiring the server to sign his messages.
2. Denial of Service Attack: DOS attack is also out of the scope of this paper. That is because DOS attack launched by any third party attacker can be dealed with using the exiting DOS countermeasure in the literature, and DOS attack launched by the cloud storage server cannot be resolved in technical manner.
3. Frame attack: We can prevent Bob from framed by Alice by requiring each party to sign every single message that he/she sends out.

2.2.2 $\mathcal{POR}$ Security Game We rewrite the $\mathcal{POR}$ security game in Juels *et al* [3] and Shacham *et al.* [1] in a standard way. The security game between an PPT adersary $\mathcal{A}$ and a PPT challenger $\mathcal{C}$ for a $\mathcal{POR}$ scheme $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{DEnc}, \mathsf{Prove}, \mathsf{Verify})$ 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 $\mathcal{A}$ adapatively make queries where each query is one of the following:

– Store query $(\mathbf{X})$: Given a data file $\mathbf{X}$ chosen by $\mathcal{A}$, the challenger $\mathcal{C}$ responses by running data encoding algorithm $(\mathsf{id}, \hat{\mathbf{X}}) \leftarrow \mathsf{DEnc}(sk, \mathbf{X})$ and sending the encoded data file $\hat{\mathbf{X}}$ 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 $\mathbf{X}$ 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 $\mathsf{Verify}(sk, \mathsf{id}, Chal, \psi)$. Denote the output as b.

  At the end $\mathcal{C}$ gives 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 $\mathbf{X}^*$ denote the data file associated to identifier id^*.

Retreive: The challenger $\mathcal{C}$ initiates ζ number of $\mathcal{POR}$ verifications with $\mathcal{A}$ w.r.t. the data file $\mathbf{X}^*$ specified by the identifier id, 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 $\mathbf{X}'$ using some PPT extractor algorithm. The adversary $\mathcal{A}$ wins this game, if and only if $\mathbf{X}' \neq \mathbf{X}^*$.

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

Definition 2 *We define the advantage of an adversary $\mathcal{A}$ against a $\mathcal{POR}$ scheme $\mathcal{E}$ as the probability that $\mathcal{A}$ wins the security game for $\mathcal{E}$, that is,*

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

Definition 3 ([3,1]) *A $\mathcal{POR}$ scheme is $(\epsilon, \zeta, \delta)$-sound, if for any PPT ϵ-admissible adversary $\mathcal{A}$, the advantage $\mathsf{Adv}_{\mathcal{A}}^{\mathcal{E}}(\zeta) \leq \delta$.*

2.2.3 $\mathcal{POR}$ Security in no-feedback setting

Furthermore, we define an alternative security game called $\mathsf{Game}_{\mathcal{A}}^{\mathcal{E},\mathsf{no\text{-}fb}}$, which is identical to $\mathsf{Game}_{\mathcal{A}}^{\mathcal{E}}$, except that in all verifications in the game, all of the verifier's (i.e. that challenger's) decision bits (accept or reject) are *completely* hidden from the prover (i.e. the adversary). Consequently, the adversary's advantage in the new no-feedback security game $\mathsf{Game}_{\mathcal{A}}^{\mathcal{E},\mathsf{no\text{-}fb}}$ is denoted as $\mathsf{Adv}_{\mathcal{A}}^{\mathcal{E},\mathsf{no\text{-}fb}}$. Without the feedback bits, the adaptive adversary's power will be weaken, in the sense that adaptive verification queries may not bring adversary more information, since the adversary may be able to generate challenge by himself with public information, although he is still benefited from adaptive store queries.

This $\mathcal{POR}$ security definition in no-feedback setting is similar to the formulation for veriable cloud computing [12,13], which allows client to outsource (or delegate) any polynomial time function to an untrusted cloud computing server by using fully homomorphic encryption [14]. The schemes proposed in [12,13] are suffering from attacks and not secure, if the decision bits for previous interactions are provided as feedback to the adversary (i.e. the potentially untrusted cloud computing server). However, in case that the adversary gets feedbacks, whether our second scheme is secure or not, remains an open problem.

2.2.4 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 $\mathsf{Game}_{\mathcal{A}}^{\mathcal{E}}$ and $\mathsf{Game}_{\mathcal{A}}^{\mathcal{E},\mathsf{no-fb}}$, especially the $\mathsf{Retreive}$ phase, are only for security formulation, and application of a $\mathcal{POR}$ scheme does not necesarily follow the description of these games. For example, in real world application, 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 $\mathsf{Retreive}$ phase in the security games just ensures that user's file *can* be recoved from multiple verifications with the cloud storage server efficiently (using some PPT extractor algorithm), as long as the cloud storage server can pass a nonnegligible fraction of challenge queries. Essentially, a secure $\mathcal{POR}$ scheme provides a mechanism, in which the data owner will be guaranteed that here data file can be efficiently recoved from the server's storage at the moment of an accepted verification, 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[2], without trusting in the cloud storage server.

[2] For information-theoretical secure $\mathcal{POR}$ schemes (e.g. [7]), there will be no such assumption.

3 Scheme $\mathsf{EPOR\text{-}I}$: Based on Short Short Homomorphic Verification Tag

3.1 Short Homomorphic Verification Tag

Let p be a large prime, $\alpha \in \mathbb{Z}_p^*$ and $\mathsf{seed} \in \{0,1\}^\lambda$. Let $F_{\mathsf{seed}} : \{0,1\}^* \to \mathbb{Z}_p \times \{0,1\}$ be a pseudorandom function. Let $F^L_{\mathsf{seed}} : \{0,1\}^* \to \mathbb{Z}_p$ and $F^R_{\mathsf{seed}} : \{0,1\}^* \to \{0,1\}$ be functions such that $F_{\mathsf{seed}}(x) = (F^L_{\mathsf{seed}}(x), F^R_{\mathsf{seed}}(x))$ for all x in the domain. Let $Q \in [p, 2p]$ be an integer. Let key $\mathcal{K} = (Q, p, \alpha, s)$. We define tag function

$$\text{For } x \in [0, 2^{\xi\lambda} - 1], \quad \mathsf{Tag}_{\mathcal{K}}(x, i) \overset{\text{def}}{=} \Big(\alpha x + F^L_{\mathsf{seed}}(i) \mod p\Big) \; + \; F^R_{\mathsf{seed}}(i) \times (Q - p) \quad \in [0,\ Q - 1] \tag{2}$$

It is worthy to point out that, the bit length of a tag (or p) is about 2λ and the bit length of x (or N) is about $\xi\lambda$.

Lemma 1 *Assume $F_{\mathsf{seed}}(\cdot)$ is a random oracle. Let $Q = p(1+\epsilon)$ (alternatively, $Q = p(2-\epsilon)$) for some negligible ϵ. Then the distribution of output of Tag is statistical close to a uniform distribution over integer interval $[0, Q-1]$.*

Note that all of p, Q and ϵ are considered as functions of the security parameter λ.

Lemma 2 *Assuming that the Riemann hypothesis holds, there exist more than $\frac{1}{4\pi}\sqrt{Q}\ln Q$ primes in the interval $(Q_0, Q]$ where $Q_0 = Q - \frac{1}{2\pi}\sqrt{Q}\ln^2 Q$ and $Q_0 \geq 2657$.*

Theorem 3 *Let $\mathcal{K} = (p, Q, \alpha, s)$ be the key of Tag, such that $Q = 2^{2(\lambda+\varsigma)}$ and p is a prime chosen at random from a subset P of interval $(Q_0, Q]$ with $Q_0 = Q - \frac{1}{2\pi}\sqrt{Q}\ln^2 Q$. Assume $F_{\mathsf{seed}}(\cdot)$ is random oracle. Given value of Q and a set $S = \{(x, \mathsf{Tag}_{\mathcal{K}}(x)) \in [0, 2^{\xi\lambda} - 1] \times [0, Q-1]\}$ of data-tag pairs, any (computationally unbounded) adversary cannot find p with probability larger than $|S_U|(Q/p - 1) + 1/(\#Prime)$, where $\#Prime$ denotes the number of primes in the subset P of interval $(Q_0, Q]$.*

3.2 Alternative Construction

$$\text{For } x \in [0, 2^{\xi\lambda} - 1], \quad \mathsf{Tag}_{\mathcal{K}}(x, i) \overset{\text{def}}{=} \alpha x + F_{\mathsf{seed}}(i) \mod p \tag{3}$$

3.3 Efficient $\mathcal{POR}$ scheme based on Short HVT

We present our scheme as below and call it $\mathsf{EPOR\text{-}I}$.

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

Generate the tag key $\mathcal{K} = (Q, p, \alpha, \mathsf{seed})$, such that $Q = 2^{2(\lambda+\varsigma)}$, p is a prime chosen at random from $(Q_0, Q]$ with $Q_0 = Q - \frac{1}{2\pi}\sqrt{Q}\ln^2 Q$, α is chosen from $\mathbb{Z}_p^*$ and seed is chosen at random from $\{0,1\}^\lambda$. Let $F_{\mathsf{seed}} : \{0,1\}^* \to \mathbb{Z}_p \times \{0,1\}$, $\mathsf{seed} \in \{0,1\}^\lambda$, be a pseudorandom function. Let $F^L_{\mathsf{seed}} : \{0,1\}^* \to \mathbb{Z}_p$ and $F^R_{\mathsf{seed}} : \{0,1\}^* \to \{0,1\}$ be functions such that $F_{\mathsf{seed}}(x) = (F^L_{\mathsf{seed}}(x), F^R_{\mathsf{seed}}(x))$ for all x in the domain. Make $pk := Q$ public public, and keep $sk := (p, \alpha, \mathsf{seed})$ secret.

DEnc$(sk, X) \to (\mathsf{id}, \{(x_i, t_i)\}_{i\in[n]})$

Given a file $\mathbf{X}$, first apply the error correcting code to obtain $\mathbf{X}'$; then split $\mathbf{X}'$ into n blocks and each block has exactly[3] $\xi\lambda$ bits: $\mathbf{X}' := (x_1, x_2, \ldots, x_n)$ where each $x_i \in [0, 2^{\xi\lambda} - 1]$, $1 \le i \le n$. Choose a unique identifier[4] id in the domain $\{0,1\}^{2\lambda}$. Generate an authentication tag t_i for each block x_i as below

$$t_i := \mathsf{Tag}_{\mathcal{K}}(x_i, i) = \left(\alpha x_i + F^L_{\mathsf{seed}}(\mathsf{id}, i) \mod p\right) + F^R_{\mathsf{seed}}(\mathsf{id}, i) \times (Q - p) \quad \in [0,\ Q-1]. \tag{4}$$

Send the identifier id, data blocks $(x_1, x_2, \ldots, x_n)$ together with their tags $(t_1, t_2, \ldots, t_n)$ to the server.

Prove$(pk, \mathsf{id}, \{(x_i, t_i)\}, C) \to (\psi_1, \psi_2)$

The prover receives challenge query $C = \{(i, \beta_i) : i \in [1, n], \beta_i \in [1, 2^\lambda]\}$ from the verifier. He computes (ψ_1, ψ_2) as below and sends them to the verifier as response:

$$\psi_1 := \sum_{(i,\beta_i)\in C} \beta_i x_i; \tag{5}$$

$$\psi_2 := \sum_{(i,\beta_i)\in C} \beta_i t_i. \tag{6}$$

Note: Equation (5) *and Equation* (6) *are computed over integer domain.*

Verify$(sk, \mathsf{id}, C, \psi_1, \psi_2) \to$ accept or reject

The verifier recieves from the prover a response (ψ_1, ψ_2) for the challenge query $C = \{(i, \beta_i) : i \in [n], \beta_i \in [1, 2^\lambda]\}$, and outputs accept if the following equation hold:

$$\psi_2 - (Q - p) \sum_{(i,\beta_i)\in C} F^R_{\mathsf{seed}}(\mathsf{id}, i)\beta_i \overset{?}{=} \alpha\psi_1 + \sum_{(i,\beta_i)\in C} \beta_i F^L_{\mathsf{seed}}(\mathsf{id}, i) \mod p \tag{7}$$

Otherwise, the verifier outputs reject.

Theorem 4 *The proposed scheme* EPOR-Int *is a secure* $\mathcal{POR}$ *scheme.*

The proof is similar to Shacham and Waters [1]: the Part II and Part III proof are identical; the difference between Part I protocol is handled by Theorem 3.

Remark

– The challenge $C = \{(i, \beta_i) : i \in [n], \beta_i \in [1, 2^\lambda]\}$ can be represented compactly by two λ bits random seeds using a pseudorandom function.

3.4 Alternative Construction

We can change the tag function to the alternative construction in (3) and obtain an alternative $\mathcal{POR}$ scheme.

Table 1: Asymptotic Performance: All schemes have $O(\lambda)$ bits private key. A challenge contains w index-coefficient pairs.

Scheme	Communication (Bits)	Communication (IP Packets)	Storage Overhead	Computation (Prover)	Computation (Verifier)
EPOR-I	$(\xi+5)\lambda + 2\log w$	$((\xi+5)\lambda + 2\log w)/MTU$	$FileSize/\xi$	$2w$ mul+ $2w$ add + w PRF	w mul+ w add + w PRF

3.5 Performance Analysis

3.6 Parameter Selection

For 80 bits security, we set $\lambda = 80$ and $\varsigma = 40$. The bit length of a tag value will be $2(\lambda+\varsigma) = 240$. Assume the Maxmum Transimision Unit (MTU) is 1500 Bytes. We set $N = 2^{1460\times 8}$. The bit length of a data block will be $1460 \times 8 - 100 = 11580$. The ratio of tag to data is $240/11580 \approx 2\%$. If we allow 4 IP packets per verification, we can squeeze the storage overhead further to 1%.

4 Scheme EPOR-II

In this section, we construct a new functional encryption scheme, and apply it into Water and Brents [1] (private) scheme in order to reduce communication.

4.1 Functional encryption: Definition

A functional encryption [15, 16] scheme w.r.t. a keyed function $f_\rho(\cdot)$ consists of four algorithms (fSetup, fEnc, fKeyGen, fDec):

- $f\mathsf{Setup}(1^\lambda) \to (pk, sk)$: The probabilistic setup algorithm takes as input the security parameter λ, and outputs a pair of public-private key (pk, sk).
- $f\mathsf{Enc}(sk, \mathsf{Msg}) \to \mathsf{CT}$: The encryption scheme takes as input a private key sk and a plaintext Msg, and outputs a ciphertext CT.
- $f\mathsf{KeyGen}(sk, \rho) \to k_\rho$: The key generating algorithm takes a secret key sk and a function key ρ as input, and outputs a decryption key k_ρ.
- $f\mathsf{Dec}(pk, k_\rho, \mathsf{CT}) \to f_\rho(\mathsf{Msg})$: The decryption algorithm takes a public key pk, a decryption key k_ρ, and a ciphertext CT as input, and outputs a decrypted value, which is supposed to be $f_\rho(\mathsf{Msg})$ if CT is the ciphertext of Msg.

4.2 Functional encryption: Construction

We construct a functional encryption scheme, denoted as FE, w.r.t. the function

$$f_\rho(\mathsf{Msg}) \overset{\text{def}}{=} \varOmega^{(\rho+\varsigma)\mathsf{Msg}} \in \widetilde{\mathbb{G}} \qquad \text{where } \varOmega \in \widetilde{\mathbb{G}} \text{ and } \varsigma \in \mathbb{Z}_p \text{ are some constants.}$$

[3] The only possible exception is that the bit-length of the last block x_n is less than $\xi\lambda$.

[4] For example, the data owner Alice can choose a random nonce and set the identifier as the hash value of concatenation of Alice's account name, registed in the cloud storage server, and the nonce.

$\mathsf{KeyGen}(1^\kappa)$

Choose a bilinear map $(p, \mathbb{G}, \widetilde{\mathbb{G}}, e)$, where p is λ bits prime, both $\mathbb{G}$ and $\widetilde{\mathbb{G}}$ are cyclic multiplicative group of order p, and $e : \mathbb{G} \times \mathbb{G} \to \widetilde{\mathbb{G}}$ is a bilinear map. Choose a random generator g of group $\mathbb{G}$, a random $\alpha \xleftarrow{\$} \mathbb{Z}_p$ and set $g_1 = g^\alpha$. Choose at random $g_2, g_3, h \xleftarrow{\$} \mathbb{G}$ from $\mathbb{G}$. Choose $\tau, \varsigma \xleftarrow{\$} \mathbb{Z}_p^*$ at random. The public key is $pk := (g, g_1 = g^\alpha, g_2, g_3, h, \Omega = e(g_1, g_2))$, and the private key is $sk := (pk, g_2^\alpha, \tau, \varsigma)$. For convinience, we *implicitly* treat bilinear map $(p, \mathbb{G}, \widetilde{\mathbb{G}}, e)$ as part of pk. Output (pk, sk).

$f\mathsf{Enc}(sk, \mathsf{Msg})$

To encrypt a message $\mathsf{Msg} \in \mathbb{Z}_p^*$, find $s \in \mathbb{Z}_p^*$ such that $\mathsf{Msg} = -s\tau \mod p$, and generate the ciphertext CT as below

$$\mathsf{CT} := (\Omega^{\varsigma \mathsf{Msg}},\ g^s,\ g_3^s) \quad \in \widetilde{\mathbb{G}} \times \mathbb{G} \times \mathbb{G}.$$

Output CT.

$f\mathsf{KeyGen}(sk, \rho)$

Given a function key $\rho \in \mathbb{Z}_p^*$, choose $r \xleftarrow{\$} \mathbb{Z}_p^*$ at random, and compute the decryption key k_ρ as below

$$k_\rho := (K_0, K_1) = (g_2^{\alpha \cdot \rho\tau} \cdot g_3^r,\ g^r).$$

Output k_ρ.

$f\mathsf{Dec}(pk, k_\rho, \mathsf{CT})$

To decrypt a ciphertext CT, parse CT as $(A, B, C) \in \widetilde{\mathbb{G}} \times \mathbb{G} \times \mathbb{G}$, and compute the decrypted value as below

$$A \cdot \frac{e(K_1, C)}{e(B, K_0)} = A \cdot \frac{e(g^r,\ g_3^s)}{e(g^s,\ g_2^{\alpha \cdot \rho\tau} \cdot g_3^r)} = A \cdot \Omega^{-s\tau\rho} = \Omega^{\mathsf{Msg} \cdot (\rho + \varsigma)}$$

Output $\Omega^{\mathsf{Msg} \cdot (\rho + \varsigma)}$.

4.3 The Construction Based on Functional Encryption scheme FE

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

Run the key generating algorithm $\mathsf{KeyGen}(1^\lambda)$ of FE and obtain bilinear map $(p, \mathbb{G}, \widetilde{\mathbb{G}}, e)$ and public-private key (fpk, fsk). Choose a PRF key seed at random from the key space of the PRF $\{F_{\mathsf{seed}}\}$ and s random numbers $\alpha_1, \ldots, \alpha_s \xleftarrow{\$} \mathbb{Z}_p$. For each $j, 1 \le j \le s$, encrypt α_j using the functional encryption scheme FE to obtain a ciphertext CT_j as below:

$$\mathsf{CT}_j := f\mathsf{Enc}(fsk, \alpha_j).$$

The public key is $pk := (fpk, p, \{\mathsf{CT}_j : 1 \le j \le s\})$ and the private key is $sk := (fsk, \mathsf{seed}, \alpha_1, \ldots, \alpha_s)$.

$\mathsf{DEnc}(sk, \mathbf{X}) \rightarrow (\mathsf{id}, \hat{\mathbf{X}})$

Given the file $\mathbf{X}$, first apply the error correcting code to obtain $\mathbf{X}'$; then split $\mathbf{X}'$ into n blocks (for some n) such that each block consists of s elements from $\mathbb{Z}_p$: $\{x_{ij} \in \mathbb{Z}_p\}_{\substack{1\le i\le n\\1\le j\le s}}$. Choose a unique identifier id from domain $\{0,1\}^{2\lambda}$. Now, for each i, $1 \le i \le n$, compute an authentication tag t_i as below

$$t_i := F_{\mathsf{seed}}(\mathsf{id}, i) + \sum_{j=1}^{s} \alpha_j x_{i,j} \mod p.$$

The processed file $\hat{\mathbf{X}}$ of data file $\mathbf{X}$ consists of $\{x_{i,j}\}$, $1 \le i \le n, 1 \le j \le s$, together with authentication tags $\{t_i\}$, $1 \le i \le n$.

$\mathsf{Prove}(pk, \mathsf{id}, \hat{\mathbf{X}}, (k_\rho, C)) \rightarrow (\psi, \sigma)$

The challenge is (k_ρ, C), where k_ρ is a FE decryption key and C is an ℓ-element set $\{(i, \nu_i)\}$, with i's distinct, each $i \in [1, n]$, and each $\nu_i \in B$. Compute

$$\mu_j := \sum_{(i,\nu_i)\in C} \nu_i x_{i,j} \mod p \text{ for } 1 \le j \le s, \tag{8}$$

$$\sigma := \sum_{(i,\nu_i)\in C} \nu_i t_i \mod p. \tag{9}$$

Decrypt ciphertexts CT_j's using decryption key k_ρ to obtain

$$H_j := f\mathsf{Dec}(k_\rho, \mathsf{CT}_j), \quad 1 \le j \le s. \qquad \textit{Note: } H_j \textit{ is supposed to be } g^{(\rho+\varsigma)\alpha_j}.$$

Compute $\psi := \prod_{j=1}^{s} H_j^{\mu_j}$. Output (ψ, σ).

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

Parse C as an ℓ-element set $\{(i, \nu_i)\}$, with i's distinct, each $i \in [1, n]$, and each $\nu_i \in B$. If the following equation holds, then output accept; otherwise, output reject.

$$g^{\sigma} \overset{?}{=} \psi^{(\rho+\varsigma)^{-1} \mod p}\, g^{\sum_{(i,\nu_i)\in Q} \nu_i F_{\mathsf{seed}}(\mathsf{id}, i) \mod p}$$

Remark. We remark that, in scheme $\mathsf{EPOR\text{-}II}$, the challenge query in a verification appears in two different forms from the view of verifier and prover.

- To the verifier, the challenge query is $(\rho, C = \{(i, \nu_i)\})$, where ρ is a random nonce chosen by the verifier, and C is a set of index-coefficient pairs chosen at random by the verifier. The secret random nonce ρ is required to execute the algorithm Verify in scheme $\mathsf{EPOR\text{-}II}$.
- To the prover, the challenge query is $(k_\rho, C = \{(i, \nu_i)\})$, where k_ρ is the decryption key w.r.t. ρ generated by the verifier using the functional encryption scheme, and C is a set of index-coefficient pairs chosen at random by the verifier.

4.4 Two Unsecure Alternative Schemes

In order to explain the importance of the role of our functional encryption scheme FE in the design of scheme $\mathsf{EPOR\text{-}II}$, we introduce two alternative schemes and show that both alternative schemes are unsecure.

4.4.1 Scheme with Constant ρ

This alternative scheme is the same as our scheme EPOR-II as above, except that the value ρ is a constant. Consequently, the functional ecnryption is not necessary and this alternative scheme can be simplified further.

However, the constant value of ρ will render the alternative scheme unsecure and vulnerable to an attack: The adversary $\mathcal{A}$ obtains values of $H_j = g^{(\rho+\varsigma)\alpha_j}$, $1 \le j \le s$ from the setup or previous verifications. For each $(i,j) \in [1,n] \times [1,s]$, $\mathcal{A}$ computes $y_{i,j} := H_j^{x_{i,j}}$. $\mathcal{A}$ keeps all $y_{i,j}$'s and removes all of $x_{i,j}$'s from his stoarge. For any challenge query with $C = \{(i,\nu_i)\}$, $\mathcal{A}$ computes $\psi := \prod_{j=1}^{s} \prod_{(i,\nu_i)\in C} y_{i,j}^{\nu_i}$ and $\sigma := \sum_{(i,\nu_i)\in C} \nu_i t_i$. It is straightforward to verify that the proof (ψ,σ) indeed passes the verification of Verify with certainty. However, there is no efficient algorithm to recover data $x_{i,j}$ from $y_{i,j} = \left(g^{(\rho+\varsigma)\alpha_j}\right)^{x_{i,j}}$, assuming the discrete log problem is hard.

4.4.2 Scheme with Simple Application of Bilinear Map

The second alternative scheme is the same as our scheme EPOR-II, except that functional encryption scheme FE is replaced by a simple application of bilinear map. More precisely, the modified scheme works in this way: in the setup phase, the cloud storage server obtains values of $\{g^{\alpha_j}, 1 \le j \le s\}$, from the data owner. During a verification, the data owner playing the role of verifier, chooses a random nonce ρ and sends g^ρ together with $C = \{(i,\nu_i)\}$ to the cloud storage server. The server playing the role of prover, is supposed to compute a proof (ψ,σ) as below

$$\psi := \prod_{j=1}^{s} e(g^{\alpha_j},\ \ g^\rho)^{\mu_j}, \text{ where } \mu_j \text{ is the same as in Equation (8)} \tag{10}$$

$$\sigma := \sum_{(i,\nu_i)\in C} \nu_i t_i \mod p. \tag{11}$$

Note that Equation (11) is identical to Equation 9 in scheme EPOR-II.

However, this alternative is also suffering from attacks: The adversary $\mathcal{A}$, playing the role of prover, can compute $y_{i,j} := (g^{\alpha_j})^{x_{i,j}}$ for $(i,j) \in [1,n] \times [1,s]$, and replace each $x_{i,j}$ with $y_{i,j}$. In a verification, $\mathcal{A}$ receives g^ρ and $C = \{(i,\nu_i)\}$ from the verifier (i.e. the data owner). Then the adversary computes ψ as below

$$\psi := \prod_{j=1}^{s} e\big(\prod_{(i,\nu_i)\in C} y_{i,j}^{\nu_i},\ \ g^\rho\big) \tag{12}$$

and computes σ as in Equation (11). It is straightforward to check that the proof (ψ,σ) generated by the adversary $\mathcal{A}$ indeed passes the verification of the second alterntive scheme described above. But no efficient algorithm can recover data $x_{i,j}$ from $y_{i,j} = (g^{\alpha_j})^{x_{i,j}}$, assuming the discrete log problem is hard.

5 Extension

Our schemes allow the data owner to generate a new set of tag values w.r.t. a new key by interacting with the cloud storage server without retrieving back the data. This online refresh on tags allows the data owner to duplicate another copy of data to another cloud storage server.

RefreshTag Here we present the procedure to generate a set of fresh tags online for our first scheme.

The data owner Alice keeps a status variable η, which is initialized to 0. If η is zero, run the key generating scheme to produce a new private key $\mathcal{K}'$. Download tags $\{t_i : i \in [\eta+1, \eta+\Delta]\}$ from the Server Bob. Verify the correctness of downloaded tags: Send challenge $C = \{(i, \beta_i) : i \in [\eta+1, \eta+\Delta], \beta_i \in [1, 2^\lambda]\}$ to server, and get response (ψ_1, ψ_2). If (ψ_1, ψ_2) passes the verification in Verify and the following equality hold, then Alice believes the downloaded tags are correct.

$$\psi_2 \overset{?}{=} \sum_{(i,\beta_i)\in C} \beta_i t_i.$$

For each $i \in [\eta+1, \eta+\Delta]$, compute new tag t'_i

$$t'_i := \mathsf{Tag}_{\mathcal{K}}(x_i, i) = \alpha'\alpha^{-1}(t_i - F_{\mathsf{seed}}(i)) + F_{s'}(i) \mod p.$$

Upload new tags $\{t'_i : i \in [\eta+1, \eta+\Delta]\}$ to the Server Bob and update the state variable $\eta := \eta + \Delta$. If $\eta > N$, reset η as zero and discard private key $\mathcal{K}$ and keep only the new private key $\mathcal{K}'$.

6 Conclusion

We proposed two new efficient and secure $\mathcal{POR}$ schemes using different techniques. Our first scheme is the first secure $\mathcal{POR}$ scheme over integer domain, and our second scheme is the most efficient $\mathcal{POR}$ scheme and relys on our new functional encrypton scheme. We believe that our construction of functional encryption scheme may have independent interests.

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. 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
4. 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
5. 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
6. 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
7. : Proofs of Retrievability via Hardness Amplification. In: TCC. (2009) 109–127
8. 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
9. 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
10. 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
11. 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
12. 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
13. 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

14. Gentry, C.: Fully Homomorphic Encryption using Ideal Lattices. In: STOC '09: ACM symposium on Theory of computing. (2009) 169–178
15. Boneh, D., Sahai, A., Waters, B.: Functional Encryption: Definitions and Challenges. In: (*will appear in*) TCC '11: Theory of Cryptography Conference. (2011) `http://eprint.iacr.org/2010/543`.
16. O'Neill, A.: Definitional issues in functional encryption. Cryptology ePrint Archive, Report 2010/556 (2010) `http://eprint.iacr.org/`.
17. Schoenfeld, L.: Sharper Bounds for the Chebyshev Functions $\theta(x)$ and $\psi(x)$. II. Mathematics of Computation **30**(134) (1976) 337–360
18. Sahai, A., Vadhan, S.: A Complete Problem for Statistical Zero Knowledge. Journal of the ACM **50** (March 2003) 196–249

A 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 combinatiorial techniques.
– Retrievability: Using ECC 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 proofs for our two schemes inherit this property, i.e. our proofs also share the second 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.

A.1 Theorem for Part-Two Proof

Theorem 5 (Shacham and Waters [1], Theorem 4.3) *Let $\mathcal{A}^{\mathsf{Retrive}}$ 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 $\mathcal{A}^{\mathsf{Retrive}}$ on file F is well-behaved: i.e. any valid proof generated by $\mathcal{A}^{\mathsf{Retrive}}$ has to be correct (with probability 1), and is ϵ-admissible: i.e. convingcingly 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 nonnegligible, it is possible to recover a ρ fraction of the encoded file blocks in $\mathcal{O}(n/(\epsilon-\omega))$ interactions with $\mathcal{A}^{\mathsf{Retrive}}$ 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.

A.2 Theorem for Part-Three Proof

Theorem 6 (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.*

B Our Tag function is Secure

Lemma 1 *Assume $F_{\mathsf{seed}}(\cdot)$ is a random oracle. Let $Q = p(1+\epsilon)$ (alternatively, $Q = p(2-\epsilon)$) for some negligible ϵ. Then the distribution of output of* Tag *is statistical close to a uniform distribution over integer interval* $[0, Q-1]$.

Note that all of p, Q and ϵ are considered as functions of the security parameter λ.

Proof. Let T_p denote the random variable generated by Tag with key on any input. Under assumption that $F_{\mathsf{seed}}(\cdot)$ is random oracles, we have

$$\forall x \in [0, 2^{\xi\lambda} - 1], \mathsf{Pr}[T_p = a] = \begin{cases} \frac{1}{2p} & (a \in [0, Q-p-1]); \\ \frac{1}{p} & (a \in [Q-p, p-1]); \\ \frac{1}{2p} & (a \in [p, Q-1]). \end{cases} \tag{13}$$

Denote with U_Q a uniform random variable over integer domain $[0, Q-1]$. The statistical difference between U_Q and T_p (for any x in the domain) is as below

$$\mathsf{SD}(U_Q, T_p) := \frac{1}{2} \sum_{a \in [0, Q-1]} |\mathsf{Pr}[U_Q = a] - \mathsf{Pr}[T_p = a]| \tag{14}$$

$$= \frac{1}{2} \frac{(Q-p)(2p-Q)}{pQ} \tag{15}$$

Part I: $Q = p(1+\epsilon)$.
Substituting $Q = p(1+\epsilon)$ into equation (15), we have the statistical difference

$$\mathsf{SD}(U_Q, T_p) = \frac{\epsilon - \epsilon^2}{1+\epsilon} \approx \epsilon, \tag{16}$$

which is negligible.

Part II: $Q = p(2-\epsilon)$.
Alternatively, substituting $Q = p(2-\epsilon)$ into equation (15), we have the statistical difference

$$\mathsf{SD}(U_Q, T_p) = \frac{\epsilon - \epsilon^2}{2(2-\epsilon)} \approx \frac{\epsilon}{4}, \tag{17}$$

which is negligible, since ϵ is negligible.

□

Lemma 2 *Assuming that the Riemann hypothesis holds, there exist more than $\frac{1}{4\pi}\sqrt{Q}\ln Q$ primes in the interval $(Q_0, Q]$ where $Q_0 = Q - \frac{1}{2\pi}\sqrt{Q}\ln^2 Q$ and $Q_0 \geq 2657$.*

Proof. Let $\pi(x)$ denote the number of primes less than or equal to x. Then the number of primes in the interval $(Q_0, Q]$ is $\pi(Q) - \pi(Q_0)$.

Lowell Schoenfeld [17] proved that, assuming the Riemann hypothesis,

$$\forall x \geq 2657, \quad |\pi(x) - \mathrm{li}(x)| < \frac{\sqrt{x}\ln x}{8\pi}, \tag{18}$$

where $\mathrm{li}(x)$ is the logarithmic integral function:

$$\mathrm{li}(x) = \int_0^x \frac{dt}{\ln t} \qquad (t \neq 1) \tag{19}$$

Applying Lowell Schoenfeld's inequality (18) on $x = Q$ and $x = Q_0$, we have

$$\begin{aligned}
|(\pi(Q) - \mathrm{li}(Q)) - (\pi(Q_0) - \mathrm{li}(Q_0))| &\leq |(\pi(Q) - \mathrm{li}(Q))| + |(\pi(Q_0) - \mathrm{li}(Q_0))| && (20)\\
&< \frac{\sqrt{Q}\,\ln Q}{8\pi} + \frac{\sqrt{Q}_0\,\ln Q_0}{8\pi} && (21)\\
&< \frac{\sqrt{Q}\,\ln Q}{4\pi}. && (22)
\end{aligned}$$

Now we bound the difference $\mathrm{li}(Q) - \mathrm{li}(Q_0)$:

$$\mathrm{li}(Q) - \mathrm{li}(Q_0) = \int_{Q_0}^{Q} \frac{dt}{\ln t} > \int_{Q_0}^{Q} \frac{dt}{\ln Q} = \frac{Q - Q_0}{\ln Q}. \tag{23}$$

By combining Equation (22) and Equation (23), we conclude that the number of primes in $(Q_0, Q]$ is

$$\begin{aligned}
\pi(Q) - \pi(Q_0) &> \mathrm{li}(Q) - \mathrm{li}(Q_0) - \frac{\sqrt{Q}\,\ln Q}{4\pi} && (24)\\
&= \frac{Q - Q_0}{\ln Q} - \frac{\sqrt{Q}\,\ln Q}{4\pi} && (25)\\
&= \frac{\sqrt{Q}\,\ln Q}{4\pi} \qquad (\text{ By substituting } Q_0 = Q - \frac{1}{2\pi}\sqrt{Q}\ln^2 Q). && (26)
\end{aligned}$$

Theorem 3 *Let $\mathcal{K} = (p, Q, \alpha, s)$ be the key of Tag, such that $Q = 2^{2(\lambda+\varsigma)}$ and p is a prime chosen at random from a subset P of interval $(Q_0, Q]$ with $Q_0 = Q - \frac{1}{2\pi}\sqrt{Q}\ln^2 Q$. Assume $F_{\mathsf{seed}}(\cdot)$ is random oracle. Given value of Q and a set $S = \{(x, \mathsf{Tag}_{\mathcal{K}}(x)) \in [0, 2^{\xi\lambda} - 1] \times [0, Q-1]\}$ of data-tag pairs, any (computationally unbounded) adversary cannot find p with probability larger than $|S_U|(Q/p - 1) + 1/(\#Prime)$, where $\#Prime$ denotes the number of primes in the subset P of interval $(Q_0, Q]$.*

Proof. Without knowing the secret key $\mathcal{K}$, a tag value $\mathsf{Tag}_{\mathcal{K}}(x))$ is independent on the input x.

Let random variable S_p denote the resulting set S conditional on prime p and random variabl S_U denote the resulting set S conditional on that the output of Tag is truely uniformly random over $[0, Q-1]$. Let random variable X represent the input x. By applying Fact 2.3 of Sahai *et al.* [18], we have

$$\mathsf{SD}(S_U, S_p) < |S_U| \times \mathsf{SD}((X, U_Q), (X, T_p)) \leq |S_U| \times \mathsf{SD}(U_Q, T_p) \approx \epsilon|S_U|.$$

Let X denote the random variable generated in this way: Choose a prime P from interval $(Q_0, Q]$ at random and then sample X from S_p. We show an upper bound for the maximum likelihood $\mathsf{ML}(p)$. Let $\mathcal{P}$ be the set of primes in $(Q_0, Q]$. The best strategy: Given any $X = a$, output the prime p_a that maximize $\mathsf{Pr}[X = a|S_p]$ among all primes in $(Q_0, Q]$.

$$\begin{aligned}
\mathsf{ML}(p) &= \sum_a \mathsf{Pr}[X = a]\mathsf{Pr}[P = p_a|X = a] && (27)\\
&= \sum_a \mathsf{Pr}[X = a, P = p_a] && (28)\\
&= \sum_a \mathsf{Pr}[P = p_a]\mathsf{Pr}[X = a|P = p_a] && (29)\\
&= \frac{1}{|\mathcal{P}|}\sum_a \mathsf{Pr}[X = a|P = p_a] && (30)\\
&\leq \frac{1}{|\mathcal{P}|} \times \frac{1}{Q_0} \times Q \quad (\text{ Since } \mathsf{Pr}[X = a|S_p] \leq \frac{1}{Q_0} \text{ for all } p) && (31)
\end{aligned}$$

□

C Our Functional Encryption Scheme FE

D Scheme $\mathsf{EPOR\text{-}I}$ is Secure

Here we provide the part one proof for our Scheme I. Informally, our part one proof shows: *The proof to a changle query in Scheme I is unforgeable, in the sense that no PPT adversary can find a valid but not correct proof with nonnegligible probability.*

Lemma 7 *For any PPT adversary $\mathcal{A}$, the probability that $\mathcal{A}$ can output a proof (ψ_1', ψ_2') such that the* Verify *algorithm accepts (ψ_1', ψ_2') and (ψ_1', ψ_2') is different from the correct proof generated by the algorithm* Prove *upon the same data file and the same challenge query, is negligible.*

More precisely, let $\mathcal{A}^{\mathsf{Retrive}}$ denote the adversary $\mathcal{A}$ in the $\mathsf{Retrieve}$ *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 $\mathsf{Game}^{\mathcal{E}}_{\mathcal{A}}$, $\mathbf{X}$ be the data file associated to id and $(\mathsf{id}, \hat{\mathbf{X}})$ be the output that $\mathcal{A}$ obtains from the challenger upon store query $(\mathbf{X})$. For any PPT adversary $\mathcal{A}$,*

$$\mathsf{Adv}^{\mathcal{E},\mathsf{forge}}_{\mathcal{A}}(\lambda) = \mathsf{Pr}\left[\begin{array}{l} C \xleftarrow{\$} \left([1,n] \times [1, 2^{\lambda}]\right)^{\ell}; \\ (\psi_1, \psi_2) \leftarrow \mathsf{Prove}(pk, \mathsf{id}, \hat{\mathbf{X}}, C); \\ (\psi_1', \psi_2') \leftarrow \mathcal{A}^{\mathsf{Retrive}}(pk, \mathsf{id}, C): \\ \qquad\qquad \mathsf{Verify}(sk, \mathsf{id}, C, \psi_1', \psi_2') = \mathsf{accept} \ \wedge \ (\psi_1', \psi_2') \neq (\psi_1, \psi_2) \end{array}\right] \leq negl(\lambda). \tag{32}$$

The probability is over all random coins and the choice of the challenge query.

Proof.
Game 1. The first game is just the one specified in Lemma 7, 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 $\mathsf{Adv}^{\mathcal{E},\mathsf{forge}}_{\mathcal{A}} = \mathsf{Pr}[\mathcal{A}$ wins **Game 1**$]$.
Game 2. The second game is the same as Game 1, except that in the scheme $\mathcal{E}$, the PRF function $F_{\mathsf{seed}}(\cdot)$ is evaluated in the following way:

- The challenger keeps a table to store all previous encounted $(v, F_{\mathsf{seed}}(v))$ 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 ranom u from the range of F_{seed}, insert $(v, F_{\mathsf{seed}}(v) := u)$ 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 $\mathsf{Verify}(sk, \mathsf{id}, C, \psi_1', \psi_2') = \mathsf{accept}$ and $(\psi_1', \psi_2') \neq (\psi_1, \psi_2) \mod p$. Note the additional modulo p operation.
Game 4 The fourth game is the same as **Game 3**, except that adversary $\mathcal{A}$ wins if and only if $\mathsf{Verify}(sk, \mathsf{id}, C, \psi_1', \psi_2') = \mathsf{accept}$ and $(\psi_1', \psi_2') \neq (\psi_1, \psi_2)$ and $(\psi_1', \psi_2') = (\psi_1, \psi_2) \mod p$. If $\mathcal{A}$ wins **Game 4**, then she can find a non-zero multipler ($\psi_1 - \psi_1'$ or $\psi_2 - \psi_2'$) of the secret prime p.

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 D.01 *If there is a 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.*

Claim D.02 *If adversary $\mathcal{A}$ wins* **Game 3** *with non-negligible probability, then $\mathcal{A}$ can break the Theorem 4.1 of of [1], i.e. the unforgeability of the first scheme (Scheme with Private Verification).*

Claim D.03 *If adversary $\mathcal{A}$ wins* **Game 4** *with non-negligible probability, then $\mathcal{A}$ breaks Theorem 3.*

E Semi-Functional Encryption Scheme

$\mathsf{Enc}(sk, M)$

Given a message $M \in \widetilde{\mathbb{G}}$, choose a random coin $s \xleftarrow{\$} \mathbb{Z}_p$ and produce a ciphertext CT as below

$$\mathsf{CT} := (A, B, C) = (M \cdot e(t, g)^s,\ g^s,\ g_3^s) \in \widetilde{\mathbb{G}} \times \mathbb{G} \times \mathbb{G}.$$

$\mathsf{KeyGen}(sk, \rho)$

Given a function key ρ, choose a random coin $r \xleftarrow{\$} \mathbb{Z}_p$ and generate the decryption key k_ρ as below

$$k_\rho := (K_0, K_1) = (g_3^r \cdot t,\ g^r)$$

$\mathsf{Dec}(k_\rho, \mathsf{CT})$

Parse the ciphertext CT as $(A, B, C) \in \widetilde{\mathbb{G}} \times \mathbb{G} \times \mathbb{G}$. Compute and ouput

$$A \cdot \frac{e(K_1, C)}{e(B, K_0)} = A \cdot \frac{e(g^r, g_3^s)}{e(g^s, g_3^r \cdot t)} = \frac{A}{e(g, t)^s} = M$$

Lemma 8 *Ciphertexts produced by* $f\mathsf{Enc}$ *and* Enc *can (with proper input) be indistinguishable and have the same decrypted value.*

F Scheme **EPOR-II** is Secure in the no-feedback setting

Game 1. The original game in the no-feedback model.
Game 2. Same as Game 1, except that adversary does not make verification queries, but still makes store queries.
Game 3. Same as Game 2, except that the functional encryption is replaced with the alternative encryption.
Game 4. Same as Game 3, except that the alternative encrption scheme is replaced with plaintexts, i.e. the verifier sends $(g^{\rho\alpha_1}, \ldots, g^{\rho\alpha_s})$ in cleartext to the prover in a verification.

Our proof sketch is described as below, where $\approx_{negl}$ means the difference between the left and right hand sides are negligible.

$$\mathsf{Adv}_{\mathcal{A}}^{\textbf{Game 1}} = \mathsf{Adv}_{\mathcal{A}}^{\textbf{Game 2}} \approx_{negl} \mathsf{Adv}_{\mathcal{A}}^{\textbf{Game 3}} \leq \mathsf{Adv}_{\mathcal{A}}^{\textbf{Game 4}}.$$

Adversary wins **Game 4** with negligible probability, under diffie-hellman assumption.