Towards Efficient Provable Data Possession in Cloud Storage

Jia Xu[1], Ee-Chien Chang[2], and Jianying Zhou[1]

[1] Insitute for Infocomm Research
{xuj,jyzhou}@i2r.a-star.edu.sg
[2] National University of Singapore
changec@comp.nus.edu.sg

Abstract. Provable Data Possession ($\mathcal{PDP}$) allows data owner to periodically and remotely audit integrity of their data stored in a cloud storage, without retrieving the file and without keeping a local copy. Ateniese *et al.* (CCS 07, ACM TISS '11) proposed the first $\mathcal{PDP}$ scheme, which is very efficient in communication and storage. However their scheme requires a lot of group exponentiation operations: In the setup, one group exponentiation is required to generate a tag per each data block. In each verification, (equivalently) $(m + \ell)$ group exponentiations are required to generate a proof, where m is the size of a data block and ℓ is the number of blocks accessed during a verification. This paper proposed an efficient $\mathcal{PDP}$ scheme. Compared to Ateniese *et al.* (CCS 07, ACM TISS '11), the proposed scheme has the same complexities in communication and storage, but is much more efficient in computation: In the setup, no expensive group exponentiations are required. In each verification, only m group exponentiations are required to generate a proof. Our experiment shows that our scheme is about 400 times faster than Ateniese *et al.* (CCS 07, ACM TISS '11) in the setup phase. The security of the proposed scheme is proved under Knowledge of Exponent Assumption and Factorization Assumption.

Keywords: Cloud Storage, Remote Integrity Check, Provable Data Possession, Homomorphic Authentication Tag

1 Introduction

Cloud storage service (e.g. Dropbox, Skydrive, Google Drive, and Amazon S3) is becoming more and more popular in recent years. How to resolve people's concern on integrity of their data stored in cloud storage server is one of major challenge to boost the adoption of cloud storage service for the public and organizations. Ateniese *et al.* [5], together with Juels and Kaliski [33], starts the new trend of research on remote data integrity check in cloud computing environment. Due to the new cloud computing setting, this line of research has distinctive features compare to the previous works [3,8,36,12]: (1) without trusting the cloud storage server (in both data integrity and integrity check); (2) without downloading the target file during verification; (3) without keeping a local copy of the target file. This new remote data integrity check tool will help to build ultimate data integrity protection solution for cloud storage service (e.g. [14,19]).

Ateniese *et al.* [5,4] proposed the first Provable Data Possession ($\mathcal{PDP}$ for short) scheme. Their scheme is very efficient in communication and storage: the size of a proof is independent on the number of blocks accessed during a verification and the storage overhead due to authentication tags is a fraction[3] of the size of the original data. However, their scheme requires a large number of modular exponentiation in both setup phase and verification phase, and is thus relative expensive in computation.

In this paper, we will propose a new $\mathcal{PDP}$ construction named EPOS, which requires no modular exponentiation in the setup phase and a smaller number of group exponentiations in

[3] This fraction is a configurable system parameter.

verification phase, without sacrificing in communication or storage aspects. We prove the security of the proposed scheme EPOS under Knowledge of Exponent Assumption [20]. We remark that both Ateniese *et al.* [5,4] [4] and the proposed scheme EPOS support only private key verification.

1.1 A Brief Description of EPOS

Setup Phase. Suppose Alice wants to backup her file $\mathtt{F}$ into a cloud storage server provided by Bob. Alice encodes file $\mathtt{F}$ with some error erasure code to obtain data blocks $(\mathtt{F}_0, \ldots, \mathtt{F}_{n-1})$. Alice chooses a RSA modulus $N = pq$, a secret seed, denoted as seed, of a pseudorandom function PRF, and a secret number τ. Let $\phi(N) = (p-1)(q-1)$. With the secret private key $sk = (\phi(N), \mathsf{seed}, \tau)$, Alice produces an authentication tag σ_i for each block $\mathtt{F}_i$:

$$\sigma_i := \tau \mathtt{F}_i \ + \ \mathsf{PRF}_{\mathsf{seed}}(i) \mod \phi(N). \tag{1}$$

We emphasize that the generated authentication tag σ_i is much shorter than a data block $\mathtt{F}_i$. At the end of setup, Alice sends data blocks and tags $\{(i, \mathtt{F}_i, \sigma_i) : i \in [0, n-1]\}$ together with a public key $pk = (N)$ to Bob.

Audit. Later, Alice may remotely verify the integrity of her data file stored with Bob periodically. In each verification session, Alice randomly selects a subset $C \subset [0, n-1]$ of indices and selects a random weights ν_i for each $i \in C$. Alice sends $\{(i, \nu_i) : i \in C\}$ as challenge to Bob. Bob then finds all data blocks $\mathtt{F}_i$'s and authentication tags σ_i's with index $i \in C$, and apply the linear homomorphism to compute an aggregated message-tag pair $(\mathtt{M}, \sigma)$ as below:

$$\mathtt{M} := \sum_{i \in C} \nu_i \mathtt{F}_i; \tag{2}$$

$$\sigma := \sum_{i \in C} \nu_i \sigma_i. \tag{3}$$

We emphasize that the above two equations are computed over integer domain, and thus the bit-length of the linear combination $\mathtt{M}$ (the aggregated authentication tag σ, respectively) is slightly larger than a data block $\mathtt{F}_i$ (an authentication tag σ_i, respectively).

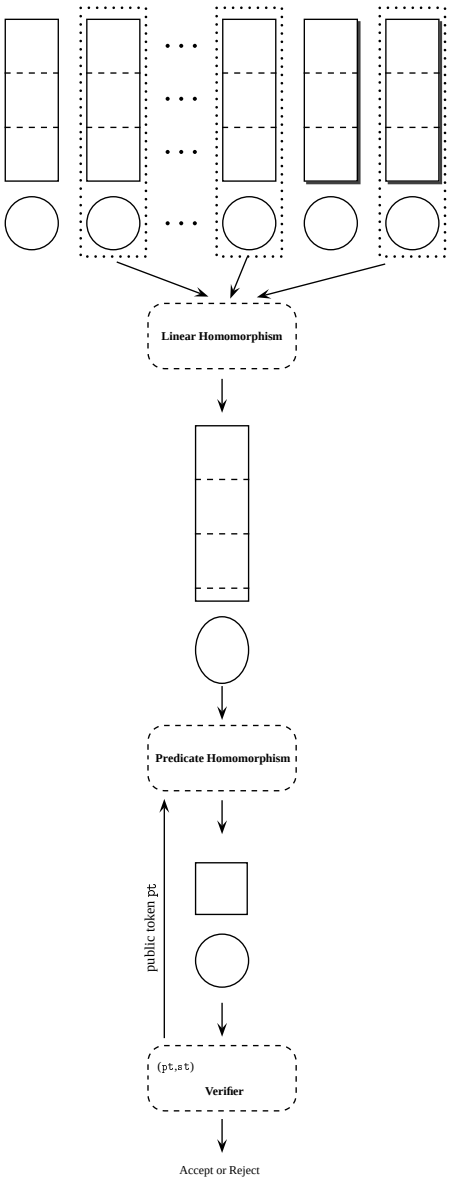

Fig. 1. An Efficient $\mathcal{PDP}$ scheme EPOS

Instead of sending the large message block $\mathtt{M}$ together with authentication tag σ directly to the verifier Alice, Bob is able to produce a *shorter* data-tag pair with the help of Alice. Alice generates a pair of public token pt and secret token st per each verification, where the public

[4] See the erratra in their full version http://eprint.iacr.org/2007/202 by Ateniese *et al.*

token pt is sent to the prover Bob and the secret token st is kept private. With pt and $(\mathtt{M}, \sigma)$, Bob is able to generate a *shorter* message-tag pair, which can be verified by the verifier Alice with the private key and the secret token st.

Illustration Picture Figure 1 illustrates the scheme EPOS briefed above. In Figure 1, a rectangle and the circle beneath it, represents a pair of data block and authentication tag. Those shaded rectangles represent data blocks that are generated by the error erasure code. In our scheme EPOS, a data block is treated as a single large integer (much larger than the RSA modulus N). Figure 1 shows an example where a data block is about three times larger than a tag, by dividing each rectangle with dashed lines.

In a verification, a subset of three pairs of blocks and tags are selected, which are aggregated into a single pair of block and tag through linear homomorphism. Since the linear combination is computed over integer domain, the aggregated block (tag, respectively) is slightly larger than an original data block (tag, respectively). With the help of the public token pt provided by the verifier, a shorter block-tag pair can be generated from the long aggregated block-tag pair. The verifier can verify the short block-tag pair using a secret token st.

1.2 Contributions

Our contributions can be summarized below.

- We propose a new $\mathcal{PDP}$ scheme, called EPOS. This scheme significantly improves the efficiency of Ateniese *et al.* [5,4]. A detailed comparison with existing works is given in Table 1 and Table 2 in Section 5 (on page 9).
- We give a rigorous proof on the security of the proposed scheme, based on Knowledge of Exponent Assumption (KEA).

1.3 Organization

The rest of this paper is organized as below. Section 2 reviews related works. Section 3 describes the syntactic definition of $\mathcal{PDP}$. Section 4 presents the construction of our $\mathcal{PDP}$ scheme EPOS. Then we analyze the performance of proposed scheme in Section 5 and security in Section 6. At the end, Section 7 closes this paper.

2 Related Works

2.1 Early Approaches

Our research is motivated by applications in remote-backup and peer-to-peer back-up [3,8,36]. Peer-to-peer backup system requires a mechanism to maintain the availability and integrity of data stored in peer nodes. Li and Dabek [36] proposed to choose neighboring nodes based on the social relationships and relies on the heuristic assumption that people are more likely cooperative with friends.

2.2 Online Memory Checker and Sublinear Authenticator

Remote integrity verification has a close relationship with memory integrity verification [12,44,38,24]. The notion of authenticator proposed by Naor and Rothblum [38] is formulated for memory integrity checker. There is an essential difference between memory checker and proofs of storage problem studied in this paper: in the memory checker problem, an honest prover will follow the specified protocol to verify its storage, where the storage is untrusted and could be altered by outside attackers or random hardware failure; in the proofs of storage problem, both the prover and its storage are untrusted, such that the prover could do *anything*[5] during a verification and the storage could be altered carefully by the dishonest prover. Consequently, any solution to a proofs of storage problem is also a solution to the memory checker problem. Thus, the lower bound on complexity of memory checker discovered by Naor and Rothblum [38] also applies to proofs of storage. Additionally, the idea of introducing redundancy to tradeoff resources is useful in proofs of storage.

2.3 Proofs of Retrievability and Provable Data Possession

Recently, there is a growing interest in the cryptographic aspects of cloud storage problem. Perhaps Filho and Barreto [26] first studied the scenario where the verifier does not have the original. They described two potential applications: *uncheatable data transfer* and *demonstrating data possession*, and proposed the RSA-based scheme. Juels and Kaliski [33] proposed a formulation called Proofs of Retrievability $\mathcal{POR}$ for the proofs of storage problem. Essentially, in a $\mathcal{POR}$ scheme, if the cloud storage server can pass verification with a noticeable probability, then the verifier can *retrieve* the original data from messages collected during polynomially many verification interactions between the verifier and the cloud storage server. So $\mathcal{POR}$ formulation allows a user to ensure whether his/her file is indeed in the cloud storage in an intact form without actually downloading the file. However, the $\mathcal{POR}$ construction in Juels and Kaliski [33] can support only a predefined constant number of verifications. A refined security formulation is given in [15] .

Ateniese *et al.* [5] gave an alternative formulation called *Provable Data Possession* for proofs of storage problem, and proposed an efficient construction. Their method can be viewed as an extension of the RSA-based scheme. The scheme EPOS proposed in this paper exploits similar idea, which is much more efficient than Ateniese *et al.* [5].

Shacham and Waters [42] proposed two efficient constructions of $\mathcal{POR}$, where one scheme supports private key verification and the other supports public key verification.

Ateniese and Kamara and Katz [7] studied how to utilize homomorphic linear identification scheme to construct proofs of storage scheme. Dodis and Vadhan and Wichs [23] studied how to construct proofs of retrievability scheme through hardness application. All of schemes in [5,17,42,7] utilize some underlying linear homomorphic authentication methods, which also has applications in network coding [2,13]. Several proofs of storage schemes with pre-defined number of verifications have been proposed in works [33,6,23]. A survey of proofs of storage is given by Yang and Jia [48].

2.4 Proofs of Storage with More Features

Very recently, several works [19,14,25,46,45] have devoted to extend proofs of storage to support more features. In [19], verifier checks whether the cloud storage server indeed keeps multiple

[5] The only limitation is that the prover's computation resource is polynomially bounded.

intact copies of a user's file. Dynamic-$\mathcal{PDP}$ [25] allows insertion and deletion of data blocks on the fly after setup. Proofs of storage schemes supporting public verifiability are proposed in Shacham and Waters [42] and Wang [46] and the privacy issue in public verification is studied in Wang [45]. Very recently, dynamic $\mathcal{POR}$ is studied by Cash *et al.* [16] and Shi [43].

2.5 More General Delegated Computation and Proofs of Storage

Kate and Zaverucha and Goldberg [34] proposed an efficient commitment scheme for polynomial and Benabbas and Gennaro and Vahlis [11] proposed a secure delegation scheme for polynomial evaluation. Both schemes can be extended to support $\mathcal{POR}$ easily but with limitations: the $\mathcal{POR}$ scheme implied in Kate and Zaverucha and Goldberg [34] has large storage cost on client side and the $\mathcal{POR}$ scheme implied in Benabbas and Gennaro and Vahlis [11] has large storage and computation cost on the server side.

The two solutions [27,18] to verifiable delegation of generic computation task based on fully homomorphic encryption [28], also imply a secure proofs of storage scheme. However, the efficiency overheads in communication, storage and computation on the server side are too large, rendering the resulting proofs of storage schemes impractical.

3 Provable Data Possession: Definition

We will give the syntactic definition of Provable Data Possession scheme below, and leave the security formulation to Section 6 where we will analyze the security feature of the proposed scheme.

A $\mathcal{PDP}$ scheme consists of five polynomial algorithms (KeyGen, $\mathsf{DEncode}$, $\mathsf{Challenge}$, Prove, Verify), which are described as below.

- $\mathsf{KeyGen}(1^\lambda) \rightarrow (pk, sk)$: Given security parameter λ, the randomized key generating algorithm outputs a public-private key pair (pk, sk).
- $\mathsf{DEncode}(sk, \mathtt{F}) \rightarrow (\mathsf{id}_\mathtt{F}, n, \hat{\mathtt{F}})$: Given the private key sk and a data file $\mathtt{F}$, the data encoding algorithm $\mathsf{DEncode}$ produces a unique identifier $\mathsf{id}_\mathtt{F}$, file size n (in term of number of blocks) and the encoded file $\hat{\mathtt{F}}$.
- $\mathsf{Challenge}(sk, \mathsf{id}, n) \rightarrow (\mathsf{pt}, \mathsf{st}, \mathsf{Chall})$: The probabilistic algorithm $\mathsf{Challenge}$ takes as input the private key sk, a file identifier id and the file size n (in term of number of blocks), and outputs a public token pt, a private token st and a query Chall.
  Note: (1) The public/secret token pair (pt, st*) is generated independently per each verification. (2) In both [5] and the scheme* EPOS *that will be presented later in this paper,* Chall *is just a subset of indices (in the range* $[0, n-1]$*) and weights (from some group), and has no secret information involved.*
- $\mathsf{Prove}(pk, \mathsf{id}_\mathtt{F}, \hat{\mathtt{F}}, \mathsf{pt}, \mathsf{Chall}) \rightarrow \psi$: Given the public key pk, an identifier $\mathsf{id}_\mathtt{F}$, an encoded file $\hat{\mathtt{F}}$, a public token pt and a challenge query Chall, the prover algorithm Prove produces a proof ψ.
- $\mathsf{Verify}(sk, \mathsf{id}_\mathtt{F}, \mathsf{st}, \mathsf{Chall}, \psi) \rightarrow$ `accept` or `reject`: Given the private key sk, an identifier $\mathsf{id}_\mathtt{F}$, the secret token st, a challenge query Chall, and a proof ψ, the deterministic verifying algorithm Verify will output either `accept` or `reject`.

Remark 1 *Compared to the $\mathcal{POR}$ formulation [33,42], in the above description for $\mathcal{PDP}$, the prover algorithm* Prover *takes as an additional input a public token* pt *and the verifier algorithm* Verify *takes a corresponding secret token* st *as an additional input, where the public/secret token pair* $(\mathsf{pt}, \mathsf{st})$ *is generated online by the verifier for each verification session.*

4 EPOS: An Efficient $\mathcal{PDP}$ Scheme

4.1 Scheme Description

Ateniese [5,4]'s $\mathcal{PDP}$ scheme requires the data owner to compute an expensive modular exponentiation for each data block to generate an authentication tag in the setup. Now we present an efficient $\mathcal{PDP}$ scheme, which removes the demand of expensive exponentiation operations in the setup.

The description of scheme EPOS = (KeyGen, $\mathsf{DEncode}$, $\mathsf{Challenge}$, Prove, Verify) is as below.

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

Choose at random a λ bits RSA modulus $N = pq$, such that all of $p, q, p' = (p-1)/2, q' = (q-1)/2$ are primes and the bit-lengths of p and q are the same. Let $\phi(N) = (p-1)(q-1) = 4p'q'$. Let $\mathcal{QR}_N$ denote the subgroup of quadratic residues modulo N. Choose at random a generator g of the subgroup $\mathcal{QR}_N$. Choose at random $\tau \xleftarrow{\$} \mathbb{Z}_{\phi(N)}$. Choose at random a seed, denoted as seed, from the key space of the pseudorandom function $\mathsf{PRF} : \{0,1\}^{2\lambda} \rightarrow \mathbb{Z}_{\phi(N)}$. The public key is $pk := (N, g)$ and the private key is $sk := (g, p, q, \tau, \mathsf{seed})$.
Note: (1) Without loss of generality, assume $p' < q'$. (2) Then size of subgroup $\mathcal{QR}_N$ equals to $\frac{1}{4}\phi(N) = p'q'$.

$\mathsf{DEncode}(sk, \mathtt{F}) \rightarrow (\mathsf{id}, \{(i, \mathtt{F}_i, \sigma_i) : i \in [0, n-1]\})$

Let $\rho \in (0,1)$ be a system parameter. Apply rate-ρ error erasure code on data file $\mathtt{F}$ to generate blocks $(\mathtt{F}_0, \ldots, \mathtt{F}_{n-1})$, such that each block $\mathtt{F}_i \in \{0,1\}^{m\lambda}$ and any ρ-fraction of blocks $\mathtt{F}_i$'s can recover the original file $\mathtt{F}$. Choose a unique identifier id for the file $\mathtt{F}$. For each data block $\mathtt{F}_i, i \in [0, n-1]$, the data owner computes an authentication tag σ_i:

$$\sigma_i := \tau \mathtt{F}_i + \mathsf{PRF}_{\mathsf{seed}}(\mathsf{id} \| i) \mod \phi(N). \tag{4}$$

Output $(\mathsf{id}, \{(i, \mathtt{F}_i, \sigma_i) : i \in [0, n-1]\})$.

$\mathsf{Challenge}(sk, \mathsf{id}, n) \rightarrow (\mathsf{pt}, \mathsf{st}, \{(i, \nu_i) : i \in C\})$

Find at random a secret value $d \xleftarrow{\$} \mathbb{Z}_{\phi(N)}$, and computes $g_d := g^d \mod N$. The public token is $\mathsf{pt} := g_d$ and the secret token is $\mathsf{st} := d$. Chooses a subset $C \subset [0, n-1]$ at random and choose weight $\nu_i \xleftarrow{\$} \mathbb{Z}_{\phi(N)}$ at random for each $i \in C$. Output $(\mathsf{pt}, \mathsf{st}, \{(i, \nu_i) : i \in C\})$.

$\mathsf{Prove}(pk, \mathsf{id}, \hat{\mathtt{F}}, \mathsf{pt}, \{(i, \nu_i) : i \in C\}) \rightarrow (\psi_1, \psi_2)$

Find all selected blocks $\mathtt{F}_i$'s and tags σ_i's, and compute (π_1, π_2) as below over integer domain (*instead of finite field*):

$$\pi_1 := \sum_{i \in C} \nu_i \mathtt{F}_i; \tag{5}$$

$$\pi_2 := \sum_{i \in C} \nu_i \sigma_i. \tag{6}$$

Compute (ψ_1, ψ_2) as below

$$\psi_1 := g_d^{\pi_1} \mod N; \tag{7}$$

$$\psi_2 := g^{\pi_2} \mod N. \tag{8}$$

Send (ψ_1, ψ_2) to the verifier.

Note: Alternatively, the prover can just send (π_1, π_2) as response to the verifier. In this alternative way, the computation in Eq (7) is saved, at the cost of larger proof size, since the bit-length of (π_1, π_2) is much longer than (ψ_1, ψ_2).

$\mathsf{Verify}(sk, \mathsf{id}, \mathsf{st}, \{(i, \nu_i)\}_{i \in C}, \psi_1, \psi_2) \rightarrow$ `accept/reject`

With the private key $sk = (g, p, q, \tau, \mathsf{seed})$ and the secret token $\mathsf{st} = d$, check whether $\psi_1 \in \mathcal{QR}_N$ is a quadratic residue modulo N and the following equality holds.

$$(\psi_1)^{\tau} \stackrel{?}{=} \left(\frac{\psi_2}{g^{\sum_{i \in C} \nu_i \mathsf{PRF}_{\mathsf{seed}}(\mathsf{id} \| i)}} \right)^d \mod N \tag{9}$$

If both verifications succeed, then output `accept`; otherwise output `reject`.

We remark that in the above scheme, we can change[6] the proof from (ψ_1, ψ_2) to $(\psi_1, \mathtt{SHA256}(\psi_2))$ to reduce the size from 2λ bits to $(\lambda + 256)$ bits using a secure hash function `SHA256` [39], similar to Ateniese *et al.* [5].

4.2 Discussions on Possible Extensions

4.2.1 Dynamic PDP. It is easy to see that, similar to Ateniese *et al.* [5] and SW scheme [42], our proposed $\mathcal{PDP}$ scheme naturally supports a simple dynamic operation: Append a new block at the end. A generic approach can help us to support other dynamic operations (i.e. `insertion`,`edition` and `deletion`): (1) **Modification Step**—Data owner treats the *difference*[7] between any two consecutive version of the file as new data block(s)[8]. The data owner generates authentication tags for these new bocks, and appends them at the end of original file stored in the cloud. We assume the data owner has marked the end of original data file. (2) **Refresh Step**—When the amount of modifications exceed some threshold, the data owner downloads original file and new data blocks from the cloud storage server, and check their integrity. If all data are intact, the data owner recovers the latest version of file by applying all "diff" on the original file, then chooses a *new* set of keys, and outsources the latest version of file to the cloud storage server by applying our $\mathcal{PDP}$ scheme.

We remark that, in the above Modification Step, the storage for the file always grows, even if the operation is `deletion`. So we require "Refresh Step" to reduce storage overhead after a amount of deletion operations. With proper setting on the frequency of the expensive "Refresh Step", the proposed dynamic $\mathcal{PDP}$ scheme can achieve a good amortized complexity.

The above approach is generic, and can also apply to existing works (e.g [5,42,47]). Furthermore, the above approach supports dynamic operation on various data units—bit, byte, character, and data block. In contrast, previous dynamic $\mathcal{PDP}$ schemes [25,46] only support dynamic operations on blocks, i.e. `insert/edit/delete` a data block.

[6] Meantime, change the range of the secret token d from $\mathbb{Z}_{\phi(N)}$ to $\mathbb{Z}^*_{\phi(N)}$, in order to recover ψ_2 from ψ_1 through Equation (9).

[7] In fact, the dynamic operation itself can be considered as the "diff".

[8] This is called "belta encoding".

4.2.2 Hardware Implementation. Our EPOS scheme requires pseudorandom function and large integer addition, multiplication and exponentiation. We notice that: (1) RSA encryption method also requires large integer operations. (2) It is a common practice to use hash function (e.g. SHA256) or AES encryption function to simulate pseudorandom function: that is $\mathsf{PRF}_k(x) := \mathtt{SHA256}(k\|x)$ or $\mathsf{PRF}_k(x) := \mathtt{AES}_k(x)$. Therefore, based on the existing hardware implementations of RSA method and SHA256 hash function (or AES method) in the literature, we could implement our EPOS scheme using hardware in the near future.

It is much easier to implement our scheme in hardware than the EPOR scheme [47].

5 Performance Analysis

The proposed scheme is efficient in storage, communication and computation. The storage overhead due to authentication tags is $1/m$ of the file size (after error erasure encoding). The proof size in a verification is 2λ bits. In the setup, the data encoding algorithm DEncode requires n number of pseudorandom function evaluations, modular additions/multiplications. In a verification session, the computation of the prover algorithm Prove is dominated by the exponentiation with large integer exponent in Equation (7), which is equivalent to m number of group exponentiation in $\mathbb{Z}_N^*$. The verifier algorithm Verify requires one modular division, three modular exponentiation in $\mathbb{Z}_N^*$, ℓ number of additions/multiplications in $\mathbb{Z}_{\phi(N)}$, and ℓ number of pseudorandom function evaluations, where $\ell = |C|$ is the number of indices selected during a verification.

5.1 Comparison

We compared the performance of the proposed scheme EPOS with existing schemes in Table 1 on page 9.

Furthermore, in table 2 on page 9, we give a concrete comparison with respect to the setting specified in the below example. We remark that both Ateniese *et al.* [5,4] and the proposed scheme supports only private key verification.

Example 1 *After erasure encoding, the file size is 1GB, block size is $m = 100$, and storage overhead due to authentication tags is about 10MB for all schemes. For all schemes, we assume that, during a verification, the challenge query $\{(i, \nu_i) : i \in C\}$ is represented by two 80 bits PRF seeds*[9]*. 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 takes roughly 5 millisecond in a standard PC.*

5.2 Experiment

We implement a prototype of our scheme in C language and using GMP [29] library version 5.02. We run the prototype in a Laptop PC with a 2.5GHz Intel Core 2 Duo mobile CPU (model T9300, released in 2008). We achieve a throughput of data preprocessing at speed 24 megabytes per second, which is about 400 times faster than Atenesis *et al.* [5,4], which achieves throughput of data preprocessing at speed 0.05 megabytes per second with a 3.0GHz desktop CPU [5]. This empirical result agrees with our previous analysis.

[9] Alternatively, one may use "hitter sampler" [30]

Table 1. Performance Comparison. All schemes support private verification only. In each scheme (except the first two in the table), a challenge set $C \subset [0, n-1]$ contains ℓ block indices and can be compactly represented with 280 bits due to results of [23,30]. In the table, "`exp`", "`mul`" and "`add`" represent exponentiation, multiplication and addition in the corresponding groups/fields; "`samp`" represents the sample method given by Goldreich [30]; notation $|\mathsf{F}|$ denotes the file size in bits. Here λ denotes the bit-length of group element size, s $(= m)$ denotes the block size, and ℓ denotes the number of data blocks accessed during on verification. *Note: In Ateniese* et al. *[5,4]'s $\mathcal{PDP}$ scheme, exponentiation with a large integer exponent 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	Group element size (bits)	Communication (bits)	Storage Overhead	Computation (Prover)	Computation (Verifier)	Computation (Data Preprocess)
RSA-based scheme [22]	$\lambda = 1024$	2λ	Zero	$\lvert\mathsf{F}\rvert/\lambda$ `exp`	1 `exp`	1 `exp`
PolyCommit [34]	$\lambda = 160$	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 [11]	$\lambda = 160$	$2\lambda + 440$	$\lvert\mathsf{F}\rvert$	$\lvert\mathsf{F}\rvert/\lambda$ (`exp` + `mul` + `add`)	2 `exp`	$\lvert\mathsf{F}\rvert/\lambda$ (`exp` + `mul` + `add`)
Ateniese [4][5]	$\lambda = 1024$	$2\lambda + 520$	$\lvert\mathsf{F}\rvert/s$	$(\ell + s)$ `exp` + 2ℓ `mult.` + ℓ `add` + 1 `hash` + 1 `samp`	ℓ (`exp` + `mult.`) + 1 `hash` + 1 `samp`	$\lvert\mathsf{F}\rvert/\lambda$ `exp` + n `hash`
S.W. [42]	$\lambda = 80$	$(s+1)\lambda + 360$	$\lvert\mathsf{F}\rvert/s$	$s\ell$ (`add` + `mult`) + 1 `samp`	$(\ell + s)$ (`add` + `mult`) + ℓ `PRF` + 1 `samp`	$\lvert\mathsf{F}\rvert/\lambda$ (`mul` + `add`) + $\lvert\mathsf{F}\rvert/(\lambda s)$ `PRF`
EPOR(E.C.) [47]	$\lambda = 160$	$3\lambda + 440$	$\lvert\mathsf{F}\rvert/s$	$(s-1)$ `exp` + $(s\ell + s + \ell)$ (`add` + `mul`) + 1 `samp`	2 `exp` + ℓ (`add` + `mult`) + ℓ `PRF` + 1 `samp`	$\lvert\mathsf{F}\rvert/\lambda$ (`mul` + `add`) + $\lvert\mathsf{F}\rvert/(\lambda s)$ `PRF`
EPOR($\mathbb{Z}_q^*$) [47]	$\lambda = 1024$	$3\lambda + 440$	$\lvert\mathsf{F}\rvert/s$	$(s-1)$ `exp` + $(s\ell + s + \ell)$ (`add` + `mul`) + 1 `samp`	2 `exp` + ℓ (`add` + `mult`) + ℓ `PRF` + 1 `samp`	$\lvert\mathsf{F}\rvert/\lambda$ (`mul` + `add`) + $\lvert\mathsf{F}\rvert/(\lambda s)$ `PRF`
EPOS (This work)	$\lambda = 1024$	$2\lambda + 416$	$\lvert\mathsf{F}\rvert/s$	$(s+1)$ `exp` + $(s\ell + s + \ell)$ (`add` + `mul`) + 1 `samp`	2 `exp` + ℓ (`add` + `mult`) + ℓ `PRF` + 1 `samp`	$\lvert\mathsf{F}\rvert/\lambda$ (`mul` + `add`) + $\lvert\mathsf{F}\rvert/(\lambda s)$ `PRF`

Table 2. Comparison with an example among the $\mathcal{PDP}$ scheme by Ateniese *et al.* [5], the $\mathcal{POR}$ scheme by Shacham and Waters [42], the $\mathcal{POR}$ scheme named EPOR proposed by Xu and Chang [47], and $\mathcal{PDP}$ scheme EPOS 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 (part of) challenge $\{(i, \nu_i) : i \in C\}$ are represented compactly with 280 bits due to results of [23,30]. 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: (1) In Ateniese* et al. *$\mathcal{PDP}$ scheme, exponentiation with a large integer exponent 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. (2) One 1024 bits modular exponentiation or one 160 bits elliptic curve exponentiation takes roughly 5 millisecond in a standard PC.*

Scheme	**Group element size (bits)**	**Communication bits**	**Computation (Data Preprocess)**	**Computation (Prove)**
Ateniese [4][5]	$\lambda = 1024$	$2\lambda + 520 = 2568$	2^{23} `exp` over $\mathbb{Z}_N^*$	$(100 + \ell)$ `exp` over $\mathbb{Z}_N^*$
Shacham and Water [42]	$\lambda = 80$	$(s+1)\lambda + 360 = 8440$	2^{27} `mul` over $\mathbb{Z}_p$	100ℓ `mul` over $\mathbb{Z}_p$
EPOR(E.C.) [47]	$\lambda = 160$	$3\lambda + 440 = 920$	2^{26} `mul` over $\mathbb{Z}_p$	100 `exp` over Elliptic Curve
EPOR($\mathbb{Z}_q^*$) [47]	$\lambda = 1024$	$3\lambda + 440 = 3512$	2^{23} `mul` over $\mathbb{Z}_p$	100 `exp` over $\mathbb{Z}_q^*$
EPOS (This work)	$\lambda = 1024$	$2\lambda + 160 + 256 = 2464$	2^{23} `mul`. over $\mathbb{Z}_N^*$	102 `exp`. over $\mathbb{Z}_N^*$

6 Security Analysis of Scheme EPOS

6.1 Security Formulation

We review the Provable Data Possession formulation proposed by Ateniese *et al.* [5,4]. The $\mathcal{PDP}$ security game between a *probabilistic polynomial time* (PPT) adversary $\mathcal{A}$ and a PPT challenger $\mathcal{C}$ w.r.t. a $\mathcal{PDP}$ scheme $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{DEncode}, \mathsf{Challenge}, \mathsf{Prove}, \mathsf{Verify})$ is as below.

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

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

- Store-query ($\mathtt{F}$): Given a data file $\mathtt{F}$ chosen by $\mathcal{A}$, the challenger $\mathcal{C}$ responses by running data encoding algorithm $(\mathsf{id}, \hat{\mathtt{F}}) \leftarrow \mathsf{DEncode}(sk, \mathtt{F})$ and sending the encoded data file $\hat{\mathtt{F}}$ 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 $\mathcal{PDP}$ verification with $\mathcal{A}$ w.r.t. the data file $\mathtt{F}$ associated to the identifier id in this way:
  - $\mathcal{C}$ runs the algorithm $\mathsf{Challenge}$ to generate a pair of public-secret tokens $(\mathsf{pt}, \mathsf{st})$ and the a challenge query Chall, and sends $(\mathsf{pt}, \mathsf{Chall})$ to the adversary $\mathcal{A}$ and keeps st safely. The secret token st will be used in the verifier Verify algorithm.
  - $\mathcal{A}$ produces a proof ψ w.r.t. the challenge Chall;
    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 $\mathsf{Verify}(sk, \mathsf{id}, \mathsf{Chall}, \psi)$. Denote the output as b.

  $\mathcal{C}$ sends the decision bit $b \in \{\mathtt{accept}, \mathtt{reject}\}$ 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 $\mathtt{F}^*$ denote the data file associated to identifier id^*.

Retrieve: The challenger $\mathcal{C}$ initiates one $\mathcal{PDP}$ verifications with $\mathcal{A}$ w.r.t. the data file $\mathtt{F}^*$, where $\mathcal{C}$ plays the role of verifier and $\mathcal{A}$ plays the role of prover, as in the **Learning** phase. Suppose the challenger $\mathcal{C}$ asks $\mathcal{A}$ to check all data blocks $\mathtt{F}_i$ of $\mathtt{F}^*$ with index $i \in \mathbf{C}$. The challenger $\mathcal{C}$ extracts file blocks $\{\mathtt{F}'_i : i \in \mathbf{C}\}$ from $\mathcal{A}$'s storage by applying a PPT knowledge extractor. The adversary $\mathcal{A}$ wins this $\mathcal{PDP}$ security game, if the challenger $\mathcal{C}$ accepts $\mathcal{A}$'s response in the verification. The challenger $\mathcal{C}$ wins this game, if the extracted blocks $\{(i, \mathtt{F}'_i) : i \in \mathbf{C}\}$ are identical to the original $\{(i, \mathtt{F}_i) : i \in \mathbf{C}\}$.

Definition 1 ([5]) *A $\mathcal{PDP}$ scheme is sound if for any PPT adversary $\mathcal{A}$, the probability that $\mathcal{A}$ wins the above $\mathcal{PDP}$ security game is negligibly close to the probability that $\mathcal{C}$ wins the same security game. That is*

$$\mathsf{Pr}[\mathcal{A} \textit{ wins } \mathcal{PDP} \textit{ game }] \leq \mathsf{Pr}[\mathcal{C} \textit{ wins } \mathcal{PDP} \textit{ game }] + \textit{negl}. \tag{10}$$

6.2 KEA Assumption

The Knowledge of Exponent Assumption (KEA) is introduced by Damgård [20] and subsequently appears in many works [32,9,10,35,21]. The below is a variant version of KEA in the RSA ring given by Ateniese *et al.* [5].

Assumption 1 (Knowledge of Exponent Assumption [20,9]) *Let $N = pq$ be a RSA modulus, $g \in \mathbb{Z}_N^*$ and s be an positive integer. For any PPT algorithm $\mathcal{A}$ that takes (N, g, g^s) as input and r as random coin and returns (C, Y) such that $Y = C^s \mod N$, there exists a PPT* extractor *algorithm $\bar{\mathcal{A}}$ which, given (N, g, g^s, r) as input, outputs x such that $g^x = C \mod N$.*

Remark 2

- *Note that the extractor $\bar{\mathcal{A}}$ has access to $\mathcal{A}$'s input (N, g, g^s) and $\mathcal{A}$'s random coin r, thus $\bar{\mathcal{A}}$ can replay step by step the process how $\mathcal{A}$ computes (C, Y) from (N, g, g^s).*
- *This assumption has been shown to hold in generic group by Abe and Fehr [1].*

Assumption 2 (Factorization Assumption [40]) *We say an integer N is a RSA modulus, if $N = pq$ and all of $p, q, \frac{p-1}{2}, \frac{q-1}{2}$ are prime numbers and bit-lengths of p and q are equal. Then for any PPT adversary $\mathcal{A}$, the probability that $\mathcal{A}$ can factorize a randomly chosen λ bits RSA modulus, is negligible in λ.*

6.3 Security Proof

Theorem 1 (EPOS is Sound) *If the Knowledge of Exponent Assumption 1 holds and the pseudorandom function* PRF *is secure, then the proposed scheme* EPOS *is sound.*

The proof below is similar to the proof of Ateniese *et al.* [5] *in the high level*: If an adversary $\mathcal{A}$ wins the security game, then a knowledge extractor (as in the assumption KEA) can find a linear combination of data blocks (See Equation 5). Then each individual block can be obtained by solving a linear equation system.

In details, our proof are very different from Ateniese *et al.* [5]. For ease of exposition, we clarify two related but distinct concepts: *valid proof* and *genuine proof*. A proof is valid, if it is accepted by the verifier. A proof is genuine, if it is the same as the one generated by an honest (deterministic) prover on the same query. In our scheme, for each query, there is only one genuine proof, and there are many valid proofs.

At first in Lemma 2, we prove Theorem 1 in a simplified **no-feedback setting**, where all decisions (acceptance or rejection) are kept secret from the adversary in the $\mathcal{PDP}$ security game. Then we will lift the security to the **feedback setting**, where all decisions (acceptance or rejection) are given to the adversary in the $\mathcal{PDP}$ security game—this is the exactly the original $\mathcal{PDP}$ securty game setting in Sec 6.1.

Lemma 2 *Suppose the Factorization Assumption 2 holds and the pseudorandom function* PRF *is secure. Then the proof (ψ_1, ψ_2) in the proposed scheme* EPOS *is unforgeable in the no-feedback setting, where all acceptance or rejection decisions are kept secure from the adversary in the $\mathcal{PDP}$ security game. More precisely, let $(\hat{\psi}_1, \hat{\psi}_2)$ denote the adversary $\mathcal{A}$'s response in the **Retrieve** phase of the $\mathcal{PDP}$ security game w.r.t.* EPOS*. The probability*

$$\begin{aligned} &\mathsf{Pr}[\textit{Verifier accepts } (\hat{\psi}_1, \hat{\psi}_2) \quad \wedge \quad (\hat{\psi}_1, \hat{\psi}_2) \neq (\psi_1, \psi_2)] \\ &\leq negl(\lambda) \end{aligned} \tag{11}$$

is negligible.

The proof of the above Lemma 2 is in Appendix A.

Lemma 3 *Suppose the Factorization Assumption 2 holds and the pseudorandom function* PRF *is secure. Then the proof* (ψ_1, ψ_2) *in the proposed scheme* EPOS *is unforgeable in the feedback setting, where all acceptance or rejection decisions are provided to the adversary in the* $\mathcal{PDP}$ *security game. (The full detailed proof of this lemma is in Appendix B.)*

Proof ((Proof Sketch of Lemma 3)). In the simulated security game, the challenger does not have all information of private key and thus cannot answer verification queries.

However, the challenger can construct a simulated verifier: The challenger can consistently keep a local copy of all files and tags and computes the genuine proof by himself/herself in each verification session. The challenger accepts a proof provided by the adversary, if and only if the received proof is identical to the genuine proof. Thus, this simulated verifier will accept only genuine proof and will reject valid but no genuine proof. Therefore, the simulated verifier and the real verifier will output different decisions, if and only if the adversary can output valid but not genuine proof.

Next, we can show that, the probability that the adversary can output valid but not genuine proof is negligible, using mathematical induction proof. The induction is taken on the number (denoted as k) of verification queries made in the learning phase in the $\mathcal{PDP}$ security game. We notice that, the standard $\mathcal{PDP}$ security game in the case $k = 0$, where adversary makes no verification-query in the learning phase, is equivalent to the modified $\mathcal{PDP}$ security game in the no-feedback-setting, where adversary can make many verification queries in the learning phase but the adversary did not get any feedback on these queries—Asking many queries without feedback is equivalent to asking no query. Therefore Lemma 2 just proves the basic case with $k = 0$. We will prove the inductive step from k to $k + 1$ using conditional probability analysis. The full detailed proof of this lemma is in Appendix B. □

Now it is the time to prove the main Theorem 1 in this paper.

Proof (of Theorem 1). Lemma 3 states that the proof in the scheme EPOS is unforgeable. Since for random value $d \in \mathbb{Z}_{\phi(N)}$, $\mathcal{A}$ can win $\mathcal{PDP}$ security game with non-negligible probability. Then for many values d_i's, $\mathcal{A}$ can compute $(\psi_{i,1} = g_{d_i}^{\pi_1}, \psi_{i,2} = g^{\pi_2})$ correctly. Let us just consider d_1 and d_2 among these d_i's. Let $c = \frac{d_2}{d_1} \mod \phi(N)$. Given input $(g^{d_1}, g^{d_2} = \left(g^{d_1}\right)^c)$, the adversary $\mathcal{A}$ can output $(g^{d_1\pi_1}, g^{d_2\pi_1} = \left(g^{d_1\pi_1}\right)^c)$. By Knowledge of Exponent Assumption (KEA [20]), there exists an extractor that can find M, such that $g^{d_1\pi_1} = g^{d_1 M} \mod N$.

Case 1: $M \neq \pi_1$ If the two integers M and π_1 are distinct (*Caution: Here we treat* M, π_1 *as large integer instead of group elements from* $\mathbb{Z}_{\phi(N)}$), then the difference $M - \pi_1$ has to be a multiple of $\frac{1}{4}\phi(N)$, from which the factorization of N can be found using Miller's result [37].

Case 2: $M = \pi_1$ In this case, the extractor finds π_1, as desired. Recall that the large integer $\pi_1 = \sum_{i \in C} \nu_i \mathtt{F}_i$ (Yes, integer, not group element) is linear equation of file blocks $\mathtt{F}_i$'s. Similar to the proof in Ateniese's PDP [5], by choose independent weights ν_i's in $|C|$ number of executions of the protocol, we obtain $|C|$ independent linear equations in the unknowns $\mathtt{F}_i, i \in C$. Thus these file blocks $\mathtt{F}_i, i \in C$, can be found by solving the linear equation system over integer domain.

Thus, Theorem 1 is proved.

7 Conclusion

In this paper, we proposed a $\mathcal{PDP}$ scheme EPOS which is very efficient in communication, storage and computation. Compared to Ateniese *et al.* [5], EPOS is much more efficient in computation (400 times faster in setup), and equally efficient in communication and storage.

References

1. Masayuki Abe and Serge Fehr. Perfect NIZK with adaptive soundness. In *TCC '07: Theory of Cryptography Conference*, pages 118–136, 2007.
2. Shweta Agrawal and Dan Boneh. Homomorphic MACs: MAC-Based Integrity for Network Coding. In *ACNS '09: International Conference on Applied Cryptography and Network Security*, pages 292–305, 2009.
3. Stephanos Androutsellis-Theotokis and Diomidis Spinellis. A survey of peer-to-peer content distribution technologies. *ACM Computing Surveys*, 36(4):335–371, 2004.
4. Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Osama Khan, Lea Kissner, Zachary Peterson, and Dawn Song. Remote data checking using provable data possession. *ACM Transactions on Information and System Security*, 14:12:1–12:34, 2011.
5. Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary Peterson, and Dawn Song. Provable data possession at untrusted stores. In *CCS '07: ACM conference on Computer and communications security*, pages 598–609, 2007.
6. Giuseppe Ateniese, Roberto Di Pietro, Luigi Mancini, and Gene Tsudik. Scalable and efficient provable data possession. In *SecureComm '08: International conference on Security and privacy in communication netowrks*, pages 9:1–9:10, 2008.
7. Giuseppe Ateniese, Seny Kamara, and Jonathan Katz. 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*, pages 319–333, 2009.
8. Christopher Batten, Kenneth Barr, Arvind Saraf, and Stanley Trepetin. pStore: A Secure Peer-to-Peer Backup System. Technical Report MIT-LCS-TM-632, MIT, 2002.
9. Mihir Bellare and Adriana Palacio. The Knowledge-of-Exponent Assumptions and 3-Round Zero-Knowledge Protocols. In *CRYPTO '04: Annual International Cryptology Conference on Advances in Cryptology*, pages 273–289, 2004.
10. Mihir Bellare and Adriana Palacio. Towards Plaintext-Aware Public-Key Encryption Without Random Oracles. In *ASIACRYPT '04: International Conference on the Theory and Application of Cryptology and Information Security*, pages 48–62, 2004.
11. Siavosh Benabbas, Rosario Gennaro, and Yevgeniy Vahlis. Verifiable Delegation of Computation over Large Datasets. In *CRYPTO '11: Annual International Cryptology Conference on Advances in Cryptology*, pages 111–131, 2011.
12. Manuel Blum, Will Evans, Peter Gemmell, Sampath Kannan, and Moni Naor. Checking the correctness of memories. In *FOCS '91: Annual Symposium on Foundations of Computer Science*, pages 90–99, 1991.
13. Dan Boneh, David Freeman, Jonathan Katz, and Brent Waters. Signing a Linear Subspace: Signature Schemes for Network Coding. In *PKC '09: International Conference on Practice and Theory in Public Key Cryptography*, pages 68–87, 2009.
14. Kevin Bowers, Ari Juels, and Alina Oprea. HAIL: a high-availability and integrity layer for cloud storage. In *CCS '09: ACM conference on Computer and communications security*, pages 187–198, 2009.
15. Kevin Bowers, Ari Juels, and Alina Oprea. Proofs of retrievability: theory and implementation. In *CCSW '09: ACM workshop on Cloud computing security*, pages 43–54, 2009.
16. David Cash, Alptekin Kupcu, and Daniel Wichs. Dynamic Proofs of Retrievability via Oblivious RAM. volume 7881 of *EUROCRYPT '13: Advances in Cryptology*, pages 279–295. 2013. `http://eprint.iacr.org/2012/550`.
17. Ee-Chien Chang and Jia Xu. Remote Integrity Check with Dishonest Storage Server. In *ESORICS '08: European Symposium on Research in Computer Security*, pages 223–237, 2008.
18. Kai-Min Chung, Yael Kalai, and Salil Vadhan. Improved Delegation of Computation Using Fully Homomorphic Encryption. In *CRYPTO '10: Annual International Cryptology Conference on Advances in Cryptology*, pages 483–501, 2010.
19. Reza Curtmola, Osama Khan, Randal Burns, and Giuseppe Ateniese. MR-PDP: Multiple-Replica Provable Data Possession. In *ICDCS '08: International Conference on Distributed Computing Systems*, pages 411–420, 2008.

20. Ivan Damgård. Towards Practical Public Key Systems Secure Against Chosen Ciphertext Attacks. In *CRYPTO '91: Annual International Cryptology Conference on Advances in Cryptology*, pages 445–456, 1992.
21. Alexander W. Dent. The Cramer-Shoup Encryption Scheme Is Plaintext Aware in the Standard Model. In *EUROCRYPT '06: Annual International Conference on Advances in Cryptology*, pages 289–307, 2006.
22. Yves Deswarte, Jean-Jacques Quisquater, and Ayda Saïdane. Remote Integrity Checking: How to Trust Files Stored on Untrusted Servers . In *Conference on Integrity and Internal Control in Information Systems*, pages 1–11, 2003.
23. Yevgeniy Dodis, Salil Vadhan, and Daniel Wichs. Proofs of Retrievability via Hardness Amplification. In *TCC '09: Theory of Cryptography Conference on Theory of Cryptography*, pages 109–127, 2009.
24. Cynthia Dwork, Moni Naor, Guy Rothblum, and Vinod Vaikuntanathan. How Efficient Can Memory Checking Be?. In *TCC '09: Theory of Cryptography Conference*, pages 503–520, 2009.
25. Chris Erway, Alptekin Küpçü, Charalampos Papamanthou, and Roberto Tamassia. Dynamic provable data possession. In *CCS '09: ACM conference on Computer and communications security*, pages 213–222, 2009.
26. Décio Filho and Paulo Barreto. Demonstrating data possession and uncheatable data transfer. Cryptology ePrint Archive, Report 2006/150, 2006. `http://eprint.iacr.org/`.
27. Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive Verifiable Computing: Outsourcing Computation to Untrusted Workers. In *CRYPTO '10: Annual International Cryptology Conference on Advances in Cryptology*, pages 465–482, 2010.
28. Craig Gentry. Fully Homomorphic Encryption using Ideal Lattices. In *STOC '09: ACM Symposium on Theory of Computing*, pages 169–178, 2009.
29. GMP. The GNU Multiple Precision Arithmetic Library. `http://www.gmplib.org/`.
30. Oded Goldreich. A Sample of Samplers - A Computational Perspective on Sampling (survey). *Electronic Colloquium on Computational Complexity (ECCC)*, 4(20), 1997.
31. Oded Goldreich. *Foundations of Cryptography: Volume 1, Basic Tools*. Cambridge University Press, New York, NY, USA, 2006.
32. Satoshi Hada and Toshiaki Tanaka. On the Existence of 3-Round Zero-Knowledge Protocols. In *CRYPTO '98: Annual International Cryptology Conference on Advances in Cryptology*, pages 408–423, 1998.
33. Ari Juels and Burton Kaliski, Jr. Pors: proofs of retrievability for large files. In *CCS '07: ACM conference on Computer and communications security*, pages 584–597, 2007.
34. Aniket Kate, Gregory Zaverucha, and Ian Goldberg. Constant-Size Commitments to Polynomials and Their Applications. In *ASIACRYPT '10: International Conference on the Theory and Application of Cryptology and Information Security*, pages 177–194, 2010.
35. Hugo Krawczyk. HMQV: A High-Performance Secure Diffie-Hellman Protocol. In *CRYPTO '05: Annual International Cryptology Conference on Advances in Cryptology*, pages 546–566, 2005.
36. Jinyang Li and Frank Dabek. F2F: Reliable Storage in Open Networks. In *IPTPS '06: International Workshop on Peer-to-Peer Systems*, 2006.
37. Gary Miller. Riemann's hypothesis and tests for primality. In *STOC '75: ACM Symposium on Theory of Computing*, pages 234–239, 1975.
38. Moni Naor and Guy Rothblum. The Complexity of Online Memory Checking. In *FOCS '05: Symposium on Foundations of Computer Science*, pages 573–584, 2005.
39. NIST. National Institute of Standards and Technology. Secure hash standard (SHS). FIPS 180-2, August 2002.
40. Ronald Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.
41. Amit Sahai and Salil Vadhan. A Complete Problem for Statistical Zero Knowledge. *Journal of the ACM*, 50:196–249, 2003.
42. Hovav Shacham and Brent Waters. Compact Proofs of Retrievability. In *ASIACRYPT '08: International Conference on the Theory and Application of Cryptology and Information Security*, pages 90–107, 2008.
43. Elaine Shi, Emil Stefanov, and Charalampos Papamanthou. Practical Dynamic Proofs of Retrievability. (will appear in) CCS '13: ACM Conference on Computer and Communications Security.
44. Edward Suh, Dwaine Clarke, Blaise Gassend, Marten van Dijk, and Srinivas Devadas. Efficient Memory Integrity Verification and Encryption for Secure Processors. In *MICRO '03: Annual IEEE/ACM International Symposium on Microarchitecture*, pages 339–350, 2003.
45. Cong Wang, Qian Wang, Kui Ren, and Wenjing Lou. Privacy-Preserving Public Auditing for Data Storage Security in Cloud Computing. In *INFOCOM '10: Annual IEEE International Conference on Computer Communications*, pages 525–533, 2010.
46. Qian Wang, Cong Wang, Jin Li, Kui Ren, and Wenjing Lou. Enabling public verifiability and data dynamics for storage security in cloud computing. In *ESORICS'09: European conference on Research in computer security*, pages 355–370, 2009.

47. Jia Xu and Ee-Chien Chang. Towards Efficient Proofs of Retrievability. In *(Full Paper) AsiaCCS '12: ACM Symposium on Information, Computer and Communications Security*, 2012. `http://eprint.iacr.org/2011/362`.
48. Kan Yang and Xiaohua Jia. Data storage auditing service in cloud computing: challenges, methods and opportunities. *Journal of World Wide Web*, 2011.

A Proof of Lemma 2

Proof (of Lemma 2).
Game 1 The first game is the same as the $\mathcal{PDP}$ security game described in Section 6.1, except that

- All acceptance or rejection decisions are kept secure from the adversary $\mathcal{A}$. Essentially, the challenger in the $\mathcal{PDP}$ security game does not answer verification queries made by the adversary.
- Adversary $\mathcal{A}$ wins in **Game 1**, if $\mathcal{A}$'s forgery proof $(\hat{\psi}_1, \hat{\psi}_2)$ is accepted and it is different from the genuine proof. Formally, let id, $\{(i, \nu_i) : i \in C\}$ and $(\mathsf{pt}, \mathsf{st})$ denote the file identifier, challenge query, and public-secret tokens respectively in the **Retrieve** phase of the $\mathcal{PDP}$ security game, let (ψ_1, ψ_2) denote the corresponding genuine proof and (pk, sk) be the public-private key pair. Adversary $\mathcal{A}$ wins in **Game 1**, if

$$\mathsf{Verify}(sk, \mathsf{id}, \mathsf{st}, \{(i, \nu_i) : i \in C\}, \hat{\psi}_1, \hat{\psi}_2) = \mathtt{accept} \\ \text{and } (\hat{\psi}_1, \hat{\psi}_2) \neq (\psi_1, \psi_2). \tag{12}$$

Game 2 The second game is the same as **Game 1**, except that the pseudorandom function PRF outputs true randomness. Precisely, the function $\mathsf{PRF}_{\mathsf{seed}}$ is evaluated in the following way:

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

Game 3 The third game is the same as **Game 2**, except that:

- The range of the function PRF is changed from $\mathbb{Z}_{\phi(N)}$ to $\mathbb{Z}_N$. Note that in this game, PRF is evaluated in the same way as in **Game 2**;
- The range of the authentication tag is also changed from $\mathbb{Z}_{\phi(N)}$ to $\mathbb{Z}_N$. More precisely, the Equation (4) (on page 6) is replaced by the following equations

$$\sigma_i := \tau_{\mathtt{F}_i} + \mathsf{PRF}_{\mathsf{seed}}(\mathsf{id} \| i) \mod N. \tag{13}$$

We remark that in **Game 3**, the challenger is not able to verify adversary's response, and the challenger does not need to do verification either, since in the no-feedback setting, the challenger will not answer verification queries made by the adversary.

Claim 1 *If there is a non-negligible difference in a PPT adversary $\mathcal{A}$'s success probability between* **Game 1** *and* **Game 2**, *then there exists another PPT adversary $\mathcal{B}$ that can break the security of the pseudorandom function* PRF. *More precisely,*

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

where N_{PRF} is the number of distinct evaluations of pseudorandom function PRF *required and $\mathsf{Adv}_{\mathcal{B}}^{\mathsf{PRF}}$ denotes the probability that $\mathcal{B}$ can distinguish the output of* PRF *from true randomness.*

The above Claim 1 can be proved using a standard hybrid argument [31]. Here we save the details.

Claim 2 *For any computationally unbounded adversary $\mathcal{A}$, the probability that $\mathcal{A}$ can find the secret value τ after interacting in* **Game 2**, *is $\frac{1}{\phi(N)}$.*

Proof ((Proof Sketch of Claim 2)). In **Game 2**, the function PRF outputs true random numbers in $\mathbb{Z}_{\phi(N)}$ and thus the secret value τ is hidden perfectly. Therefore, the probability that an (computationally unbounded) adversary $\mathcal{A}$ can find τ is $\frac{1}{\phi(N)}$. Recall that τ is chosen at random from group $\mathbb{Z}_{\phi(N)}$.

Claim 3 *For any PPT adversary $\mathcal{A}$, the probability that $\mathcal{A}$ can factorize N after interacting in* **Game 3** *is negligible.*

Proof ((Proof Sketch of Claim 3)). Recall that in **Game 3**, the authentication tag σ_i for each block is a group element chosen at random from $\mathbb{Z}_N$. Suppose a PPT adversary $\mathcal{A}$ factorizes the RSA modulus N after interacting in **Game 3**.

Based on $\mathcal{A}$, we construct a PPT adversary $\mathcal{B}$ to factorize N. Given only the RSA modulus N, the adversary $\mathcal{B}$ can play the role of challenger to setup[10] the $\mathcal{PDP}$ security game w.r.t. scheme EPOS, and answer store queries made by the adversary $\mathcal{A}$ by sampling uniform random number from $\mathbb{Z}_N$ as the authentication tag σ_i. Thus,

$$\begin{aligned} &\mathsf{Pr}[\mathcal{A} \text{ factorizes } N \text{ in } \textbf{Game 3}] \\ \leq &\mathsf{Pr}[\mathcal{B} \text{ factorizes } N] \\ = &\mathsf{Adv}_{\mathcal{B}}^{\text{FACT}}. \end{aligned} \tag{14}$$

Claim 4 *For any PPT adversary $\mathcal{A}$, the probability that $\mathcal{A}$ can factorize N after interacting in* **Game 2** *is negligible.*

Proof (of Claim 4). We will show that any PPT adversary cannot distinguish between **Game 2** and **Game 3**. As a result, Claim 3 can imply Claim 4.

Now we study the *statistical difference* [41] between uniform random variables over $\mathbb{Z}_{\phi(N)}$ and over $\mathbb{Z}_N$.

[10] From the input N, $\mathcal{B}$ can generate the public key and simulate the algorithm $\mathsf{DEncode}$. In the no-feedback setting, $\mathcal{B}$ does not need to do verification, so secret key is not be necessary.

Let X be a uniform random variable over $\mathbb{Z}_{\phi(N)}$ and Y be a uniform random variable over $\mathbb{Z}_N$. The statistical difference [41] between X and Y is

$$\begin{aligned}
\mathsf{SD}(X,Y) &\stackrel{\text{def}}{=} \frac{1}{2}\sum_{a} |\mathsf{Pr}[X=a] - \mathsf{Pr}[Y=a]| \\
&= \frac{1}{2}\sum_{a\in\mathbb{Z}_{\phi(N)}} |\mathsf{Pr}[X=a] - \mathsf{Pr}[Y=a]| \ + \\
&\quad \frac{1}{2}\sum_{a\in\mathbb{Z}_N\setminus\mathbb{Z}_{\phi(N)}} |\mathsf{Pr}[X=a] - \mathsf{Pr}[Y=a]| \\
&= \frac{1}{2}\left(\frac{1}{\phi(N)} - \frac{1}{N}\right) \times \phi(N) \ + \\
&\quad \frac{1}{2}\left(\frac{1}{N} - 0\right) \times (N - \phi(N)) \\
&= 1 - \frac{\phi(N)}{N} \\
&= 1 - (1-\frac{1}{p})(1-\frac{1}{q}) \\
&= \frac{1}{p} + \frac{1}{q} - \frac{1}{pq}.
\end{aligned}$$

Let N_0 be a positive integer. Let $X_i, i = 1, 2, \ldots, N_0$, be independently and identically distributed uniform random variables over $\mathbb{Z}_{\phi(N)}$, and $Y_i, i = 1, 2, \ldots, N_0$, be independently and identically distributed uniform random variables over $\mathbb{Z}_N$. According to Fact 2.1 and Fact 2.3 of Sahai and Vadhan [41], we have

$$\begin{aligned}
&\mathsf{SD}((X_1,\ldots,X_{N_0}),\ (Y_1,\ldots,Y_{N_0})) \\
&\qquad \leq \sum_{i\in[1,N_0]} \mathsf{SD}(X_i,\ Y_i). \qquad (15)
\end{aligned}$$

The right hand side of the above Equation (15) is

$$\begin{aligned}
\sum_{i\in[1,N_0]} \mathsf{SD}(X_i,\ Y_i) &= N_0 \times \mathsf{SD}(X,\ Y) \\
&= N_0 \left(\frac{1}{p} + \frac{1}{q} - \frac{1}{pq}\right). \qquad (16)
\end{aligned}$$

Suppose the adversary $\mathcal{A}$ obtains exactly N_{PRF} authentication tags σ_i (σ'_i respectively) for N_{PRF} different indices i's in **Game 2** (**Game 3** respectively). Since σ_i's are independently and identically distributed uniform random variables over $\mathbb{Z}_{\phi(N)}$ and σ'_is are independently and identically distributed uniform random variables over $\mathbb{Z}_N$, the difference of the adversary's views[11] in **Game 2** and **Game 3** is bounded as below

$$\mathsf{SD}(\textsc{View}_{\mathcal{A}}^{\textbf{Game 2}},\ \textsc{View}_{\mathcal{A}}^{\textbf{Game 3}}) \leq N_{\mathsf{PRF}} \left(\frac{1}{p} + \frac{1}{q} - \frac{1}{pq}\right). \qquad (17)$$

[11] Adversary's view is a transcript of all messages received.

The adversary $\mathcal{A}$ is polynomially bounded, which implies N_{PRF} is polynomially bounded. Therefore, the statistical difference $\mathsf{SD}(\text{VIEW}_{\mathcal{A}}^{\textbf{Game 2}}, \text{VIEW}_{\mathcal{A}}^{\textbf{Game 3}})$ is negligible in $\lambda \approx \log N \approx 2\log p \approx 2\log q$, and there is no adversary can distinguish between **Game 2** and **Game 3**.

Combining with Claim 3, we conclude that the probability

$$\begin{aligned}&\mathsf{Pr}[\mathcal{A} \text{ factorizes } N \text{ in } \textbf{Game 2}]\\ &\leq \mathsf{Pr}[\mathcal{A} \text{ factorizes } N \text{ in } \textbf{Game 3}] + N_{\mathsf{PRF}}\left(\frac{1}{p}+\frac{1}{q}-\frac{1}{pq}\right)\end{aligned}$$

is negligible in $\lambda \approx \log N$. The proof for Claim 4 is complete.

Claim 5 *Let $(\hat{\psi}_1, \hat{\psi}_2)$ denote the adversary $\mathcal{A}$'s output in the* **Game 2** *and (ψ_1, ψ_2) be the corresponding genuine output which shares the same values $\{(i, \nu_i) : i \in C\}$ with the forgery output. Then,*

$$\mathsf{Pr}[\mathcal{A} \textit{ wins } \textbf{Game 2} \wedge (\hat{\psi}_1, \hat{\psi}_2) \neq (\psi_1, \psi_2)] \leq \frac{1}{p'}. \tag{18}$$

$$\mathsf{Pr}[\mathcal{A} \textit{ wins } \textbf{Game 2}] \leq \frac{1}{p'}. \tag{19}$$

Proof (of Cliam 5). Suppose the adversary $\mathcal{A}$ wins **Game 2**, then $\mathcal{A}$'s forged proof $(\hat{\psi}_1, \hat{\psi}_2)$ is accepted and is different from the genuine output (ψ_1, ψ_2): $(\hat{\psi}_1, \hat{\psi}_2) \neq (\psi_1, \psi_2)$. Since both the forged proof and genuine proof are accepted by the verifier w.r.t. $\{(i, \nu_i) : i \in C\}$ and satisfy the Equation (9) (on page 7), we have

$$\left(\hat{\psi}_1\right)^{\tau} = \left(\frac{\hat{\psi}_2}{g^{\sum_{i\in C}\nu_i \mathsf{PRF}_{\mathsf{seed}}(\mathsf{id}\|i)}}\right)^{d} \mod N \tag{20}$$

$$(\psi_1)^{\tau} = \left(\frac{\psi_2}{g^{\sum_{i\in C}\nu_i \mathsf{PRF}_{\mathsf{seed}}(\mathsf{id}\|i)}}\right)^{d} \mod N \tag{21}$$

Dividing Equation (20) with Equation (21), we have

$$\left(\frac{\hat{\psi}_1}{\psi_1}\right)^{\tau} = \left(\frac{\hat{\psi}_2}{\psi_2}\right)^{d} \mod N \tag{22}$$

Recall that the verifier algorithm Verify accepts only if $\psi_1, \hat{\psi}_1 \in \mathcal{QR}_N$. Thus $\frac{\hat{\psi}_1}{\psi_1} \in \mathcal{QR}_N$ is also a quadratic residue. For any element $x \in \mathcal{QR}_N$, $x^{\frac{1}{4}\phi(N)} = 1$, and the multiplicative order of x modulo N will be a factor of $\frac{1}{4}\phi(N) = p'q'$. Since $\frac{\hat{\psi}_1}{\psi_1} \neq 1$, the multiplicative order, denoted with φ, of $\frac{\hat{\psi}_1}{\psi_1}$ modulo N is at least $\min\{p', q'\} = p'$. Thus a computationally unbounded adversary $\mathcal{B}$ can invoke the adversary $\mathcal{A}$ to obtain the above Equation (22) and find the value $(\tau \mod \varphi)$ from Equation (22) by solving a discrete log problem with $\frac{\hat{\psi}_1}{\psi_1}$ as base. The probability that $(\tau \mod \varphi) = \tau$ is

$$\mathsf{Pr}[(\tau \mod \varphi) = \tau] = \mathsf{Pr}[\tau \in \mathbb{Z}_{\varphi}] = \frac{\varphi}{\phi(N)}. \tag{23}$$

By Claim 2, we have the probability

$$\begin{aligned}
&\mathsf{Pr}[\mathcal{A} \text{ wins } \textbf{Game 2} \wedge (\hat{\psi}_1, \hat{\psi}_2) \neq (\psi_1, \psi_2)] \\
\leq\ &\frac{\mathsf{Pr}[\mathcal{B} \text{ finds } \tau \text{ in } \textbf{Game 2}]}{\mathsf{Pr}[(\tau \mod \varphi) = \tau]} \\
=\ &\frac{\frac{1}{\phi(N)}}{\frac{\varphi}{\phi(N)}} \\
=\ &\frac{1}{\varphi} \leq \frac{1}{p'}
\end{aligned}$$

is negligible in $\lambda \approx \log N \approx 2 + 2\log p'$. The proof of Claim 5 is complete.

Thus, Lemma 2 is proved.

B Proof of Lemma 3

Proof (of Lemma 3).
Game 1. The first game is just a simplified version of the $\mathcal{PDP}$ security game as described in Section 6.1 (on page 10), such that all acceptance or rejection decisions are kept secret from the adversary. Recall that we refer to such simplified version as *$\mathcal{PDP}$ security game in the no-feedback setting*. This **Game 1** is identical to the **Game 1** in the proof of Lemma 2 (on page 11).

For each integer $k \geq 0$, we define the following game:
Game 2.k. This game is the same as **Game 1**, except that, $\mathcal{A}$ adaptively makes k verification-queries in the **Learning** phase and all acceptance or rejection decisions are provided to $\mathcal{A}$ at the end of each query.

We describe two different verification strategies as below, where the first one is adopted by the challenger of the security game **Game 2.k** and the second one serves as the reference:

- **SimulatedVerifier**: The challenger keeps a local copy of all data files and authentication tags, where data files are chosen by the adversary in a store-query and authentication tags are generated by the challenger in response to that store-query. For each verification-query, the challenger computes the corresponding genuine proof (ψ_1, ψ_2) from the challenger's local copy of data files and authentication tags. If the adversary's proof (ψ_1', ψ_2') is the same as the genuine proof (ψ_1, ψ_2), i.e. $(\psi_1', \psi_2') = (\psi_1, \psi_2)$, then the challenger outputs `accept`; otherwise outputs `reject`.
- **ImaginaryVerifier**: An imaginary verification oracle $\mathcal{O}^{\mathsf{EPOS.Verify}(sk;\cdot)}$ which somehow has access to the private key sk.

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

Let us code `accept` with the bit '1' and code `reject` with the bit '0', and denote with $a_i \in \{0,1\}$ be the decision bit output by the imaginary verification oracle for the i-th verification-query made by the adversary $\mathcal{A}$ in **Game 2.k**; $b_i \in \{0,1\}$ be the corresponding decision bit output by the simulated verifier. Furthermore, let $A_k := a_1 a_2 \ldots a_k \in \{0,1\}^k$ and $B_k := b_1 b_2 \ldots b_k \in \{0,1\}^k$. We notice that

– $a_{k+1} \neq b_{k+1}$ indicates the event that the adversary wins **Game 2.k** (Note that the $(k+1)$-th query is made in the **retrieve** phase in the **Game 2.k**).
  - $(a_{k+1} = 1, b_{k+1} = 0)$ indicates the event that the adversary wins **Game 2.k**, since the adversary's proof is valid (accepted by **ImaginaryVerifier**), but different from the genuine proof (rejected by **SimulatedVerifier**).
  - $(a_{k+1} = 0, b_{k+1} = 1)$ is impossible, since in our EPOS scheme, the genuine proof is valid certainly.
– $a_{k+1} = b_{k+1}$ indicates the event that the adversary loses **Game 2.k**.

Claim 6 **Game 1** *is equivalent to* **Game 2.0** *to the view of the adversary. (Intuitively, asking many queries without feedback is equivalent to asking no queries.)*

Claim 7 *Let ξ be a negligible function implied in Lemma 2, such that for any PPT adversary $\mathcal{A}$, $\mathsf{Pr}[\mathcal{A}$ wins* **Game 1**$] \leq \xi$. *Then $\mathsf{Pr}[\mathcal{A}$ wins* **Game 2.0**$] \leq \xi$.

Claim 8 *If $\mathsf{Pr}[A_k = B_k] \geq X$, then $\mathsf{Pr}[A_{k+1} = B_{k+1}] \geq X(1-\xi)$.*

Proof (of Claim 8).

$$\begin{aligned}
\mathsf{Pr}[A_{k+1} = B_{k+1}] &= \mathsf{Pr}[A_k = B_k \wedge a_{k+1} = b_{k+1}] && (24)\\
&= \mathsf{Pr}[A_k = B_k] \times \mathsf{Pr}[a_{k+1} = b_{k+1} \mid A_k = B_k] && (25)\\
&\geq \mathsf{Pr}[A_k = B_k] \times \mathsf{Pr}[\mathcal{A} \text{ loses } \textbf{Game 1}] && (26)\\
&\geq X(1-\xi). && (27)
\end{aligned}$$

Claim 9 $\mathsf{Pr}[A_k = B_k] \geq (1-\xi)^k$.

Proof (of Claim 9). We prove the above claim using mathematical induction.
Base Case: $k = 1$**.** $\mathsf{Pr}[A_1 = B_1] = \mathsf{Pr}[a_1 = b_1] = \mathsf{Pr}[\mathcal{A}$ loses **Game 2.0**$] \geq (1-\xi)$.
Induction Step: from k **to** $k+1$**.** This is just Claim 8.

Claim 10 $\mathsf{Pr}[\mathcal{A}$ *wins* **Game 2.k**$] \leq \xi + \left(1 - (1-\xi)^k\right) = (k+1)\xi + o(\xi)$.

Notice that here $o(\cdot)$ denote the little-O notation.

Proof (of Claim 10).

$$\begin{aligned}
&\mathsf{Pr}[\mathcal{A} \text{ wins } \textbf{Game 2.k}]\\
&= \mathsf{Pr}[\mathcal{A} \text{ wins } \textbf{Game 2.k} \wedge A_k = B_k] \;+\; \mathsf{Pr}[\mathcal{A} \text{ wins } \textbf{Game 2.k} \wedge A_k \neq B_k]\\
&\leq \mathsf{Pr}[\mathcal{A} \text{ wins } \textbf{Game 2.k} \mid A_k = B_k] \times \mathsf{Pr}[A_k = B_k] \;+\; \mathsf{Pr}[A_k \neq B_k]\\
&\leq \mathsf{Pr}[\mathcal{A} \text{ wins } \textbf{Game 2.k} \mid A_k = B_k] \;+\; \mathsf{Pr}[A_k \neq B_k]\\
&\leq \mathsf{Pr}[\mathcal{A} \text{ wins } \textbf{Game 1}] \;+\; \mathsf{Pr}[A_k \neq B_k]\\
&\leq \xi + \left(1 - (1-\xi)^k\right) = (k+1)\xi + o(\xi).
\end{aligned}$$

Notice that $\mathsf{Pr}[\mathcal{A}$ wins **Game 2.k** $\mid A_k = B_k] \leq \mathsf{Pr}[\mathcal{A}$ wins **Game 1**$]$, since in **Game 2.k**, $A_k = B_k$ indicates that the adversary gains absolutely no new information from the k verification-queries in the **Learning** phase, thus equivalent to the no-feedback setting.

Therefore, Lemma 3 is concluded from Claim 10. □