Publicly Verifiable Remote Data Integrity

Ke Zeng

NEC Laboratories, China
zengke@research.nec.com.cn

Abstract. More and more customers are outsourcing data storage to remote archive service providers that are responsible for properly preserving the data. As such, it has become crucial for an archive service to be capable of providing evidence to demonstrate the integrity of data for which it is responsible, from the time it receives the data until the expiration of the archival period. Pairing-based provable data integrity (PDI) scheme is proposed that enables not only the customer but also a third-party verifier to check remote data integrity. This PDI scheme is provably secure and efficient. Compared to the best-known prior art, our experiments under defined conditions show that our PDI scheme works 50 times faster in fingerprinting the data, and the resulting fingerprints are 30 times smaller in size.

Keywords: data outsourcing, integrity, public verifiability, pairing.

1 Introduction

Data storage outsourcing is a current trend on the Internet, in which data is stored with a global archive service instead of using one's own local storage. Internet-based online archive services are now providing their end users, including individual customers and businesses, huge amounts of storage space. For example, Apple's iDisk [1] provides 10 GB of online storage space to each .Mac member. Amazon Simple Storage Service (Amazon S3) [2] goes even further. It provides a web services interface that can be used to store and retrieve an unlimited amount of data, with fees metered in GB-months and data-transfer amounts.

Yet with more and more archive services available to store customers' digital assets such as photos, videos, emails, and file system backups, the security of the services that retain this tremendous amount of data becomes critical. As summarized in the IETF's "Long-Term Archive Service Requirements" (RFC 4810) [3], security requirements for archive services must include non-repudiation of data existence, integrity, and origin. Of these requirements, we will herein focus on data integrity. Various news reports have already revealed that even the most popular service providers may damage customers' data [4], such as emails or photos, which may have great personal value. Indeed, archive services are vulnerable to three classes of data integrity threats, namely latent faults (e.g., caused by a bit error in the storage medium), correlated faults (e.g., caused by a lack of geographic location diversity), and recovery faults (e.g., caused by improperly debugged procedures) [4]. Thus, archive service customers, in making the decision to outsource particular data, must be able to evaluate the risk of losing that data.

L. Chen, M.D. Ryan, and G. Wang (Eds.): ICICS 2008, LNCS 5308, pp. 419–434, 2008.

One solution is to empower the customers with tools that can help them periodically conduct integrity checks of their data. Another solution, as both Shah et al. [4] and Ateniese et al. [5] proposed, is to introduce a third-party verifier to whom the customers could delegate the periodic task of checking data integrity. More interestingly, the third-party verifier, who has expertise and capabilities that the customers do not, can act as an external auditor of the archive service providers. He can periodically check the integrity of all the data stored with the archive service providers and then release an audit report. Based on the audit report, customers can evaluate the risks associated with any particular archive service provider before they decide to rely on its service. The audit report may also be beneficial to the archive service provider. A positive audit report from a third party may assist the archive service provider in obtaining a favorable insurance rate [4].

It is therefore a must to develop tools for customers, the third-party verifier, and the archive service provider, such that the archive service provider can prove the integrity of data for which it is responsible, from the time it receives the data until the expiration of the archival period [3].

1.1 Related Work

There is a simple solution to tackle the data integrity issue. Initially, the data source divides the data into multiple fragments and for each fragment pre-computes a message authentication code (MAC). Whenever a verifier, be it the data source or a third party, needs to check data integrity, he retrieves from the archive service provider a number of randomly selected fragments and re-computes the MAC of each fragment for comparison. This simple solution has a drawback that its communication complexity is linear with respect to the queried data size. Moreover, in the case of a third-party verifier, sending user data to the verifier is prohibitive because it violates the data source's privacy. To avoid retrieving data from the archive service provider, one may improve this simple solution by choosing multiple secret keys and pre-computing multiple keyed-hash MACs for the data. Thus the verifier can each time reveal a secret key to the archive service provider and ask for a fresh keyed-hash MAC for comparison. However, in this method, the number of times a particular data item can be verified is limited by the number of secret keys that must be fixed a priori. When all possible secret keys are exhausted, it is then necessary to retrieve data from the archive service provider in order to compute new MACs.

Golle et al. [6] proposed a scheme to verify data storage commitment, a concept that is weaker than integrity. The drawback to their proposal is that the verifier's public key is about twice as large as the data file.

Juels et al. [7] proposed to verify data retrievability, a concept that is similar to integrity, by first encrypting the data file then embedding disguised blocks (so-called sentinels) in the ciphertext. One drawback to their proposal is that it is not data format independent, i.e., only encrypted data files can be handled. Hence their proposal is not applicable to a general archive system. Another drawback is that it allows only a limited number of challenges on the data files, which is determined by the number of sentinels embedded at the preprocessing phase.

Schwarz et al. [8] proposed to verify data integrity using an algebraic signature. The drawback to their proposal is that the communication complexity is linear with respect to the queried data size. In addition, the security of their proposal is not proven and remains in question.

Deswarte et al. [9] and Filho et al. [10] proposed to verify data integrity using an RSA-based hash function. Their proposals have the drawback that the archive service provider has to exponentiate the entire data file. As a reference, given a 2048-bit RSA modulus, MIRACL library v5.3.1 [11] reports that one modular exponentiation with a 2048-bit exponent takes 21.8 milliseconds on an Intel Core2 Quad 2.66 GHz processor[1]. Thus it would take 5715 seconds to generate one integrity proof for a merely 64-MB data file. In addition, the security of their proposals remains in question. There is no clear security reduction to the RSA problem or any well-known variant [7].

Further to [9] and [10], Sebe et al. [12] proposed to verify data integrity by first fragmenting the data, fingerprinting each fragment, and then using an RSA-based hash function on the fragments. Their proposal does not require the exponentiation of the entire file. However, the verifier has to have a local copy of the fingerprints, whose size is linear to the number of fragments. In addition, the verifier must not be a third party. Otherwise the secret information of the data source is divulged.

Yamamoto et al. [13] proposed to verify data integrity through batch verification of homomorphic hash functions on randomly selected fragments of data. The drawback to their proposal is that the verifier has to have a local copy of the hash values, whose size is linear to the number of fragments.

Shah et al. [4] proposed allowing a third-party verifier by first encrypting the data then sending a keyed hash of the encrypted data to the verifier. One drawback to their proposal is that the number of times a particular data item can be verified is limited by the number of secret keys that must be fixed beforehand. Another drawback is that their proposal is not data format independent.

Ateniese et al. [5] proposed an S-PDP scheme (where "S" stands for "sampling"), and an S-PDP-PV scheme (where "PV" stands for "public verifiability") that allows a third-party verifier. Both schemes verify data integrity by first fragmenting the data, then fingerprinting each fragment. The data integrity proof is computed, exploiting the homomorphism of the fingerprints (so-called homomorphic verifiable tags). Both schemes are provably secure and data format independent. They do not require a local copy of the data or the fingerprints and do not confine in advance the maximum number of queries. This sampling strategy introduced by Ateniese et al. is particularly beneficial in the third-party auditing scenario, where the archive service provider needs to prove the integrity of huge volumes of data, e.g., 1 TB of data, to an auditor. Sampling releases the archive service provider from having to access its entire storage, thus largely reducing the required disk I/O overhead. In addition, Ateniese et al. proposed to simplify the S-PDP scheme which yields E-PDP scheme. The E-PDP scheme with weaker security guarantees, i.e., only guarantees possession of the sum of the file blocks, is alleged more efficient than the S-PDP scheme [5].

[1] Throughout this paper, MIRACL reference speeds are always measured using only 1 of the processor's 4 CPU cores in Windows Vista 32-bit edition.

Unfortunately, both the S-PDP-PV scheme and the E-PDP scheme are unsatisfactory. The S-PDP-PV scheme has the drawbacks that it generates fingerprints that are too large in size and the time it takes to generate the fingerprints is too long as well. As a concrete example, given a 2048-bit RSA modulus, the RSA public key e is chosen to be a 6168 bits prime[2]. The S-PDP-PV scheme mandates that each fragment be less than $e/2$, otherwise the S-PDP-PV scheme is not provably secure. So a 64-MB data file would be divided into at least 87,056 fragments, each of which has a 2048-bit-long fingerprint. Thus, the combined size of all the fingerprints is about 21.2 MB. Further, given a 2048-bit RSA modulus, MIRACL library v5.3.1 reports that one RSA decryption takes 7 milliseconds. Since generating one fingerprint is computationally expensive than doing two RSA decryptions, it would take the data source at least 1218 seconds to fingerprint all the fragments of the 64-MB file.

In terms of the E-PDP scheme, Ateniese et al. [5] argued that its weaker security guarantee is of no practical issue. However, in Section 4.2 we will prove that the E-PDP scheme has no efficiency gain in practice.

To summarize, it seems rational to aim for a third-party-verifier-friendly data integrity proof protocol that satisfies the following four requirements:

1. The verifier does not need a copy of the data or fingerprints.
2. It is provably secure.
3. It is data format independent.
4. The number of allowable queries is not limited in advance.

In addition, an efficient data integrity proof protocol should take the following five factors into consideration:

1. The size of the public key
2. The computation cost for fingerprinting the data
3. The size of the fingerprints
4. The computation cost due to each protocol instance
5. The amount of communication required by each protocol instance

From this viewpoint, the S-PDP-PV scheme is the only solution we are aware of that achieves public verifiability, but it is inefficient due to its excessively high storage and computation consumption.

Our Contributions

We propose a pairing-based provable data integrity (PDI) scheme that is third-party-verifier-friendly and efficient, i.e., it satisfies all the nine requirements above.

Moreover, we implement the PDI scheme to demonstrate its efficiency. In particular, we show experimentally that it takes the PDI scheme only 22.4 seconds to fingerprint one 64-MB file and the fingerprints collectively consume 713 KB. Compared to the S-PDP-PV scheme (under defined conditions above), the PDI scheme is more than 50 times faster in generating the fingerprints and the fingerprints themselves are 30 times smaller in size.

[2] Here we choose a 6168 bits prime e for the S-PDP-PV scheme because it would result in data integrity proof that consumes 771 bytes. We will explain and justify this choice in Section 5.

2 Notations and Number-Theoretic Preliminaries

We first define some notations and review a few number-theoretic preliminaries.

2.1 Notations

If $\mathcal{S}$ is a finite set, $x \in_R \mathcal{S}$ denotes that x is chosen from $\mathcal{S}$ uniformly at random. For two algorithms $A(\cdot)$ and $B(\cdot)$, $(x;y) \leftarrow (A \,||\, B)(\varsigma)$ denotes the joint execution of $A(\cdot)$ and $B(\cdot)$ on the same input string ς and the same random tape, and $A(\cdot)$'s output is assigned to x and $B(\cdot)$'s to y. Let $\Omega(\cdot)$ be an arbitrary Boolean predicate, i.e., a function that upon input of some string ς outputs either TRUE or FALSE. By $\varsigma \leftarrow A(x) : \Omega(\varsigma)$ we denote that $\Omega(\varsigma)$ is TRUE after ς was obtained by running algorithm $A(\cdot)$ on input x.

A function $adv(\kappa)$ is said to be negligible (in κ) if for every positive polynomial $p(\cdot)$ and sufficiently large κ, $adv(\kappa) < 1 / p(\kappa)$.

2.2 Number-Theoretic Preliminaries

Throughout this paper, we use the traditional multiplicative group notation, instead of the additive notation often used in elliptic curve settings.

Let $\mathbb{G}_1 = \langle g_1 \rangle$ and $\mathbb{G}_2 = \langle g_2 \rangle$ be two finite cyclic groups with additional group $\mathcal{G} = \langle \boldsymbol{g} \rangle$ such that $|\mathbb{G}_1| = |\mathbb{G}_2| = |\mathcal{G}| = p$ where p is a large prime. Bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathcal{G}$ is a function, such that it is: bilinear – for all $h_1 \in \mathbb{G}_1$, $h_2 \in \mathbb{G}_2$, and for all $a, b \in \mathbb{Z}_p$, $e(h_1^{\,a}, h_2^{\,b}) = e(h_1, h_2)^{ab}$; non-degenerate – $\exists h_1 \in \mathbb{G}_1$, $\exists h_2 \in \mathbb{G}_2$ such that $e(h_1, h_2) \neq \mathcal{I}$ where $\mathcal{I}$ is the identity element of $\mathcal{G}$; and computable – there exists an efficient algorithm for computing e.

We suppose there is a setup algorithm $Setup(\cdot)$ that, upon input of security parameter 1^κ, outputs the above settings of the bilinear map and writes this as $(p, \mathbb{G}_1, \mathbb{G}_2, \mathcal{G}, g_1, g_2, e) \leftarrow Setup(1^\kappa)$. We omit $\boldsymbol{g}$ from the expression as it is easy to see that $\boldsymbol{g} = e(g_1, g_2)$.

q-SDH Assumption. For all probabilistic polynomial time (p.p.t.) adversaries $\mathcal{A}$, $adv(\kappa)$ defined as follows is a negligible function:

$$(p, \mathbb{G}_1, \mathbb{G}_2, \mathcal{G}, g_1, g_2, e) \leftarrow Setup(1^\kappa); a \in_R \mathbb{Z}_p;$$
$$\Pr\left[(x, y) \leftarrow \mathcal{A}(g_2^{\,a}, g_2^{\,a^2}, \cdots, g_2^{\,a^q}) : x \in \mathbb{Z}_p \wedge y = g_1^{\,1/(a+x)}\right] = adv(\kappa)$$

The q-SDH assumption has been shown to hold in generic bilinear groups by Boneh et al. [14].

KEA1 Assumption. For any p.p.t. adversaries $\mathcal{A}$, there exists a p.p.t. extractor $\mathcal{E}$, such that $adv(\kappa)$ defined as follows is a negligible function:

$$(p,\mathbb{G}_1,\mathbb{G}_2,\mathcal{G},g_1,g_2,e)\leftarrow Setup(1^\kappa);x\in_R\mathbb{Z}_p;h\in_R\mathbb{G}_1;$$
$$\Pr\left[(Y,C;\alpha)\leftarrow(\mathcal{A}\,||\,\mathcal{E})(h^x):Y\in\mathbb{G}_1\wedge Y=C^x\wedge C\neq h^\alpha\right]=adv(\kappa)$$

Informally, the KEA1 assumption says that the only way any adversary can output $Y=C^x$ from h^x is to pick some $\alpha\in\mathbb{Z}_p$, let $C=h^\alpha$ and let $Y=(h^x)^\alpha$.

q-KEA Assumption. For any p.p.t. adversaries $\mathcal{A}$, there exists a p.p.t. extractor $\mathcal{E}$, such that $adv(\kappa)$ defined as follows is a negligible function:

$$(p,\mathbb{G}_1,\mathbb{G}_2,\mathcal{G},g_1,g_2,e)\leftarrow Setup(1^\kappa);x\in_R\mathbb{Z}_p;\{h_j\}\in_R\mathbb{G}_1{}^q;$$
$$\Pr\left[(Y,C;\{\alpha_j\})\leftarrow(\mathcal{A}\,||\,\mathcal{E})(\{h_j{}^x\}):Y\in\mathbb{G}_1\wedge Y=C^x\wedge C\neq\prod_j h_j{}^{\alpha_j})\right]=adv(\kappa)$$

In the example case of $q=2$, the 2-KEA assumption (referred to as KEA3 assumption in [15] and XKEA assumption in [16]), says that the only way any adversary can output $Y=C^x$ from $h_1{}^x$ and $h_2{}^x$ is to pick some $(\alpha_1,\alpha_2)\in\mathbb{Z}_p{}^2$, let $C=h_1{}^{\alpha_1}h_2{}^{\alpha_2}$ and let $Y=(h_1{}^x)^{\alpha_1}(h_2{}^x)^{\alpha_2}$.

It is proved in [15] that the KEA3 assumption is a natural extension of KEA1. Following that, the q-KEA assumption is trivially provable as a natural extension of the KEA1 assumption as well.

The KEA1 assumption has been shown, by Abe et al. [16] and also by Dent [17] independently, to hold in generic (bilinear) groups. In [16], Abe et al. further proved Lemma 1 as shown below.

Lemma 1. Under the KEA1 assumption, for any p.p.t. adversaries $\mathcal{A}$, there exists a p.p.t. extractor $\mathcal{E}$, such that $adv(\kappa)$ defined as follows is a negligible function:

$$(p,\mathbb{G}_1,\mathbb{G}_2,\mathcal{G},g_1,g_2,e)\leftarrow Setup(1^\kappa);x\in_R\mathbb{Z}_p;\{h_1,h_2\}\in_R\mathbb{G}_1{}^2;$$
$$\Pr\left[\begin{array}{r}(\{Y_i\},\{C_i\};\{\alpha_{i1},\alpha_{i2}\})\leftarrow(\mathcal{A}\,||\,\mathcal{E})(h_1{}^x,h_2{}^x):\\ \exists i(Y_i\in\mathbb{G}_1\wedge Y_i=C_i{}^x\wedge C_i\neq h_1{}^{\alpha_{i1}}h_2{}^{\alpha_{i2}}),i=1,2,\ldots,n\end{array}\right]=adv(\kappa)$$

Lemma 2, below, is a natural extension of Lemma 1. The proof technique for Lemma 1 applies to Lemma 2 as well.

Lemma 2. Under the KEA1 assumption, for any p.p.t. adversaries $\mathcal{A}$, there exists a p.p.t. extractor $\mathcal{E}$, such that $adv(\kappa)$ defined as follows is a negligible function:

$$(p,\mathbb{G}_1,\mathbb{G}_2,\mathcal{G},g_1,g_2,e)\leftarrow Setup(1^\kappa);x\in_R\mathbb{Z}_p;\{h_j\}\in_R\mathbb{G}_1{}^q;$$
$$\Pr\left[\begin{array}{r}(\{Y_i\},\{C_i\};\{\alpha_{ij}\})\leftarrow(\mathcal{A}\,||\,\mathcal{E})(\{h_j{}^x\}):\\ \exists i(Y_i\in\mathbb{G}_1\wedge Y_i=C_i{}^x\wedge C_i\neq\prod_j h_j{}^{\alpha_{ij}}),i=1,2,\ldots,n\end{array}\right]=adv(\kappa)$$

3 Provable Data Integrity Scheme

We start with the definition and description of our provable data integrity (PDI) scheme, followed by its security analysis, discussion, and performance evaluation.

3.1 Definition

A PDI scheme is a collection of four algorithms, $KeyGen(\cdot)$, $Fingerprint(\cdot)$, $GDIP(\cdot)$, and $VDIP(\cdot)$.

$(pk, sk) \leftarrow KeyGen(1^{\kappa})$. This probabilistic algorithm takes as input security parameter 1^{κ}, and returns public key pk and private key sk.

$(z_k, T_i^k) \leftarrow Fingerprint(pk, sk, \mathcal{F}_k)$. This algorithm takes as input public key pk, private key sk, and a file $\mathcal{F}_k$. Let FN_k denote $\mathcal{F}_k$'s file reference, which is, for example, the file name of $\mathcal{F}_k$ plus a unique serial number. The file $\mathcal{F}_k$ is an ordered collection of super-blocks M_i^k while each super-block M_i^k is an ordered collection of file blocks m_j^{ik}. This algorithm returns a file key z_k for $\mathcal{F}_k$ and the fingerprint T_i^k for M_i^k. There is a one-to-one mapping between z_k and FN_k. The length of the file block and the number of file blocks that one super-block can aggregate are determined by certain parameters of pk.

$\mathrm{V} \leftarrow GDIP(pk, FN, \mathcal{F}, \mathrm{T}, chal)$. This algorithm takes as input public key pk, a file $\mathcal{F}$ that is an ordered collection of super-blocks M_i, the file reference FN of $\mathcal{F}$, an ordered collection of fingerprints $\mathrm{T} = \{T_i\}$ for $\{M_i\}$, and a challenge $chal$. It returns a data integrity proof V.

$\{\mathrm{TRUE}, \mathrm{FALSE}\} \leftarrow VDIP(pk, FN, z, chal, \mathrm{V})$. This algorithm takes as input public key pk, a file reference FN, a file key z, a challenge $chal$, and the data integrity proof V. It returns TRUE if the integrity of the file $\mathcal{F}$ determined by FN is verified as correct, or FALSE otherwise.

Based on the PDI scheme, a PDI system could be easily constructed in three phases, **Setup**, **Store**, and **Challenge**.

Setup: A client's public key and private key are initialized by invoking $KeyGen(\cdot)$. The client publishes his public key.

Store: A client in possession of a file runs $Fingerprint(\cdot)$ to fingerprint the file and stores the file and its fingerprints with the server. The client deletes the file and the fingerprints from his local storage.

Challenge: Any one, a third party or the client himself, can verify integrity of the client's file by invoking $VDIP(\cdot)$, which requires to challenge the server and the

server executing $GDIP(\cdot)$ to respond. It is notable that this phase can be executed an unlimited number of times.

3.2 The PDI Scheme

Now we start to depict our PDI scheme. For simplicity, we regard $(p, \mathbb{G}_1, \mathbb{G}_2, \mathcal{G}, g_1, g_2, e) \leftarrow Setup(1^\kappa)$ and pseudo random functions $prf_1 : \{0,1\}^* \rightarrow \mathbb{Z}_p$, $prf_2 : \{0,1\}^* \rightarrow \mathbb{G}_1$, $prf_3(\phi) : \{0,1\}^* \rightarrow \mathbb{Z}_{2^\phi}$ as system parameters.

KeyGen(·): Select $sk = (x, s, t) \in_R \mathbb{Z}_p^3$. Compute $Y = g_2^x$, $Y_s = Y^s$, $g_{2s} = g_2^s$, $h_j = g_1^{prf_1(j,t)}$, and $S_j = h_j^s$, $j = 1, 2, \ldots, n_B$. Choose a positive integer $len < \log p$ that determines the length in bits of each file block and another positive integer n_B which determines the number of file blocks that one super-block contains.

Finally, output $sk = (x, s, t)$ and $pk = \left(Y, Y_s, g_{2s}, \left\{h_j, S_j\right\}, len, n_B\right)$.

Given pk, a file $\mathcal{F}$ with file reference FN can be divided into $N = \left\lceil Flen / len \right\rceil$ file blocks m_i, each of which is len bits long, where $Flen$ is the length of $\mathcal{F}$ in bits. And every n_B consecutive file blocks constitute one super-block. In other words, the file $\mathcal{F}$ is logically divided into N file blocks and organized into $n_{SB} = \left\lceil N / n_B \right\rceil$ super-blocks. Notice that the file $\mathcal{F}$ will be logically padded with zero in the case that $Flen < N \cdot len$ or $Flen < n_{SB} \cdot (n_B \cdot len)$.

Fingerprint(·): Upon input of $sk = (x, s, t)$, $pk = \left(Y, Y_s, g_{2s}, \left\{h_j, S_j\right\}, len, n_B\right)$, and a file $\mathcal{F}$ with file reference FN, select $z \in_R \mathbb{Z}_p$ as $\mathcal{F}$'s file key and execute the following for each super-block M_i of $\mathcal{F}$, $i = 1, 2, \ldots, n_{SB}$:

a) Compute a locator $W_i = prf_2(i, z, FN) \in \mathbb{G}_1$.

b) Compute $R_i = \sum_{j=1}^{n_B} prf_1(j,t) \cdot m_{(i-1)\cdot n_B + j} \in \mathbb{Z}_p$ and $T_i = (W_i \cdot g_1^{R_i})^{\frac{1}{x+z}} \in \mathbb{G}_1$.

Finally, output the file key z and the fingerprints $\mathrm{T} = \left\{T_i\right\} \in \mathbb{G}_1^{n_{SB}}$.

Notice that $T_i = (W_i \cdot g_1^{R_i})^{\frac{1}{x+z}} = (W_i \cdot \prod_{j=1}^{n_B} h_j^{m_{(i-1)\cdot n_B + j}})^{\frac{1}{x+z}}$ for all $i = 1, 2, \ldots, n_{SB}$.

GDIP(·): Upon input of $pk = \left(Y, Y_s, g_{2s}, \left\{h_j, S_j\right\}, len, n_B\right)$; a file $\mathcal{F}$ with file reference FN, file key z, and fingerprints $\mathrm{T} = \left\{T_i\right\}$; $chal = (l, \psi, \gamma_1, \gamma_2)$, where $1 \le \psi \le l$ and $(\gamma_1, \gamma_2) \in_R \mathbb{Z}_p^2$; execute the following:

a) Compute $\phi = \lfloor l / \psi \rfloor + 1$.

b) For each $k = 1,2,\ldots,\psi$ execute the following atomic proof procedure once (i.e., repeat the atomic proof procedure independently ψ times).

1) Consider there to be $\Phi = 2^{\phi}$ buckets in logic. For each bucket, initialize a packed fingerprint $\vec{T}_v = \mathcal{O}$, and packed file blocks $e_{vj} = 0$, where $\mathcal{O}$ is the identity of $\mathbb{G}_1$, $v = 0,1,\ldots,\Phi - 1$, $j = 1,2,\ldots,n_B$.

2) For each super-block M_i and its corresponding fingerprint T_i, randomly assign them into one bucket and add them to the bucket's packed file blocks and packed fingerprint, respectively. Specifically, for each $i = 1,2,\ldots,n_{SB}$, conduct the following:

i. Compute $\sigma = prf_3(i,k,\gamma_1) \in \mathbb{Z}_{2^{\phi}}$.

ii. Compute $\vec{T}_{\sigma}$ *= T_i (i.e., compute $\vec{T}_{\sigma} \cdot T_i$ and store back to $\vec{T}_{\sigma}$).

iii. For each $j = 1,2,\ldots,n_B$, compute $e_{\sigma j}$ += $m_{(i-1)\cdot n_B + j} \bmod p$.

3) Initiate a transformed fingerprint $\mathrm{T}_k = \mathcal{O} \in \mathbb{G}_1$ and transformed file blocks $E_j = 0$, $j = 1,2,\ldots,n_B$.

4) Assign each bucket a random number to randomize its packed fingerprint and packed file block, then add them to the transformed fingerprint and the transformed file blocks, respectively. Specifically, for each $v = 0,1,\ldots,\Phi - 1$, conduct the following:

i. Compute $a_v = prf_3(v,k,\gamma_2) \in \mathbb{Z}_{2^{\phi}}$.

ii. Compute T_k *= $\vec{T}_v^{\,a_v}$.

iii. For each $j = 1,2,\ldots,n_B$, compute E_j += $a_v \cdot e_{vj} \bmod p$.

5) Compute $\mathrm{H}_k = \prod_{j=1}^{n_B} S_j^{E_j} \in \mathbb{G}_1$ as the knowledge proof of the transformed file block T_k.

Finally, output the data integrity proof $(\mathrm{T}_k, \mathrm{H}_k)$, $k = 1,2,\ldots,\psi$ [3].

***VDIP(·)*:** Upon input of $pk = \left(Y, Y_s, g_{2s}, \{h_j, S_j\}, len, n_B\right)$; a file reference FN; a file key z; $chal = (l, \psi, \gamma_1, \gamma_2)$, where $1 \le \psi \le l$ and $(\gamma_1, \gamma_2) \in_R \mathbb{Z}_p^{\,2}$; and a data integrity proof $(\mathrm{T}_k, \mathrm{H}_k) \in \mathbb{G}_1^{\,2}$, $k = 1,2,\ldots,\psi$; execute the following:

[3] The size of the data integrity proof is proportional to the number of atomic proof procedures. However this is of little efficiency concern in practice. See Section 5 for detailed discussion on this matter.

a) For each $k = 1,2,\ldots,\psi$, execute the following atomic verification procedure once (i.e., repeat the atomic verification procedure independently ψ times).
   1) Initialize a transformed locator $\mathrm{W}_k = \mathcal{O}$.
   2) Consider there to be $\Phi = 2^{\phi}$ buckets in logic. For each bucket, initialize a packed locator $\vec{W}_v = \mathcal{O}$, $v = 0,1,\ldots,\Phi-1$.
   3) Re-compute the locators, randomly assign them into the buckets, and add them to the bucket's packed locators. Specifically, compute $\sigma = prf_3(i,k,\gamma_1) \in \mathbb{Z}_{2^{\phi}}$ and $\vec{W}_{\sigma} *= prf_2(i,z,FN)$ for each $i = 1,2,\ldots,n_{SB}$.
   4) Assign the buckets random numbers to randomize their packed locators, then add them to the transformed locator. Specifically, compute $a_v = prf_3(v,k,\gamma_2) \in \mathbb{Z}_{2^{\phi}}$ and $\mathrm{W}_k *= \vec{W}_v^{-a_v}$ for each $v = 0,1,\ldots,\Phi-1$.
   5) If $e(\mathrm{H}_k, g_2) = e(\mathrm{T}_k, Y_s \cdot g_{2s}^{\ z}) \cdot e(\mathrm{W}_k, g_{2s})$, the output is TRUE, otherwise the output is FALSE.

$VDIP(\cdot)$ outputs TRUE if and only if all the atomic verification procedures output TRUE.

3.3 Security of the PDI Scheme

Informally, the security of this PDI scheme is equivalent to the nonexistence of an adversary that is capable, within the confines of a certain game, of forging the data integrity proof on the condition that at least one file block is not present. We define the security of the PDI scheme as an adaptive chosen-file-block game. In this model, the adversary $\mathcal{A}$ is given a single public key. His goal is to output a data integrity proof. We give the adversary the power to choose all the file blocks as well as the super-blocks. The adversary is also given oracle access to fingerprint issuance on the super-blocks.

Remote Data Integrity Game

Setup. The challenger runs $(pk, sk) \leftarrow KeyGen(1^{\kappa})$, sends pk to the adversary, and keeps sk secret.

Queries. The adversary $\mathcal{A}$ makes fingerprint queries adaptively; it selects file blocks m_j^{ik} that form super-block M_i^k, assigns M_i^k to a file $\mathcal{F}_k$ that is referenced by FN_k, and sends FN_k and M_i^k to the challenger. The challenger computes $(z_k, T_i^k) \leftarrow Fingerprint(pk, sk, FN_k, M_i^k)$, stores z_k, and sends T_i^k back to $\mathcal{A}$. The file key z_k is not revealed to the adversary at this phase. The challenger needs to ensure one-to-one mapping between z_k and FN_k. One restriction on $\mathcal{A}$ is that it must not query different super-blocks with the same indexes k and i. Another restriction on $\mathcal{A}$ is that the length of the file block and the number of file blocks that

one super-block has aggregated must comply with those being determined by the parameters of pk.

Challenge. The adversary selects a file reference FN that determines a file $\mathcal{F}$. The file $\mathcal{F}$ is an ordered collection of super-blocks M_i, each of which has a fingerprint T_i. And each super-block M_i is an ordered collection of file blocks m_j^i. The challenger first outputs z that is the file key of FN, then generates a challenge $chal$ and asks the adversary for a data integrity proof.

Restricted Queriesx. The adversary $\mathcal{A}$ is allowed to continue querying fingerprints as before, except for when adding a new super-block to $\mathcal{F}$ and querying the fingerprint for the new super-block. In other words, the adversary is not allowed to expand the content of $\mathcal{F}$.

Output. The adversary $\mathcal{A}$ outputs a data integrity proof V.
The adversary wins the game if $\mathrm{TRUE} \leftarrow VDIP(pk, FN, z, chal, \mathrm{V})$.

Definition (Security of PDI Scheme). The PDI scheme is secure if for any p.p.t. adversary $\mathcal{A}$ the probability that $\mathcal{A}$ wins the Remote Data Integrity Game on a set of file blocks is negligibly close to the probability that the challenger can extract the file blocks by means of a knowledge extractor $\mathcal{E}$.

Theorem 1. The PDI scheme that achieves public verifiability is secure under the q-SDH assumption and the KEA1 assumption in the random oracle model.

4 Discussions

4.1 Sampling for the PDI Scheme

We can easily add the sampling strategy to the PDI scheme, yielding an S-PDI scheme. The S-PDI scheme in addition requires a $prf_4(n_{SB}) : \{0,1\}^* \rightarrow \mathbb{Z}_{n_{SB}}$ and the challenge will contain a third key $\gamma_3 \in_R \mathbb{Z}_p$ and a positive integer $\Lambda < n_{SB}$. By computing $\mathrm{i}_1 = prf_4(1, \gamma_3)$, $\mathrm{i}_2 = prf_4(2, \gamma_3)$, up to $\mathrm{i}_\Lambda = prf_4(\Lambda, \gamma_3)$, the indexes of Λ randomly selected super-blocks are determined. Each atomic proof/verification procedure will then only deal with the super-blocks being selected. This entails no more than changing all $i = 1, 2, \ldots, n_{SB}$ in $GDIP(\cdot)$ and $VDIP(\cdot)$ to $i = \mathrm{i}_1, \mathrm{i}_2, \ldots, \mathrm{i}_\Lambda$.

Theorem 2. The S-PDI scheme that achieves public verifiability is secure under the q-SDH assumption and the KEA1 assumption in the random oracle model.

4.2 Knowledge Error and Its Practical Implication

The standard notion of proofs of knowledge as per [19], specifically, Definition 3 of [19], contains a knowledge error that is the probability that the verifier might accept even if the prover did not in fact "know" the witness. When constructing a proof of

knowledge system, it is the knowledge error that determines the number of necessary repetitions in practice. By repetition, the knowledge error could be reduced to an arbitrarily small amount. For example, if the knowledge error is 1/2, Γ times of sequential iterations may reduce the knowledge error to $2^{-\Gamma}$ [20].

Since repetitions linearly increase the computation complexity and communication complexity of a proof of knowledge system, quantifying the knowledge error is a must before analyzing practical efficiency of a proof of knowledge scheme. However, this part was missing in [5] for the E-PDP, S-PDP, and S-PDP-PV schemes.

Claim 1. The E-PDP scheme of [5] attains knowledge error no smaller than 1/2.

Proof (Sketch). The Atomic Random Subset Test method disclosed in [18] could be used to construct a specific attack on the E-PDP scheme. Although both guarantee that congruence $\sum_{i=1}^{c} m_i' = M^* = M = \sum_{i=1}^{c} m_i$ hold, the E-PDP scheme allows a stronger adversary than the Atomic Random Subset Test method does. This is because the verifier by the Atomic Random Subset Test method receives all the file blocks m_i', whereas, the verifier by the E-PDP scheme receives only M^*. From this viewpoint, what the E-PDP scheme does is no more than first choosing c file blocks as the full set then applying subset test on the full set itself. As per Lemma 3.1[4] of [18], this allows for knowledge error 1/2. □

Recall that in [5] it was shown that a challenge on $c = 460$ randomly selected file blocks can reach a detection probability of 99.02% in the case of failure of 1% of the file blocks. However, calculating this probability has a precondition, i.e., all the c file blocks must be correctly possessed by the archive service with overwhelming probability. If with probability 1/2 that at least one of the c file blocks is incorrect, the E-PDP scheme has to be iterated in practice. For instance, $\Gamma = 15$ reiterations reduce the knowledge error to 2^{-15}, i.e., the probability that at least one of the c file blocks is incorrect is 2^{-15}. Thus the overall detection probability can reach $(1-2^{-15}) * 99.02\% > 99\%$. As the consequence, both the computation complexity and the communication complexity of the E-PDP scheme would be Γ times larger in practice.

Claim 2. The S-PDP scheme (also the S-PDP-PV scheme) of [5] attains knowledge error no smaller than 2^{-l} for each file block, where l determines the bit length of the coefficients a_i's.

Proof (Sketch). Very briefly, the Small Exponent Test method disclosed in [18] could be used to construct a specific attack on the S-PDP scheme. □

[4] Lemma 3.1 of [18] is originally proved over a finite cyclic group G of prime order p. It's not hard to generalize the proof for Lemma 3.1 to the case of a finite cyclic group of composite order p'q' regardless of whether p'q' is available to the adversary or not, e.g., G=QR(N). Theorem 3.3 of [BRG 98] can be generalized to work on QR(N) as well.

Claim 3. The PDI scheme attains knowledge error no smaller than 2^{-l} for each file block.

Proof (Sketch). Very briefly, the atomic proof procedure and the atomic verification procedure of the PDI scheme are in principle reusing the Bucket Test method, as per the findings of [18]. □

Claim 4. The S-PDP scheme (also the S-PDP-PV scheme) of [5] attains knowledge error no larger than $(c-1)\cdot 2^{-l}$ for each file block, where l determines the bit length of the coefficients a_i's and $c \geq 2$ is the number of the file blocks being chosen.

Proof (Sketch). Our proof reuses the proof technique that Damgard utilized to prove his Theorem 1 in [21]. □

Claim 5. The S-PDI scheme attains knowledge error no larger than $\frac{c}{\psi}\cdot 2^{-l}$ for each file block, where ψ is the number of repetitions required for the atomic proof/verification procedure and $c \geq \psi$ is the number of the super-blocks being chosen.

5 Efficiency of the PDI Scheme

In order to demonstrate its efficiency, we implement the PDI scheme using MIRACL library v5.3.1 in Windows Vista 32-bit edition. All experiments were conducted on a Dell Precision 390 workstation with an Intel Core2 Quad 2.66-GHz processor, 4 GB of ECC RAM, and a 400-GB RAID-0 disk array consisting of 3 SCSI drives.

The 64-MB data file we use for the experiments is AES-CBC encrypted such that our experiment results are not necessarily subject to the entropy of the data file.

We use a pairing-friendly non-supersingular curve, as per Brezing et al. [22]. Its parameters as listed below are provided by MIRACL library source code, wherein the modulus q is a 256-bit prime, the group order p is a 192-bit prime, and the embedding degree is 8. The security depends on the difficulty of a 2048-bit discrete logarithm problem [11].

$q = \text{CC485D26177A1A5FCC9D53BA93DA298FD7F2F23D8FC02A8123BF24F9548A5F15}$

$p = \text{9D0261DD89CF83D5D20198162C22C942EF68622A6DF25621}$

$A = 0,\ B = 2,\ cof = 14D13FF7298021445,\ k = 8$

We choose SHA-256 as $prf_1 : \{0,1\}^* \rightarrow \mathbb{Z}_p$ and UMAC [23] as $prf_3(\phi) : \{0,1\}^* \rightarrow \mathbb{Z}_{2^\phi}$. And we choose SHA-256 to build $prf_2 : \{0,1\}^* \rightarrow \mathbb{G}_1$.

In order to fingerprint the file, we choose $n_B = 256$ and $len = 184$, thus each super-block contains 5888 bytes and the 64-MB file consists of 11,398 super-blocks.

Our experiments show that the size of the public key is 32.5 KB[5], generating fingerprints takes 22.4 seconds[6], and the size of the fingerprints is about 713 KB.

In order to evaluate the efficiency of generating data integrity proof and verifying the proof, we choose $l = 84$ and $\psi = 12$, which means that the atomic proof/verification procedure will be reiterated 12 times to attain security level $1 - \frac{11398}{12} \cdot 2^{-84} > 1 - 2^{-74}$ for every 184 bits of the 64-MB data file. Each atomic proof/verification procedure needs to handle 256 buckets.

Our experiments show that generating the data integrity proof takes 6.395 seconds, verifying the proof takes 6.961 seconds, the size of the challenge is 50 bytes, and the size of the data integrity proof is 771 bytes.

Recall that the S-PDP-PV scheme needs at least 1218 seconds to fingerprint a 64-MB data file and those fingerprints collectively require 21.2 MB. It is therefore clear that our PDI scheme is more than 50 times faster in generating the fingerprints and the fingerprints themselves are 30 times smaller in size. It is notable that in this comparison we choose a 6168 bits prime e for the S-PDP-PV scheme. As per the S-PDP-PV scheme, this choice results in data integrity proof that consumes 771 bytes ([5], p.14). We note that choosing a larger e for the S-PDP-PV scheme could reduce the size of its fingerprints and the time to generate the fingerprints as well. Whereas a larger e at the same time increases the size of its data integrity proof. We thus compare efficiency of the PDI scheme and the S-PDP-PV scheme on condition that they generate the same size of data integrity proof. This comparison is appropriate at least for the case that the amount of communication required by each protocol instance is strictly restricted.

5.1 Speeding Up the PDI Implementation

Noting that fingerprinting can be done on multiple super-blocks simultaneously, and the atomic proof/verification procedures run independently, parallel computation should therefore be beneficial to the PDI scheme. Utilizing OpenMP [25] technology, we make the fingerprinting, proof generation, and proof verification all run in parallel. Making use of all 4 of the CPU cores of the Intel Core2 Quad 2.66 GHz processor, our experiments show that for the 64-MB file, generating fingerprints takes 5.632 seconds, generating the data integrity proof takes 1.997 seconds, and verifying the proof takes 1.769 seconds. Using the 4 CPU cores yields roughly a three- to fourfold speed increase compared to using only 1 CPU core.

Another approach for speeding up the implementation of the PDI scheme is simply reducing the intended security level. For instance, choosing $l = 48$ and $\psi = 8$ will reduce the security level to $1 - 2^{-37}$ and our experiments show that generating one integrity proof now takes 1.279 seconds, compared to 1.997 seconds for security level $1 - 2^{-74}$. It is, however, more interesting to exploit the sampling strategy, i.e., make use of the S-PDI scheme.

---

[5] Except for storing the fingerprints, we always use point compression technology, by which each elliptic curve point is stored as its x coordinate and the LSB of its y coordinate.

[6] All experimental results on time consumptions represent the mean of 10 trials.

Similar to the analysis given by [5], a challenge on 460 super-blocks can reach a detection probability of 99.02% in the case of failure of 1% of the super-blocks. Therefore, choosing $l = 23$, $\psi = 4$, and 460 super-blocks per challenge, the overall detection probability would be $(1 - \frac{460}{4} \cdot 2^{-23}) * 99.02\% > 99\%$. We implement the S-PDI scheme and our experiments show 0.312 seconds in generating one integrity proof and 0.077 seconds in verifying the proof, when all 4 CPU cores are utilized. Note that in this case, the size of the data integrity proof is mere 257 bytes.

6 Conclusions and Future Work

In this paper, we present a provably secure and efficient scheme, which enables not only the data source but also a third-party verifier to check remote data integrity.

There are still some problems yet to be resolved. First, the PDI scheme is not able to fingerprint data file incrementally. Our Remote Data Integrity Game prohibits a data file been modified in whatever manners once the file has been fingerprinted. Second, in the case that the data source is malicious, a third-party verifier who dares to be responsible for the integrity of the data is in danger because the data source can arbitrarily manipulate the data stored with the archive service provider while the fingerprints remain valid. Third, in the case that a third-party verifier colludes with the archive service provider, one can easily surmise that the verifier can fabricate a favorable audit report for a customer's precious file even if the archive service provider has deleted the file completely. Last but not least, it is always interesting to find novel third-party-verifier-friendly schemes that are in the standard model.

Acknowledgements

The author is indebted to Hao Lei, Ye Tian, Li-Ming Wang, and Yu-Liang Zheng for insightful discussions during the development of the ideas herein presented. The author is particularly grateful to the anonymous referees for their useful comments. The author would like to thank Min-Yu Hsueh and Toshikazu Fukushima for their support to this research.

References

1. Apple iDisk, `http://www.apple.com/dotmac/idisk.html`
2. Amazon Simple Storage Service (Amazon S3), `http://aws.amazon.com/s3`
3. Wallace, C., Pordesch, U., Brandner, R.: Long-Term Archive Service Requirement, RFC 4810. IETF Network WG (2007)
4. Shah, M.A., Baker, M., Mogul, J.C., Swaminathan, R.: Auditing to Keep Online Storage Services Honest. In: 11th Workshop on Hot Topics in Operating Systems (HotOS-XI), Usenix (2007)
5. Ateniese, G., Burns, R., Curtmola, R., Herring, J., Kissner, L., Peterson, Z., Song, D.: Provable Data Possession at Untrusted Stores. In: 14th ACM conference on Computer and Communications Security (CCS 2007), pp. 598–609. ACM Press, New York (2007), `http://eprint.iacr.org/2007/202/`

6. Golle, P., Jarecki, S., Mironov, I.: Cryptographic primitives enforcing communication and storage complexity. In: Blaze, M. (ed.) FC 2002. LNCS, vol. 2357, pp. 120–135. Springer, Heidelberg (2003)
7. Juels, A., Kaliski, B.S.: PORs: Proofs of Retrievability for Large Files. Report 2007/243, Cryptology ePrint archive (2007)
8. Schwarz, T.S.J., Miller, E.L.: Store, Forget, and Check: Using Algebraic Signatures to Check Remotely Administered Storage. In: IEEE International Conference on Distributed Computing Systems (ICDCS 2006), p. 12. IEEE Press, Los Alamitos (2006)
9. Deswarte, Y., Quisquater, J.J., Saidane, A.: Remote Integrity Checking. In: 6th IFIP TC-11 WG 11.5. In: Working Conference on Integrity and Internal Control in Information Systems (IICIS 2003), pp. 1–11. IFIP Press (2003)
10. Filho, D.L.G., Baretto, P.S.L.M.: Demonstrating Data Possession and Uncheatable Data Transfer. Report 2006/150, Cryptology ePrint Archive (2006)
11. MIRACL, Multi-precision Integer and Rational Arithmetic C Library, http://www. shamus.ie
12. Sebe, F., Ferrer, J.D., Balleste, A.M., Deswarte, Y., Quisquater, J.J.: Efficient Remote Data Possession Checking in Critical Information Infrastructures. IEEE Transactions on Knowledge and Data Engineering 20(8), 1034–1038 (2007)
13. Yamamoto, G., Oda, S., Aoki, K.: Fast Integrity for Large Data. In: Workshop on Software Performance Enhancement for Encryption and Decryption (SPEED 2007), pp. 21–32. COSIC Press (2007)
14. Boneh, D., Boyen, X.: Short Signatures without Random Oracles. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 56–73. Springer, Heidelberg (2004)
15. Bellare, M., Palacio, A.: The Knowledge-of-Exponent Assumptions and 3-Round Zero-Knowledge Protocols. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 273–289. Springer, Heidelberg (2004)
16. Abe, M., Fehr, S.: Perfect NIZK with Adaptive Soundness. Report 2006/423, Cryptology ePrint Archive (2006)
17. Dent, A.W.: The Hardness of the DHK Problem in the Generic Group Model. Report 2006/156, Cryptology ePrint Archive (2006)
18. Bellare, M., Garay, J.A., Rabin, T.: Fast Batch Verification for Modular Exponentiation and Digital Signatures. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 236–250. Springer, Heidelberg (1998)
19. Bellare, M., Goldreich, O.: On Defining Proofs of Knowledge. In: Brickell, E.F. (ed.) CRYPTO 1992. LNCS, vol. 740, pp. 390–420. Springer, Heidelberg (1993)
20. Damgard, I., Pfitzmann, B.: Sequential Iteration of Interactive Arguments and an Efficient Zero-Knowledge Argument for NP. In: Larsen, K.G., Skyum, S., Winskel, G. (eds.) ICALP 1998. LNCS, vol. 1443, pp. 772–783. Springer, Heidelberg (1998)
21. Damgard, I.: On Σ-protocols, http://www.daimi.au.dk/~ivan/Sigma.pdf
22. Brezing, F., Weng, A.: Elliptic Curves Suitable for Pairing Based Cryptography. Report 2003/143, Cryptology ePrint Archive (2003)
23. Black, J., Halevi, S., Krawczyk, H., Krovetz, T., Rogaway, P.: UMAC: Fast and Secure Message Authentication. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 215–233. Springer, Heidelberg (1999)
24. Kaliski, B.: TWIRL and RSA Key Size. RSA Laboratories Technical Notes and Reports, http://www.rsa.com/rsalabs/node.asp?id=2004
25. OpenMP, Open Multi-Processing Application Program Interface (API) Specification for Parallel Programming, http://www.openmp.org
26. OpenSSL, The Open Source Toolkit for SSL/TLS, http://www.openssl.org