A Novel Proof of Data Possession Scheme based on Set-Homomorphic Operations

Nesrine Kaaniche, Maryline Laurent, Sébastien Canard

A Novel Proof of Data Possession Scheme based on Set-Homomorphic Operations

Nesrine Kaaniche[1], Maryline Laurent[1], Sébastien Canard[2]
[1]
SAMOVAR, Telecom SudParis, CNRS, University Paris-Saclay
e–mail: {Nesrine.Kaaniche, Maryline.Laurent}@telecom-sudparis.eu
[2]
Orange Labs, 42 rue des Coutures, 1400 Caen, France

***Abstract*—The prospect of outsourcing an increasing amount of data to a third party and the abstract nature of the cloud promote the proliferation of security and privacy challenges, namely, the remote data possession checking.**
This work addresses this security concern, while supporting the verification of several data blocks outsourced across multiple storing nodes. We propose a set homomorphic proof of data possession, called SHoPS, supporting the verification of aggregated proofs. It proposes a deterministic Proof of Data Possession (PDP) scheme based on interactive proof protocols. Our approach has several advantages. First, it supports public verifiability where the data owner delegates the verification process to another entity, thus releasing him from the burden of periodical verifications. Second, it allows the aggregation of several proofs and the verification of a subset of data files' proofs while providing an attractive communication overhead.

I. Introduction

The explosive growth of data continues to rise the demand for new storage and network capacities, along with an increasing need for cost effective architectures [4]. Thus, recent years have witnessed the trend of leveraging cloud data storage.
However, these promising data storage services bring many challenging design issues, mainly due to the loss of control on outsourced data. That is, cloud data are often subject to a large number of attack vectors and the responsibility of securely managing these data is splitting across multiple storage capacities. Nonetheless, in order to reduce operating costs and save storage capacities, dishonest providers might intentionally slight these replication procedures, resulting in unrecoverable data errors or even data loss. Even when cloud providers implement a fault tolerant policy, clients have no technical means of verifying that their files are not vulnerable, for instance, to drive-crashes. There is an implementation of remote data checking at the three following levels:
(1) between a client and a CSP – a cloud client should have an efficient way to perform periodical integrity verifications, without keeping the data locally. This client's concern is magnified by his constrained storage and computation capabilities and the large size of outsourced data.
(2) within a CSP – for the CSP to check the integrity of data blocks stored across multiple storage nodes, in order to mitigate byzantine failures and drive-crashes.
(3) between two CSPs – in the case of the cloud of clouds scenarios, where data are divided on different cloud infrastructures. Therefore, a CSP, through its cloud gate, should periodically verify the authenticity of data blocks hosted by another cloud platform.
Many approaches have been proposed, in order to ensure remote data checking [1], [5], [3], [2], [7]. These schemes are referred to as *Provable Data Possession* PDP schemes. Under different security models, several schemes ensure integrity verifications of stored data on untrusted remote servers. They are designed to guarantee several requirements, namely lightweight and robust verification, computation efficiency and constant communication cost. These PDP techniques are widely analyzed into two categories, according to the role of the verifier: private verifiability, where only the data owner can verify the server's data possession, and public verifiability, where any authorized entity can perform the verification procedure.

In this work, we present SHoPS, a Set-Homomorphic Proof of data possession Scheme, supporting the 3 levels of data verification. That is, stored across multiple storage nodes, SHoPS takes advantage of the computation and storage capabilities of the storage nodes. Each node has to provide proofs of local data block sets. Then, the cloud gate is responsible for performing operations on received proofs, while preserving the authenticity of the resulting proof.
Indeed, we introduce the set homomorphism property, as our scheme allows verifying sets in a way that any authorized verifier can check the union of two proof sets, while considering the whole data file, or the intersection between two data blocks, while checking integrity proofs over versions of logging files, as conversations on social networks.
In addition, in the key role of public verifiability and the privacy preservation support, our proposed scheme aims to address the issue of provable data possession in cloud storage environments, following three substantial aspects: *security level, public verifiability* and *performances*.

Paper Organization– the remainder of this paper is organized as follows. First, Section II introduces the security requirements. Then, Section III gives a SHoPS overview and presents our construction. Finally, Section IV discusses the set-homomorphic operations, before concluding in Section V.

II. Security Requirement Analysis

The Proof of Data Possession is a challenge response protocol enabling a client to check whether a file data D stored on a remote cloud server is available in its original form. The simplest solution to design a PDP scheme is based on a hash function H. That is, the client pre-calculates k random challenges $c_i, i \in \{1, k\}$ and computes the corresponding proofs, $p_i = H(c_i||D)$. During the challenging procedure, the client sends c_i to the server which computes $p'_i = H(c_i||D)$. If the comparison holds, the client assumes that the server preserves the correct data file. This solution is concretely unfeasible because the client can verify the authenticity of the files on the server only k times.

Additionally, stored across multiple storage nodes, each node has to compute the related possession proof, based on a received challenge. As such, the aggregation process results in the removal of some redundant proofs transmitted from different nodes, in order to minimize the communication latency. Generally, the processing overhead at the client side is also reduced, but the new proof is longer than the original generated proofs. As such, to guarantee the authenticity of the resulting proof while avoiding the shortcuts of the classical forwarding, we propose an aggregate proof scheme, using set-homomorphic properties.

The design of our protocol is motivated by providing support of both robustness and efficiency. SHoPS has to fulfill the following requirements:

- ***Public verifiability–*** the public verification is an important requirement, allowing an authorized entity to verify the correctness of data. Thus, the data owner is relieved from the burden of storage and computation.
- ***Unlimited challenges–*** the number of challenges should be unlimited. This condition is considered as important to the efficiency of a PDP scheme.
- ***Low computation complexity–*** on one hand, for scalability reasons, the amount of computation at the cloud server should be minimized, as it may be involved in concurrent interactions. On the other hand, the proposed scheme should also have low processing complexity, at the client side.
- ***Low communication overhead–*** an efficient PDP should minimize the usage of bandwidth.
- ***Low storage cost–*** the limited storage capacities of the user devices has a critical importance in designing our solution. As such, low storage cost at the client side is highly recommended.

III. SHoPS: A Set-Homomorphic PDP Scheme

A. Model

We now describe the following main algorithms related to a homomorphic proof of data possession scheme, where λ is a security parameter. We first give the generation algorithm.

- `gen` : $\{1\}^\lambda \rightarrow {\mathcal{K}_{pub}}^2 \times \mathcal{K}_{pr} \times {\mathbb{G}_2}^{2q-1}$ – given a security parameter λ, this algorithm outputs the data owner public and secret keys $(pk, \hat{pk}, sk)$, and a set of public credentials, with respect to the Diffie-Hellman Exponent assumption.

To tolerate drive failures, each data file F is stored with redundancy, based on n-block erasure coding algorithm: $F = B_1 \| \cdots \| B_n$. Hence, each outsourced data file F is divided into n *blocks*, and each block B_i is divided into q *subblocks*, where q is part of the $param$ output by the `pgen` algorithm above. We now consider the file storing part. The following algorithm should be executed for each block B_i of a file F.

- `stp` : $(pk, sk, B) \rightarrow (ID, \varpi)$ – given the key pair (pk, sk) and a data block $B \in \{0,1\}^*$ sent to the cloud service provider, the setup algorithm generates a block identifier ID and a corresponding tag ϖ that are both published.

The last phase occurs when the verifier wants to check that the cloud service provider is in possession of a given data block file. This phase is divided into 3 mandatory algorithms (challenge `clg`, proof `prf` and verification `vrf`) and 1 optional (homomorphic operation `homop`). The verification algorithm `vrf` has two different variants.

- `clg` : $(pk, ID) \rightarrow (c, \eta)$ – this algorithm is computed by the verifier (client or user) and takes as input the public key pk and the identifier ID of a block of a file. It generates a challenge c (sent to the prover, i.e., the cloud service provider) and a nonce η (kept secret by the verifier).
- `prf` : $(pk, ID, c) \rightarrow \sigma$ – this algorithm computes the cloud service provider's response σ to a challenge c, using the encoded file blocks B stored on the server disks and related to the identifier ID.
- `homop` : $(pk, c, \sigma_i, \sigma_j, f) \rightarrow \sigma_h$ – this algorithm permits, using pk, a proof σ_i related to a block B_i and a challenge c and a proof σ_j related to a block B_j and the challenge c, to obtain the proof σ_h related to $f(B_i, B_j)$ and the challenge c, where f is a function on the two blocks B_i and B_j.
- $\texttt{vrf}_c$: $(pk, sk, ID, \varpi, c, \eta, \sigma) \rightarrow (0/1)$ – this is a verification algorithm, executed by the client, taking on input the key pair (pk, sk), the identifier ID of a block B, the tag ϖ generated during the `stp` algorithm, the challenge c and nonce η and using the cloud server's response σ. The output 1 denotes *accept*, i.e., the client has successfully verified correct storage by the cloud storage provider. Conversely, 0 denotes *reject*.
- $\texttt{vrf}_u$: $(pk, ID, \varpi, c, \eta, \sigma) \rightarrow (0/1)$ – this is a verification algorithm, executed by an authorized user, and similar to the previous one $\texttt{vrf}_c$, except that it does not take on input the secret key sk.

The correctness of a homomorphic proof of data possession scheme states that, after:
– the execution of the generation algorithms to obtain (pk, sk);
– the creation of two tags tag_i and tag_j, for a block B_i (resp. B_j), by the `stp` algorithm together with the creation of a challenge and nonce (c, n) for the same block;
– the creation of a proof σ_i (resp. σ_j), on input the challenge

c and the block B_i (resp.B_j) ;
– the creation of σ_h on input σ_i and σ_j for any function f;
the verification algorithms $\texttt{vrf}_c$ and $\texttt{vrf}_u$ necessarily outputs 1 for the proofs σ_i, σ_j and σ_h.

B. Construction

SHoPS is based on techniques closely related to the well-known Pederson commitment scheme [6]. That is, we extend the Pederson scheme to obtain a kind of a generalized commitment, in a subblock-index manner, providing fault-tolerance stateless verification. As such, for each verification session, the verifier generates a new pseudo random value and new index challenge position of the considered data file block, thus making messages personalized for each session. Additionally, we propose two verification processes. The first scheme restricts the verification to the data owner using only his private key. The second applies when the verification is performed using public credentials.

In this section, we presents SHoPS single block PDP scheme. The single data block proof is a PDP scheme restricted to a single block. The proofs correspond to all subblocks of a data block. In the following, we provide a detailed description of the steps, introduced in Section III-A, that are conducted in each of the two aforementioned phases. The gen and stp are, respectively, presented by Algorithm 2

Algorithm 1 gen procedure

Input: system security parameter (λ)
Output: public keys $(pk, \hat{pk})$, master secret key pr and public parameters $param = \{g_i\}_{1 \leq i \leq 2q; i \neq q+1}$
Choose a multiplicative group $\mathbb{G}_1$ of a prime order q,
Select g a generator of $\mathbb{G}_1$;
$\alpha \xleftarrow{R} {\mathbb{Z}_p}^*$; $param = \{g\}$;
for all $j \in [1 \dots 2q]$ **do**
  $param \leftarrow param \cup \{g^{\alpha^j}\}$
end for
$s \xleftarrow{R} \mathbb{Z}_p$; $pr \leftarrow s$; $pk \leftarrow g^s$; $\hat{pk} \leftarrow g_{q+1}^s$;
return $(pk, \hat{pk}, pr, \{g_i\}_{1 \leq i \leq 2q; i \neq q+1})$

and Algorithm 1. That is, each set of subblocks $\pi_{i,j}$ of B_i is presented by an accumulator $\varpi_i = \prod_{j=1}^{q} g_{q+1-j}^{\pi_{i,j}\ pr}$.

Algorithm 2 stp procedure

Input: Data block (B_i), private key pr and $param$
Output: Data block accumulator ϖ
$\varpi_i = 1$;
for all $j \in [1 \dots q]$ **do**
  $\varpi_i \leftarrow \varpi_i * g_{q+1-j}^{\pi_{i,j}\ pr}$;
end for
return (ID_{B_i}, ϖ_i)

1) clg *procedure:* The clg procedure is executed by the client and yields a challenge for the cloud server. The client chooses at random a subblock position $k \in \{1, q\}$ and a nonce η. The challenge $c \in \mathcal{C}$ consists on a random block index and the public key element $\hat{pk}$ hidden with a random nonce η as $c = (k, \hat{pk}^{\eta})$.

2) prf *procedure:* The prf, executed by the server, has to generate a valid proof of data possession of a given data block B_i. That is, in his response, the server has to provide a new valid accumulator using the random η sent by the client. In our construction, the prf is presented by Algorithm 3. For the sake of consistency, we suppose that the server possesses a version of the data block file which is potentially altered. Hereafter, this version is denoted by $\hat{B}_i$.

Algorithm 3 prf procedure

Input: File data block (B_i), public keys $(pk, \hat{pk})$, the public parameters $param$ and the challenge $c = (k, \hat{pk}^{\eta})$
Output: Proof $P = (\sigma_1, \sigma_2)$
$\sigma_1 \leftarrow (\hat{pk}^{\eta})^{\pi_{i,k}}$;
$\hat{\varpi}_i = 1$;
for all $j \in [1 \dots q]$ **do**
  if $j \neq k$ **then**
    $\hat{\varpi}_i \leftarrow \hat{\varpi}_i * g_{q+1-j+k}^{\pi_{i,j}}$;
  end if
end for
$\sigma_2 \leftarrow \hat{\varpi}_i$;
return (σ_1, σ_2)

3) vrf *procedure:* In this section, we first present the public verification correctness. Then, we introduce the private verification process, which restricts the verification to the data owner.

Public Single Data Block Verification – An authorized verifier checks the correctness of the server response, based on public parameters. It is worth noticing that the client does not store any additional information for the proof verification. That is, the verification procedure makes only use of $param$. The verifier checks the following equality, using the random secret η, the challenge c, and the server response $P = (\sigma_1, \sigma_2)$, as presented in Equation 1.

$$[\hat{e}(g_k, \varpi_i)\hat{e}(pk, \sigma_2)^{-1}]^{\eta}\hat{e}(g, \sigma_1)^{-1} = 1 \quad (1)$$

If the equality holds, the verifier has a proof that the data block B_i exists and that it has not been altered.

Lemma 3.1: ***Public Single Data Block Verification –*** The verification procedure of Equation 1 holds if, and only if the data block file $\hat{B}_i = B_i$.

Private Single Data Block Verification – SHoPS proposes a lightweight private verification variant, relying on the private key of the data owner. For this purpose, we squeeze the proposed checking algorithm, presented in Equation 1, in order to support only two pairing functions computation. As such the private verification of a single data block B_i is as follows:

$$\hat{e}({g_k}^{\eta}, \varpi_i) \star \hat{e}(g, \sigma_1 {\sigma_2}^{s\eta})^{-1} = 1 \quad (2)$$

IV. Set-Homomorphic Properties of SHoPS

We now extend the design of the single PDP scheme, in order to support subsets of data blocks. That is, the verifier requests the cloud for data correctness proofs, while considering a sequence of set-homomorphism properties[1].

In the following, we refer to the proof aggregation, every set-operation over multiple proofs, namely, the union, the intersection and the inclusion operator.

Definition 4.1: ***Set-Homomorphic based Proof –*** We consider a message space $\mathcal{M}$, a proof space $\mathcal{P}$, a private key space $\mathcal{K}_{pr}$ and a public key space $\mathcal{K}_{pub}$. A set homomorphic based proof scheme is defined as follows. There exist two operations such as: $\odot : \mathcal{P} \times \mathcal{P} \rightarrow \mathcal{P}$ and $\circledcirc : \mathcal{K}_{pub} \times \mathcal{K}_{pub} \rightarrow \mathcal{K}_{pub}$, that satisfy the homomorphism and the correctness properties, for a set operation $\bullet$ for any messages B_i and B_j in $2^{\mathcal{M}}$, such that

$$\texttt{prf}_2(B_i \bullet B_j, c) = \texttt{prf}_2(B_i, c) \odot \texttt{prf}_2(B_j, c) \quad (3)$$

In order to prove that our scheme is set-homomorphic with regard to the union operator, we use the received proofs $\texttt{prf}(c, B_i)$ and $\texttt{prf}(c, B_j)$ corresponding to B_i and B_j, respectively, to express $\texttt{prf}(c, B_i \cup B_j)$, based on the same challenge c.

Lemma 4.2: For every data block B_i and B_j, the union operator is defined as: $B_i \cup B_j = B_i + B_j - B_i \cap B_j$

First, we express $\varpi_{B_i \cup B_j}$, using ϖ_{B_i} and ϖ_{B_j}, such as:

Lemma 4.3: For every data blocks $B_i = \{\pi_{i,1}, \cdots, \pi_{i,q}\}$ and $B_j = \{\pi_{j,1}, \cdots, \pi_{j,q}\}$, where $\pi_{i,k} \in 2^{\mathcal{M}}$ and $1 \leq k \leq q$; and given the accumulators ϖ presented in Algorithm 3, the union accumulator is such that :$\varpi_{B_i \cup B_j} = lcm(\varpi_{B_i}, \varpi_{B_j})$

Proof 4.4: The computation of $\varpi_{B_i \cup B_j}$, is performed as:

$$\begin{aligned} \varpi_{B_i \cup B_j} &= \prod_{\pi_{k,l} \in B_i \cup B_j; l \in [1,q]; k \in \{i,j\}} g_{q+1-l}^{\pi_{k,l}\ pr} \\ &= lcm\Big(\prod_{\pi_{i,l} \in B_i; l \in [1,q]} g_{q+1-l}^{pr * \pi_{i,l}}, \prod_{\pi_{j,l} \in B_j; l \in [1,q]} g_{q+1-l}^{pr * \pi_{j,l}}\Big) \\ &= lcm(\varpi_{B_i}, \varpi_{B_j}) \end{aligned}$$

To compute the least common multiple of B_i and B_j, we use the relation between gcd and lcm, as: $gcd(\varpi_{B_i}, \varpi_{B_j}) * lcm(\varpi_{B_i}, \varpi_{B_j}) = \varpi_{B_i} \varpi_{B_j}$.

In the sequel, we have:

$$\varpi_{B_i \cup B_j} = lcm(\varpi_{B_i}, \varpi_{B_j}) = \frac{\varpi_{B_i} \star \varpi_{B_j}}{gcd(\varpi_{B_i}, \varpi_{B_j})} \quad (4)$$

For instance, using the Bézout's lemma, there exist unique integers a and b, such that:

$$a\varpi_{B_i} + b\varpi_{B_j} = gcd(\varpi_{B_i}, \varpi_{B_j}) \quad (5)$$

As such, using the Equation 4 and Equation 5, we find the lcm of the two data blocks B_i and B_j as follows:

$$\varpi_{B_i \cup B_j} = lcm(\varpi_{B_i}, \varpi_{B_j}) = \frac{\varpi_{B_i} \star \varpi_{B_j}}{a\varpi_{B_i} + b\varpi_{B_j}} \quad (6)$$

[1]For ease of presentation, we prove the different properties, using two different data blocks B_i and B_j. Our operations can be extended easily to support multiple data blocks checking.

Therefore, we obtain the proof of the lemma 4.3.

Theorem 4.5: ***Union Operator –*** SHoPS considers the algorithms $\texttt{clg}, \texttt{prf}$ and $\texttt{vrf}$ defined above. Let $\texttt{homop}$ be the algorithm, presented in section III-A, such that $\bullet$ is the set union operator, as follows.

$$\texttt{prf}_2(B_i \bullet B_j, c) = \texttt{prf}_2(B_i, c) \odot \texttt{prf}_2(B_j, c) = \texttt{prf}_2(B_i, c) \star \texttt{prf}_2(B_j, c)(a * \texttt{prf}_2(B_i, c) + b * \texttt{prf}_2(B_j, c))^{-1} \quad (7)$$

where a and b satisfy : $a\texttt{prf}_2(B_i, c) + b\texttt{prf}_2(B_j, c) = gcd(\texttt{prf}_2(B_i, c), \texttt{prf}_2(B_j, c))$

Proof 4.6: we know that $a\texttt{prf}_2(B_i, c) + b\texttt{prf}_2(B_j, c) = a\hat{\varpi}_{B_i} + b\hat{\varpi}_{B_j}$. Thus, we can write: $b\hat{\varpi}_{B_i}^{-1} + a\hat{\varpi}_{B_j}^{-1} = \hat{\varpi}_{B_i \cup B_j}^{-1}$. Consequently, using Equation 6, we can write that:

$$\begin{aligned} a\hat{\varpi}_{B_i} + b\hat{\varpi}_{B_j} &= \frac{(a\hat{\varpi}_{B_i} + b\hat{\varpi}_{B_j}) \star \hat{\varpi}_{B_i}^{-1} \hat{\varpi}_{B_j}^{-1}}{\hat{\varpi}_{B_i}^{-1} \hat{\varpi}_{B_j}^{-1}} \\ &= \frac{a\hat{\varpi}_{B_i} \hat{\varpi}_{B_i}^{-1} \hat{\varpi}_{B_j}^{-1} + b\hat{\varpi}_{B_j} \hat{\varpi}_{B_i}^{-1} \hat{\varpi}_{B_j}^{-1}}{\hat{\varpi}_{B_i}^{-1} \hat{\varpi}_{B_j}^{-1}} \\ &= \hat{\varpi}_{B_i \cup B_j}^{-1} \star \hat{\varpi}_{B_i} \star \hat{\varpi}_{B_j} \end{aligned}$$

As such, we demonstrate that $\hat{\varpi}_{B_i \cup B_j}^{-1} = \frac{a\hat{\varpi}_{B_i} + b\hat{\varpi}_{B_j}}{\hat{\varpi}_{B_i} \star \hat{\varpi}_{B_j}}$.

V. Conclusion and Future Work

In this work, we presented SHoPS, a Set-Homomorphic Proof of Data Possession scheme, supporting the 3 levels of data verification. Indeed, we presented the set-union homomorphism property, which extends malleability to set operations properties. Additionally, our proposal is deliberately designed to support public verifiability and constant communication and storage cost. That is, SHoPS allows an implementation of remote data checking at the three networking interfaces, that ensures the flexibility of SHoPS application and enables fulfilling each verifier request. Our future work consists of proving that SHoPS is resistant to data leakage attacks, while considering either a fraudulent prover or a cheating verifier, on the basis of a Data Possession game. In addition, an experimental study has to be conducted in order to show the feasibility of our proposal and gives support to theoretical performance measurements.

References

[1] G. Ateniese, R. Burns, and et al. Provable data possession at untrusted stores. CCS '07, NY, USA, 2007. ACM.

[2] K. D. Bowers, van Dijk, and et al. How to tell if your cloud files are vulnerable to drive crashes. CCS '11, Chicago, Illinois, USA, 2011.

[3] Y. Dodis and e. a. Vadhan. Proofs of retrievability via hardness amplification. TCC '09, San Francisco, 2009.

[4] B. J. Gantz and D. Reinsel. The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east. *IDC iView*, 2012.

[5] A. Juels and B. S. Kaliski. Pors: proofs of retrievability for large files. Virginia, USA, CCS'07.

[6] T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. CRYPTO '91, London, UK, 1991. Springer-Verlag.

[7] M. van Dijk, A. Juels, and et al. Hourglass schemes: how to prove that cloud files are encrypted. CCS '12, NY, USA, 2012.