SHoPS: Set Homomorphic Proof of data Possession Scheme in cloud storage applications

Nesrine Kaaniche, Maryline Laurent

SHoPS: Set Homomorphic Proof of Data Possession Scheme in Cloud Storage Applications

Nesrine Kaaniche, Maryline Laurent

Institut Mines-Telecom, Telecom SudParis, UMR CNRS 5157 SAMOVAR
e–mail: {Nesrine.Kaaniche, Maryline.Laurent}@telecom-sudparis.eu

***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 paper addresses this security concern, while supporting the verification of several data blocks outsourced across multiple storing nodes. We propose a new 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 more cost effective architectures [9]. As such, recent years have witnessed the trend of leveraging cloud data storage, since it provides efficient remote storage services in a pay per use business model.
However, these promising data storage services bring many challenging design issues, considerably 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 outsourced 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], [2], [10], [3], [7], [6], [15]. These schemes are called *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 paper, we present SHoPS, a novel 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 addresses the issue of provable data possession in cloud storage environments, following three substantial aspects: *security level, public verifiability* and *performances*.
The remainder of this paper is organized as follows. First, Section II describes the state of the art of existing PDP schemes. Then, Section III gives a SHoPS overview and provides the security assumptions. Section IV presents our contribution and Section V gives a brief security analysis. Finally, a performance evaluation of the proposed scheme is

given before concluding in Section VII.

II. Requirement Analysis & Related Work

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 a new aggregate proof scheme, using set-homomorphic properties.

A. Requirement Analysis

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.

B. Related Work

The notion of PDP has been introduced by Ateniese et al. in [1]. That is, the client divides the file data D into blocks and creates one tag for each block b_i as $T_{i,b_i} = (H(W_i)g^{b_i})^d) mod N$, where N is an RSA modulus, g is a public parameter, d is the secret key of the data owner and $H(W_i)$ is a random value. The scheme is efficient as there is no need to retrieve data for the verification of data possession. The main drawbacks are computation complexity due to the usage of RSA numbers and the private verifiability feature. In [2], Ateniese et al. propose a publicly verifiable version, which allows any entity to challenge the cloud server. However, [2] is insecure against replay attacks in dynamic scenarios because of the dependencies of index blocks in proof generation and the lack of homomorphism property in the verification procedure.
Juels et al. [10] introduce a method to detect unauthorized changes of stored data by adding *sentinels* in the original data. Their scheme, called Proof Of Retrievability (POR), does not support public verifiability. In addition, only a fixed number of challenges is allowed. On the basis of [10], Shacham et al. [14] propose an improved scheme to realize public data possession verification based on bilinear signature. However, the number of authentication is proportional to the number of data blocks, and the proposed technique does not prevent from leakage of data blocks.
Recently, in [6], Bowers et al. provide a different formulation of the threats that cloud users face. That is, RAFT proposes an approach confirming data redundancy on storage systems, based on a time measure function. The main disadvantages of this scheme are the communication cost depending on the number of blocks in the challenging request, and the important storage cost. The authors exposed two verification approaches. First, they propose a private verification scheme to check the exactitude of server responses based on a local copy of data. While this option may efficiently work for some scenarios, it is restrictive in many cases as it undermines much of the benefits of cloud outsourcing. Second, to improve storage capacity, they refer to the Merkle Tree signature. Thus, this technique also requires the use of a secret for each data file.
To theoretically evaluate the performances of SHoPS, we compare, in Table I, our protocol with four of the most closely-related schemes [1], [7], [14], [8], to our context, in terms of bandwidth, computation and storage costs.
Table I shows that none of the presented schemes does cover the totality of the fixed requirements, in Section II-A.

Metrics	[1]	[7]	[14]	[8]	SHoPS
Nb. of chall.	fixed	∞	∞	∞	∞
Public verif	Yes	No	Yes	No	Yes
CSP CPU cost	$O(1)$	$O(n)$	$O(n)$	$O(logn)$	$O(logn)$
User CPU cost	$O(1)$	$O(n)$	$O(n)$	$O(logn)$	$O(nlogn)$
Band. cost	$O(1)$	$O(1)$	$O(l)$	$O(l)$	$O(n)$
Storage cost	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$

TABLE I
COMPLEXITY COMPARISON BETWEEN DIFFERENT PDP TECHNIQUES (n IS THE NUMBER OF DATA BLOCKS AND l IS THE NUMBER OF ELEMENTS IN EACH DATA BLOCK)

III. Model Description

A. SHoPS Overview

SHoPS introduces three participating entities: the client, the authorized user and the cloud service provider.

SHoPS considers that each data file is divided into *blocks*, and each block B into q *subblocks*, where q is a system parameter. Each subblock is represented by a single element of the multiplicative group $\mathbb{G}_2$. Our single data block proof scheme is made up of five algorithms, on the basis of two phases. During the first phase, the system initialization procedures are executed. This phase is performed once when the file is uploaded.

- $\texttt{gen} : \{1\}^{\lambda} \rightarrow {\mathcal{K}_{pub}}^2 \times \mathcal{K}_{pr} \times {\mathbb{G}_2}^{2q-1}$ – 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.
- $\texttt{stp} : 2^{\mathcal{M}} \times {\mathbb{G}_2}^q \rightarrow \mathbb{G}_2$ – given a data block $B_i \in \{0,1\}^*$ and the public key pk, the setup algorithm generates the corresponding accumulator $\{B_i, \varpi_i\}$, where $i \in \{1, \cdots, n\}$, and n is the number of blocks of a given data file.

The second phase occurs when the verifier wants to check the authenticity of a given data block file.

- $\texttt{clg} : {\mathbb{Z}_p}^* \times {\mathbb{Z}_p}^* \rightarrow \mathcal{C}$ – this algorithm is computed by the verifier and takes as input the number of data blocks q. It generates a challenge $c \in \mathcal{C}$ consisting on a random block index and the public key element $\hat{pk}$ hidden with a random nonce η as $c = (i, \hat{pk}^{\eta})$.
- $\texttt{prf} : \mathcal{K}_{pub} \times 2^{\mathcal{M}} \times \mathcal{C} \rightarrow \mathcal{P}$ – the $\texttt{prf}$ algorithm computes the server's response $P = (\sigma_1, \sigma_2)$ to a challenge, using the encoded file blocks stored on the server disks.
  In the following, we denote the algorithm that calculates the second element of the proof σ_2, by $\texttt{prf}_2$. That is, we have $P = \texttt{prf}(pk, B, c) = \{\sigma_1, \texttt{prf}_2(B, c)\}$, where B is the related data block and c represents the challenge.
- $\texttt{vrf} : \mathcal{P} \times {\mathcal{K}_{pub}}^2 \rightarrow \{0,1\}$ – a verification function for the cloud server's response P, where 1 denotes *accept*, i.e., the client has successfully verified correct storage by the server. Conversely, 0 denotes *reject*.

The difference between SHoPS and traditional proof schemes is that the generation of the possession proof operates on sets of data blocks in $2^{\mathcal{M}}$, instead of operating in data blocks in $\mathcal{M}$. Our choice is mainly motivated by proofs' authenticity, malleability concerns and energy efficiency while applying proof aggregation, as described in Section II.

For instance, taking advantage of the storage and processing capabilities of the storing nodes, SHoPS saves energy within the CSP by distributing the computation over multiple nodes. In fact, each node provides proofs of local data block sets. This is to make applicable, a resulting proof over sets of data blocks, satisfying several needs, such as, proofs aggregation.

Supporting the public verifiability, SHoPS allows an implementation of remote data checking at the three networking interfaces, namely, the client-CSP interface, the CSP-storing nodes interface and between two CSPs interface. This ensures the flexibility of SHoPS application and enables fulfilling each verifier request. This verifier can be:

(1) a data owner, or an authorized verifier, challenging his provider for a data possession proof. The proof aggregation, presented by the union of several data blocks proofs, is an interesting feature, as it allows a unique verification per file and ensures energy efficiency at the client side.

(2) a CSP gate challenging the storing nodes. As the operations on proof' sets are not limited to the aggregation of proofs, the intersection is important to detect byzantine failures at the storing nodes.

(3) a CSP challenging another CSP, in the case of interleaved clouds. The CSP may ask the hosting cloud to provide an aggregated proof or the intersection between two history log files of complex trade systems. In addition, the subset operator may interest the CSP verifier to check the correctness of replicated data hosted on remote servers. 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 3.1: **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 $2^{\mathcal{M}}$.

– Homomorphism:

$$\texttt{prf}_2(B_i \bullet B_j, c) = \texttt{prf}_2(B_i, c) \odot \texttt{prf}_2(B_j, c) \tag{1}$$

– Correctness:

$$\texttt{vrf}(\texttt{prf}(B_i \bullet B_j, c), pk, \hat{pk}) = \texttt{vrf}(\texttt{prf}(B_i, c), pk, \hat{pk}) \wedge \texttt{vrf}(\texttt{prf}(B_j, c), pk, \hat{pk}) \tag{2}$$

We define SHoPS $= \{\texttt{gen}, \texttt{stp}, \texttt{clg}, \texttt{prf}, \texttt{vrf}, \texttt{agg}\}$, where the algorithm $\texttt{agg} : \mathcal{P} \times \mathcal{P} \rightarrow \mathcal{P}$ returns an aggregate proof as:

$$\texttt{agg}(\texttt{prf}_2(B_i, c); \texttt{prf}_2(B_j, c)) = \texttt{prf}_2(B_i, c) \odot \texttt{prf}_2(B_j, c) \tag{3}$$

B. Complexity Assumptions

Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two cyclic multiplicative groups of the same prime order p. An admissible symmetric pairing function $\hat{e}$ from $\mathbb{G}_1 \times \mathbb{G}_1$ in $\mathbb{G}_2$ has to be bilinear, non degenerate and efficiently computable [13], [12].

q-Diffie Hellman Exponent Problem (q-DHE) – Let $\mathbb{G}$ be a group of a prime order p, and g is a generator of $\mathbb{G}$. The q-DHE problem is, given a tuple of elements $(g, g_1, \cdots, g_q, g_{q+2}, \cdots, g_{2q})$, such that $g_i = g^{\alpha^i}$, where $i \in \{1, \cdots, q, q+2, \cdots, 2q\}$ and $\alpha \xleftarrow{R} \mathbb{Z}_p$, there is no efficient probabilistic algorithm $\mathcal{A}_{qDHE}$ that can compute the missing group element $g_{q+1} = g^{\alpha^{q+1}}$.

Computational Diffie Hellman Assumption (CDH) – Let $\mathbb{G}$ be a group of a prime order p, and g is a generator of $\mathbb{G}$. The

CDH problem is, given the tuple of elements (g, g^a, g^b), where $\{a, b\} \xleftarrow{R} \mathbb{Z}_p$, there is no efficient probabilistic algorithm $\mathcal{A}_{CDH}$ that computes g^{ab}.

In the following, we denote by $\star$ two elements multiplication belonging to a multiplicative group.

IV. SHoPS: A New Set Homomorphic PDP Scheme

SHoPS is based on techniques closely related to the well-known Pederson commitment scheme [11]. 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. This is inspired by the Boneh-Gentry-Waters (BGW) broadcast encryption system [5]. The public key consists on a sequence of group elements $(g, g_1, \cdots, g_q, g_{q+2}, \cdots, g_{2q})$, where $g_i = g^{\alpha^i}$, defined upon the bilinear Diffie-Hellman exponent assumption.

A. Single Data Block SHoPS

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, that are conducted in each of the two aforementioned phases. The `gen` and `stp`

Algorithm 1 `gen` procedure

Input: system security parameter (ξ)
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 $(pk, \hat{pk}, pr, \{g_i\}_{1 \leq i \leq 2q; i \neq q+1})$

are, respectively, presented by Algorithm 2 and Algorithm 1. That is, each set of subblocks $\pi_{i,j}$ of B_i is presented by an accumulator $\varpi_i = \prod_{j=1}^{q} g_{q+1-j}^{\pi_{i,j}\ pr}$.

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

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)

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 4.

$$[\hat{e}(g_k, \varpi_i)\hat{e}(pk, \sigma_2)^{-1}]^{\eta}\hat{e}(g, \sigma_1)^{-1} = 1 \tag{4}$$

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

Lemma 4.1: **Public Single Data Block Verification Correctness** The verification procedure of Equation 4 holds if, and only if the data block file $\hat{B}_i = B_i$.

Proof: Having received (σ_1, σ_2) from the cloud, the verifier first calculates $\hat{e}(pk, \sigma_2)$, using the public key of the data owner pk. Then, he computes $\hat{e}(g_k, \varpi_i)$.
Hereafter, based on the random nonce η, the verifier checks that $[\hat{e}(g_k, \varpi_i)\hat{e}(pk, \sigma_2)^{-1}]^{\eta}$ is equal to $\hat{e}(g, \sigma_1)$ as:

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

$$= [\hat{e}(g^{\alpha^k}, \prod_{j=1}^{q} g_{q+1-j}^{\pi_{i,j}}{}^{s}) \star \hat{e}(g^s, \prod_{j=1;j\neq k}^{q} g_{q+1-j+k}^{\pi_{i,j}})^{-1}]^{\eta}$$

$$= [\hat{e}(g^{\alpha^k}, g^{\sum_{j=1}^{q} \pi_{i,j} * \alpha^{q+1-j}}{}^{s}) \star \hat{e}(g^s, g^{\sum_{j=1;j\neq k}^{q} \pi_{i,j} * \alpha^{q+1-j+k}})^{-1}]^{\eta}$$

$$= [\frac{\hat{e}(g^{\alpha^k}, g^{\sum_{j=1}^{q} \pi_{i,j} * \alpha^{q+1-j}}{}^{s})}{\hat{e}(g^s, g^{\sum_{j=1;j\neq k}^{q} \pi_{i,j} * \alpha^{q+1-j+k}})}]^{\eta}$$

$$= [\frac{\hat{e}(g, g^{s* \sum_{j=1}^{q} \pi_{i,j} * \alpha^{q+1-j+k}})}{\hat{e}(g, g^{s* \sum_{j=1;j\neq k}^{q} \pi_{i,j} * \alpha^{q+1-j+k}})}]^{\eta}$$

$$= [\frac{\hat{e}(g, g_{q+1}{}^{s*\pi_{i,k}} \star g^{s* \sum_{j=1,j\neq k}^{q} \pi_{i,j} * \alpha^{q+1-j+k}})}{\hat{e}(g, g^{s* \sum_{j=1;j\neq k}^{q} \pi_{i,j} * \alpha^{q+1-j+k}})}]^{\eta}$$

$$= [\frac{\hat{e}(g, g^{s* \sum_{j=1,j\neq k}^{q} \pi_{i,j} * \alpha^{q+1-j+k}}) \hat{e}(g, g_{q+1}{}^{s*\pi_{i,k}})}{\hat{e}(g, g^{s* \sum_{j=1;j\neq k}^{q} \pi_{i,j} * \alpha^{q+1-j+k}})}]^{\eta}$$

$$= \hat{e}(g, g_{q+1}{}^{s*\pi_{i,k}})^{\eta}$$

$$= \hat{e}(g, g_{q+1}{}^{s*\eta*\pi_{i,k}}) = \hat{e}(g, \sigma_1)$$

This proves the correctness of the verification step (i.e., $\hat{B}_i = B_i$). The non-singularity of the pairing function allows to state that Equation 4 is true if, and only if $\hat{B}_i = B_i$. ■

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 4, in order to support only two pairing functions computation. As such the private verification of a single data block B_i is as follows:

$$\hat{e}(g_k{}^{\eta}, \varpi_i) \star \hat{e}(g, \sigma_1\sigma_2{}^{s\eta})^{-1} = 1 \tag{5}$$

Similarly, we prove the correctness of private single data block verification.

B. Set-Homomorphic Properties of the proposed Scheme

In this section, we extend the design of the data block elementary checking, 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.
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.

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

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

To this purpose, we first express $\varpi_{B_i \cup B_j}$, using ϖ_{B_i} and ϖ_{B_j}, as follows.

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

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

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:

$$\varpi_{B_i \cup B_j} = lcm(\varpi_{B_i}, \varpi_{B_j}) = \frac{\varpi_{B_i} \star \varpi_{B_j}}{gcd(\varpi_{B_i}, \varpi_{B_j})} \tag{6}$$

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

$$a\varpi_{B_i} + b\varpi_{B_j} = gcd(\varpi_{B_i}, \varpi_{B_j}) \tag{7}$$

As such, using the Equation 6 and Equation 7, we find the lcm of the two data blocks B_i and B_j as follows:

$$\varpi_{B_i \cup B_j} = lcm(\varpi_{B_i}, \varpi_{B_j}) = \frac{\varpi_{B_i} \star \varpi_{B_j}}{a\varpi_{B_i} + b\varpi_{B_j}} \tag{8}$$

Therefore, we obtain the proof of the lemma 4.3. ■

Theorem 4.4: **Set-Homomorphism Property – Union Operator** SHoPS considers the algorithms $\mathtt{clg}, \mathtt{prf}$ and $\mathtt{vrf}$ defined above. Let $\mathtt{agg}$ be the algorithm, presented in Equation 1, such that $\bullet$ is the set union operator, as follows.

$$\mathtt{prf}_2(B_i \bullet B_j, c) = \mathtt{prf}_2(B_i, c) \odot \mathtt{prf}_2(B_j, c) = \mathtt{prf}_2(B_i, c) \star \mathtt{prf}_2(B_j, c)(a * \mathtt{prf}_2(B_i, c) + b * \mathtt{prf}_2(B_j, c))^{-1} \tag{9}$$

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

Proof: We prove that SHoPS fulfills the homomorphism and correctness properties.
– Proof of Homomorphism : We know that $a\mathtt{prf}_2(B_i, c) + b\mathtt{prf}_2(B_j, c) = a\hat{\varpi}_{B_i} + b\hat{\varpi}_{B_j}$. 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 8, we can write that:

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

As such, we demonstrate that $\hat{\varpi}^{-1}_{B_i \cup B_j} = \frac{a\hat{\varpi}_{B_i} + b\hat{\varpi}_{B_j}}{\hat{\varpi}_{B_i} \star \hat{\varpi}_{B_j}}$.
This proves that our framework fulfills the homomorphism property.

– Proof of Correctness: We show that an authorized challenger may check the correctness of two different data blocks B_i and B_j, using an aggregate proof $\texttt{prf}_2(B_i \cup B_j, c)$, based on a challenge $c = (k, \hat{pk}^{\eta})$.
We suppose that $\pi_{i,k} \neq \pi_{j,k}$. That is, as presented in Equation 2, the correctness of SHoPS is that $\texttt{vrf}(\texttt{prf}(pk, B_i \cup B_j), pk, \hat{pk}) = 1$. We have:

$$\hat{e}(g_k, \varpi_{B_i \cup B_j}) \hat{e}(pk, \sigma_{2,B_i \cup B_j})^{-1} = \frac{\hat{e}(g^{\alpha^k}, \varpi_{B_i \cup B_j})}{\hat{e}(g^{pr}, \hat{\varpi}_{B_i \cup B_j})}$$

$$= \frac{\hat{e}(g^{\alpha^k}, \frac{\varpi_{B_i} \varpi_{B_j}}{gcd(\varpi_{B_i}, \varpi_{B_j})})}{\hat{e}(g^s, \frac{\hat{\varpi}_{B_i} \hat{\varpi}_{B_j}}{gcd(\hat{\varpi}_{B_i}, \hat{\varpi}_{B_j})})}$$

$$= \frac{\hat{e}(g, \frac{\prod_{l=1}^{q} {g_{q+1-l+k}}^{s\pi_{i,l}} \prod_{l=1}^{q} {g_{q+1-l+k}}^{s\pi_{j,l}}}{gcd(\varpi_{B_i}, \varpi_{B_j})^{\alpha^k}})}{\hat{e}(g, \frac{\prod_{l=1;l\neq k}^{q} {g_{q+1-l+k}}^{s\pi_{i,l}} \prod_{l=1;l\neq k}^{q} {g_{q+1-l+k}}^{s\pi_{j,l}}}{gcd(\hat{\varpi}_{B_i}, \hat{\varpi}_{B_j})^{s}})}$$

$$= \frac{\hat{e}(g, \frac{{g_{q+1}}^{s\pi_{i,k}} \prod_{l=1;l\neq k}^{q} {g_{q+1-l+k}}^{s\pi_{i,l}} g_{q+1}^{s\pi_{i,k}} \prod_{l=1;l\neq k}^{q} {g_{q+1-l+k}}^{s\pi_{j,l}}}{gcd(\varpi_{B_i}, \varpi_{B_j})^{\alpha^k}})}{\hat{e}(g, \frac{\prod_{l=1;l\neq k}^{q} {g_{q+1-l+k}}^{s\pi_{i,l}} \prod_{l=1;l\neq k}^{q} {g_{q+1-l+k}}^{s\pi_{j,l}}}{gcd(\hat{\varpi}_{B_i}, \hat{\varpi}_{B_j})^{s}})}$$

$$= \frac{\hat{e}(g, \frac{\hat{\varpi}^s_{B_i} \hat{\varpi}^s_{B_i}}{gcd(\varpi_{B_i}, \varpi_{B_j})^{\alpha^k}}) \hat{e}(g, {g_{q+1}}^{s\pi_{i,k}} {g_{q+1}}^{s\pi_{j,k}})}{\hat{e}(g, \frac{\hat{\varpi}^s_{B_i} \hat{\varpi}^s_{B_i}}{gcd(\hat{\varpi}_{B_i}, \hat{\varpi}_{B_j})^{s}})}$$

$$= \hat{e}(g, {g_{q+1}}^{s\pi_{i,k}} {g_{q+1}}^{s\pi_{j,k}})$$

As such, based on the random challenge η, we can write $[\hat{e}(g_k, \varpi_{B_i \cup B_j}) \hat{e}(pk, \sigma_{2,B_i \cup B_j})^{-1}]^{\eta}$ as follows:

$$\begin{aligned} &= [\hat{e}(g, {g_{q+1}}^{s\pi_{i,k}} {g_{q+1}}^{s\pi_{j,k}})]^{\eta} \\ &= \hat{e}(g, {g_{q+1}}^{s\pi_{i,k}\eta} {g_{q+1}}^{s\pi_{j,k}\eta}) \\ &= \hat{e}(g, \sigma_{1,B_i \cup B_j}) \end{aligned}$$

■

2) Set-Inclusion Operator: In this section, we prove that SHoPS is homomorphic with respect to the set-inclusion operator.

Theorem 4.5: **Set-Homomorphism Property – Subset Operator** SHoPS considers the algorithms $\texttt{clg}, \texttt{prf}$ and $\texttt{vrf}$ defined above. Let $\texttt{agg}$ be the algorithm, presented in Equation 1, such that $\bullet$ is the set inclusion operator, as follows.

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

$$\texttt{prf}_2(B_j, c) \star \texttt{prf}_2(B_i, c)^{-1} \tag{10}$$

where B_i and B_j are two data blocks of $\in 2^{\mathcal{M}}$, and $B_i \subset B_j$. We prove the homomorphism and the correctness of SHoPS with respect to the set inclusion operator.

Proof: Let B_i and B_j be two data blocks, where $B_i \subset B_j$, and k is the index challenge sent by the verifier.

– Proof of Homomorphism: We have $\texttt{prf}_2(B_j, c) = \hat{\varpi}_{B_j}$. This can write, where $l \neq k$:

$$\begin{aligned} \hat{\varpi}_{B_j} &= \prod_{\pi_{j,l} \in B_j, l \in [1,q]} g_{q+1-l+k}^{s\pi_{j,l}} \\ &= \prod_{\pi_{j,l} \in B_j \setminus B_i, l \in [1,q]} g_{q+1-l+k}^{s\pi_{j,l}} \prod_{\pi_{i,l} \in B_i, l \in [1,q]} g_{q+1-l+k}^{s\pi_{i,l}} \end{aligned}$$

As such, we show that $\prod_{\pi_{j,l} \in B_j \setminus B_i, l \in [1,q]; l \neq k} g_{q+1-l+k}^{s\pi_{j,l}}$

$$= \prod_{\pi_{j,l} \in B_j, l \in [1,q]; l \neq k} g_{q+1-l+k}^{s\pi_{j,l}} \star \prod_{\pi_{i,l} \in B_i, l \in [1,q]; l \neq k} g_{l-q-1-k}^{-s\pi_{i,l}} \tag{11}$$

Using Equation 11, we demonstrate that SHoPS is homomorphic with respect to the set-inclusion operator:

$$\begin{aligned} \texttt{prf}_2(B_j \setminus B_i, c) &= \prod_{\pi_{j,l} \in B_j \setminus B_i, l \in [1,q]; l \neq k} g_{q+1-l+k}^{\pi_{j,l}} \\ &= \hat{\varpi}_{B_j} \star \hat{\varpi}_{B_i}^{-1} \\ &= \texttt{prf}_2(B_j, c) \star \texttt{prf}_2(B_i, c)^{-1} \end{aligned}$$

–Proof of Correctness : The correctness of SHoPS is that $\texttt{vrf}(pk, \hat{pk}, \texttt{prf}_2(B_j \setminus B_i, c)) = 1$, where $B_i \subset B_j$.

$$\begin{aligned} [\frac{\hat{e}(g_k, \varpi_{B_j \setminus B_i})}{\hat{e}(pk, \sigma_{2,B_j \setminus B_i})}]^{\eta} &= [\frac{\hat{e}(g^{\alpha^k}, \varpi_{B_i}^{-1} \star \varpi_{B_j})}{\hat{e}(g^{pr}, \hat{\varpi}_{B_i}^{-1} \star \hat{\varpi}_{B_j})}]^{\eta} \\ &= [\frac{\hat{e}(g, {\sigma_{1,B_i}}^{\eta^{-1}} {\sigma_{1,B_j}}^{\eta^{-1}} \hat{\varpi}_{B_i}^{-s} \star \hat{\varpi}_{B_j}^{s})}{\hat{e}(g, \hat{\varpi}_{B_i}^{-s} \star \hat{\varpi}_{B_j}^{s})}]^{\eta} \\ &= [\frac{\hat{e}(g, {\sigma_{1,B_i}}^{\eta^{-1}} {\sigma_{1,B_j}}^{\eta^{-1}}) \hat{e}(g, \hat{\varpi}_{B_i}^{-s} \star \hat{\varpi}_{B_j}^{s})}{\hat{e}(g, \hat{\varpi}_{B_i}^{-s} \star \hat{\varpi}_{B_j}^{s})}]^{\eta} \\ &= \hat{e}(g, {\sigma_{1,B_i}}^{\eta^{-1}} {\sigma_{1,B_j}}^{\eta^{-1}})^{\eta} \\ &= \hat{e}(g, \sigma_{1,B_j \setminus B_i}) \end{aligned}$$

Therefore, we obtain the proof of correctness of Theorem 4.5. ■

3) Set-Intersection Operator: We proved that SHoPS allows the generation of aggregate proofs with respect to the subset and the union operators. That is, we extend our discussion, using the relations between these two set operators. For instance, based on Theorem 4.4 and Theorem 4.5, we demonstrate that SHoPS is set-homomorphic with respect to the intersection operator. Note that the intersection operation between B_i and B_j may be expressed in terms of the union and the set difference operators as follows.

$$B_i \cap B_j = (((B_i \cup B_j) \setminus (B_i \setminus B_j)) \setminus (B_j \setminus B_i)) \tag{12}$$

V. Security Discussion

In this section, we present a brief security discussion of SHoPS, based on two different threat models.

A. Threat Model

For our technique to be efficient in cloud storage applications, we have to consider realistic threat models. We first point out the case of a *lazy* cloud service provider. In such cases, the storage server wants to reduce its

resources consumption. That is, this *lazy* server claims doing the requested computations to provide responses to the challenger. Second, we consider the case of a malicious verifier that intends to get information about the outsourced data of the data owner. The fact that the verification process can be performed using public elements makes it possible for malicious clients to gain information about files stored on the untrusted servers.

B. Security and Privacy Discussion

We describe the security of SHoPS, using a game that captures the data correctness property. In fact, this game consists in a *lazy* storage server, as an adversary, that attempts to construct a valid proof without processing the original data block as follows. When the verifier wants to check the server's possession of a data block, he sends a random query (c_g, k_g) to the adversary.

- *Challenge* – the verifier requests the adversary to provide a valid proof of the requested data block, determined by a random challenge c_g and the index k_g.
- *ForgeProof* – without processing on the original data file, the adversary tries to compute a proof (σ_1^*, σ_2^*), the challenge c_g, and the public credentials $params$.

The adversary wins the data possession game, if the `vrf` procedure returns 1.

According to the standard definition of proof systems, SHoPS has to fulfill two security requirements: completeness and soundness of verification.

– *Soundness of Verification* – The soundness means that it is infeasible to confound the verifier to accept false proofs (σ_1^*, σ_2^*). That is, even if a collusion is attempted, the CSP cannot prove its possession.

The soundness of our proposition is relatively close to the *Data Possession Game*. Hence, the soundness meets the correctness of verification (Equation 4 and Equation 5), while considering the non singularity of the pairing functions. This property prevents from forging the soundness of verification of our protocol. In order to prove the nonexistence of a fraudulent server prover, we assume that there is a knowledge extractor algorithm Ψ, which gets the public parameters as input, and then attempts to break the CDH assumption in $\mathbb{G}_2$. The Ψ algorithm interacts as follows:

Learning 1– the first learning only relies on the data owner public key $pk = g^{sk}$ as input. Ψ tries to get knowledge of the client secret key sk. That is, the extractor algorithm Ψ picks at random $r_i \in_R [0,\ R[$, where $i \in \mathbb{Z}_p$ and computes g^{r_i}. For each r_i, Ψ checks whether the comparison holds between pk and g^{r_i}. Based on our assumption, Ψ cannot extract the secret key of the client with noticeable probability.

Learning 2– the input of the second learning is the tuple $(pk, \hat{pk}, g)$. The algorithm attempts to extract the secret key sk by performing following steps:

1) $\hat{e}(pk, \hat{pk}) = \hat{e}(g^s, {g_{q+1}}^s) = {g_{q+1}}^{s^2}$
2) $\hat{e}(g, pk) = \hat{e}(g, g^s) = g^s$
3) $\hat{e}(g, {g_{q+1}}^s) = {g_{q+1}}^s$

This learning cannot hold, because of the DDH assumption. In [4], Boneh demonstrates that the DDH assumption is far stronger than the CDH.

– *Completeness of Verification* – In our scheme, the completeness property implies public verifiability property, which allows any entity, not just the client (data owner), to challenge the cloud server for data possession or data integrity without the need for any secret information. That is, public verification elements, needed in the verification process are publicly known. Thereby, any authorized user may challenge the server storage and efficiently verifies the proof of data possession. Hence, SHoPS is a public verifiable protocol.

Lemma 5.1: **Completeness of verification** Given the tuple of public elements $(pk, \hat{pk}, \sigma_1, \sigma_2, params)$ and $B_i = \hat{B}_i$, the completeness of verification condition implies that Equation 4 holds in $\mathbb{G}_2$.

C. Resistance to attacks

In the following analysis, we discuss the resistance of SHoPS to data leakage attacks, when only considering the vulnerabilities over the data file. As such, we suppose a malicious verifier. He attempts to gain knowledge about the outsourced data, based on the public elements of the data owner and multiple interactions with the legitimate storage server. We suppose that the challenger is not considered to perform preservation of computation resources by reusing the same random challenge η from one possession proof session to another. The verifier is assumed to renew the random scalar η to calculate the challenge $\hat{pk}^{\eta}$ for each session.

We suppose that the goal of the fraudulent verifier is to obtain information about the outsourced data file. That is, the attacker may request the same position index challenge k. As such, using two different sessions $((\alpha), (\beta))$, the attacker computes Equation13 and Equation 14 as follows:

$$ {\sigma_1}^{(\alpha)} \star {\sigma_1}^{(\beta)} = \hat{pk}^{\pi_{i,k}\eta_{(\alpha)}} \star \hat{pk}^{\pi_{i,k}\eta_{(\beta)}} = \hat{pk}^{\pi_{i,k}(\eta_{(\alpha)}+\eta_{(\beta)})} \tag{13} $$

$$ {\sigma_1}^{(\alpha)} \star {\sigma_1}^{(\beta)^{-1}} = \hat{pk}^{\pi_{i,k}\eta_{(\alpha)}} \star \hat{pk}^{-\pi_{i,k}\eta_{(\beta)}} = \hat{pk}^{\pi_{i,k}(\eta_{(\alpha)}-\eta_{(\beta)})} \tag{14} $$

Knowing the challenge k, the attacker cannot reconstruct pieces of the file data, based on the CDH assumption. The prover sends only the pair (σ_1, σ_2) to the verifier. Hence, it is likely impossible to extract information $\{\pi_{i,j}\}_{j\in[1,q]}$ from the server response. Thus, the randomness property is also necessary for the non triviality of the proof.

VI. Performance Evaluation

In this section, we present a theoretical performance evaluation in terms of computation, communication and storage costs

A. Computation Cost Evaluation

As presented in Section III, our single data block proof is made up 5 randomized algorithms: `gen`, `stp`, `clg`, `prf` and

vrf. Among these algorithms, gen and stp are performed by the data owner. To generate the public parameters, the client performs $2q + 1$ exponentiations in $\mathbb{G}$. In the stp procedure, this latter executes q exponentiations and multiplications in order to generate the accumulator ϖ, which remains linearly dependent on the data size. Note that, this gen algorithm is one-time cost for the data owner and can be performed apart the other procedures.

For each proof generation, the server computes the accumulator of a given data block with respect to the position index k, and performs q exponentiations and multiplications in order to generate the couple (σ_1, σ_2). Upon receiving the server proof, the verifier conducts 3 pairing computations. In Section II, we presented a brief comparison between SHoPS and the most closely-related schemes ([7], [1], [8], [14]). That is, Table I states the computation cost comparison between our scheme and previous works, at both client and server side.

On the server side, SHoPS distributes the processing overhead over the multiple storing nodes. The cloud gate computes only i multiplications, where i is the number of the requested nodes, in an aggregated proof. Therefore, contrary to the other approaches, SHoPS achieves a $O(logn)$ server computation complexity.

On the verifier side, we brought additional computation cost, in order to perform a public verifiability. That is, the public verification procedure can also be performed by authorized challengers without the participation of the data owner. As such, this concern can be handled in practical scenarios, compared to the private scheme ([7], [8]) which have to centralize all verification tasks to the data owner. In our scheme, the authorized verifier has to generate two random scalars $c \in]0, R[$ and $k \in [1, q]$, in order to conduct his challenge request. Then, he checks the received proof from the cloud server, while performing three pairing computations, regardless the number of data blocks. Thus, the public verifiability introduces a $O(nlogn)$ processing cost at the verifier side.

B. Bandwidth Cost Evaluation

In SHoPS, the bandwidth cost comes from the generated challenge message clg algorithm and the proof response in each verification request. On one hand, the exchanging challenge algorithm consists in transmitting one random position index k, where $k \in_R \mathbb{Z}_q$ and one element $\hat{pk}^{\eta}$. For a recommended security, we consider a security parameter $\lambda = 80$ bits, thus, the total cost of the challenge message is the double size of a group element of a multiplicative $\mathbb{G}$.

On the other hand, the proof response consists only in two elements $(\sigma_1, \sigma_2) \in \mathbb{G}^2$. Therefore, the total bandwidth cost becomes constant and the bandwidth complexity of our scheme is $O(1)$. As shown in Table I, [7] and [14] present $O(l)$ bandwidth complexity, where l is the number of elements in each encoded block of data. As a consequence, the bandwidth cost of these algorithms is linear to l. Considering the number of permitted challenges, [1] suffers from the problem of pre-fixed number of challenges, which is considered as an important requirement to the design of our construction. Nevertheless, their scheme presents a constant bandwidth cost, just like our proposed protocol.

C. Storage Cost Evaluation

On the client side, SHoPS only requires the data owner to keep secret his private key sk. The public elements of a data file consist in the different accumulators of each data block $\varpi_i \{i \in [1, n]\}$, where n is the number of data blocks. Thus, the storage size of each client is $|sk|$. We must note that $|sk|$ is the size of the secret key of a SHoPS client, which is dependent on the security parameter λ. This storage overhead remains acceptable and attractive for resource constrained devices, mainly as it not dependent on the number of data blocks and the size of data.

VII. Conclusion

The growing need for secure cloud storage services and the attractive properties of an interactive proof system, lead us to define an innovative solution for proof of possession.

In this paper, we define SHoPS supporting high security level and low processing complexity. Hence, it is shown to resist to data leakage attacks, while considering either a fraudulent prover or a cheating verifier.

Additionally, our proposal is deliberately designed to support public verifiability and constant communication and storage cost.

References

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

[2] G. Ateniese, R. Burns, and et al. Remote data checking using provable data possession. *ACM Trans. Inf. Syst. Secur.*, 14(1):12:1–12:34, 2011.

[3] G. Ateniese, S. Kamara, and et al. Proofs of storage from homomorphic identification protocols. ASIACRYPT '09, Tokyo, Japan, 2009.

[4] D. Boneh. The decision diffie-hellman problem. ANTS-III, 1998.

[5] D. Boneh, C. Gentry, and et al. Collusion resistant broadcast encryption with short ciphertexts and private keys. CRYPTO'05, Santa Barbara, CA, 2005.

[6] 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.

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

[8] C. Erway and et al. Dynamic provable data possession. CCS '09, Chicago,Illinois, USA, 2009.

[9] 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.

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

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

[12] D. Ratna, B. Rana, and S. Palash. Pairing-based cryptographic protocols : A survey. Cryptology ePrint Archive, Report 2004/064, 2004.

[13] K. W. Regan. Minimum-complexity pairing functions. 45:285 – 295, 1992.

[14] H. Shacham and B. Waters. Compact proofs of retrievability. ASIACRYPT '08, Melbourne, 2008.

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