A Novel Zero-Knowledge Scheme for Proof of Data Possession in Cloud Storage Applications

Nesrine Kaaniche, Ethmane El Moustaine, Maryline Laurent,
Institut Mines-Telecom, Telecom SudParis, UMR CNRS 5157 SAMOVAR
9 rue Charles Fourier, 91011 Evry, France
e–mail: {Nesrine.Kaaniche, Ethmane.Elmoustaine, Maryline.Laurent}@telecom-sudparis.eu

***Abstract*—Recent technological advances have given rise to the popularity and success of cloud storage. However, the prospect of outsourcing an increasing amount of data to a third party and the abstract nature of the cloud foster the proliferation of security and privacy challenges, namely, the remote data possession checking.**

This paper addresses this critical security concern, when storing sensitive data in a cloud storage service, and the need for users to trust commercial cloud providers. It proposes a deterministic Proof of Data Possession (PDP) scheme based on Interactive Proof System (IPS) and an original usage of the GPS scheme. Our approach has several advantages. First, it supports public verifiability which releases data owners from the burden of a periodical verification. Second, it provides constant communication complexity, where the exchanged messages between the storage server and the client are composed of constant number of group elements. Third, our solution is efficient and provably secure, as it is resistant to the fraudulence of the prover and the leakage of verified data.

I. Introduction

Nowadays, the explosive growth of digital contents continues to rise the demand for new storage and network capacities, along with an increasing need for more cost-effective use of storage and network bandwidth for data transfer. The US International Data Corporation (IDC) [13] predicts that more than a zettabyte of data has been produced and replicated, in 2011, growing by a factor of 9 in just five years.

As such, the use of remote storage systems is gaining an expanding interest, namely the cloud storage based services, since it provides cost efficient architectures. These architectures support the transmission, storage, and intensive computation of outsourced data 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. These challenges have significant influence on the security and performances of the cloud system. First, the US Patriot Act [20] gives the government unprecedented access to outsourced data which are either physically hosted in the USA territory or generated by an American actor. This US regulation law threatens the European legislation which has strict privacy laws. That is, it provides governments with expedited access to cloud data. As such, it is important for a cloud user to have the possibility to verify his outsourced data are still hosted in a specific geographic perimeter, supporting the requested legislation.

Second, one of the biggest concerns with cloud data storage is data integrity verification on untrusted servers. In fact, cloud providers generally claim storing data files with redundancy to protect against data loss. Additionally, they often disperse these data across multiple storage placements. Such distribution provides resilience against hardware. Nonetheless, in order to reduce operating costs and save storage capacities, dishonest providers might intentionally neglect these replication procedures, resulting into irrecoverable data errors or even data loss. More seriously, many byzantine attacks may give rise to data leakage attacks.

Hence, a cloud customer should have an efficient way to perform periodical remote integrity verifications, without keeping the data locally. This customer's concern is magnified by his constrained storage and computation capabilities and the large size of outsourced data.

Recently, to mitigate these concerns, many efforts have been proposed under different systems and security models [2]–[4], [10], [17], [18], [23]. These schemes are called *Provable Data Possession* PDP schemes. They ensure integrity verifications of stored data on untrusted remote servers, and are designed to guarantee several requirements, namely lightweight and robust verification, computation efficiency and constant communication cost, based on different security assumptions. These PDP techniques are widely analyzed into two categories, according to the role of the verifier in the model: 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.

Even though existing PDP schemes have addressed various security properties, we still need a careful consideration of potential attacks, namely data leakage attacks, that may cause potential risks for privacy preservation. To design an effective security model, it is important to analyze the PDP scheme under the framework of Zero-Knowledge Proof System (ZKPS), while considering an Interactive Proof System (IPS) between the client and the cloud cluster in a requested geographic perimeter [15].

As such, in the key role of public verifiability and the privacy preservation support, this work addresses the issue of provable data possession in cloud storage environments, following three substantial aspects: *security level, public verifiability,* and *performance*. This paper proposes an efficient verification framework based on a fundamental arithmetic Euclidean Division, adapted to limited storage capacities. The

framework is demonstrated to be resistant against data privacy leakage within a ZKPS. Introduction of a probabilistic method helps to reduce its computation and communication overheads.

The remainder of this paper is organized as follows. First, Section II describes the state of the art of existing PDP schemes, introducing the general concept of these schemes and highlighting their limitations and their security challenges. Then, Section III provides a detailed analysis of the security requirements needed to ensure a secure, scalable and dynamic PDP scheme with public verifiability. Section IV presents our contribution and Section V gives a security analysis. Finally, a performance evaluation of the proposed scheme is given before concluding in Section VII.

II. Requirement Analysis and 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. A PDP scheme consists of four procedures: pre-process, challenge, proof, verification. For building meta-data of a file, the client runs the *pre-processing* procedure. In most of the cases, the client keeps the meta-data secret and sends a version of the data file to the cloud server (e.g., encrypted data, error coding, embedded watermark). To check the possession of the data file, the client sends a randomized challenge to the server for a proof of a specified file data. In response, the server generates the proof. This computation requires the possession of the original data and is based on the received challenge to avoid the replay attacks. Once received, the client compares the proof with the locally stored meta-data.

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. In practice, many sophisticated protocols have been proposed in the literature to address PDP. This section highlights the requirements that should be fulfilled by a PDP scheme and provides an overview on the approaches that have been proposed in the literature to address the verification of the authenticity of data stored on untrusted servers.

A. Requirement Analysis

III The design of our protocol is motivated by providing support of both robustness and efficiency, while considering the limited storage and processing resources of user devices. It has to fulfill the following requirements:

- **Public verifiability:** the public data possession verification is an important requirement, permitting any authorized entity to verify the correctness of outsourced data. Thus, the data owner can be relieved from the burden of storage and computation.
- **Stateless verification:** proofs should be generated according to a randomly produced challenge. Thus, stateless verification requires the use of unpredictable values.
- **Low computation overhead:** on one hand, for scalability reasons, the amount of computation at the cloud storage server should be also minimized, as the server may be involved in concurrent interactions. On the other hand, the proposed algorithms should also have low processing complexity, at the client side.
- **Low communication overhead:** an efficient PDP should minimize the usage of bandwidth, relying on low communication cost.
- **Low storage cost:** the limited storage capacities of the user devices has a critical importance in designing our solution. So, low storage cost at the client side is highly recommended.
- **Unlimited challenges:** the number of challenges should be unlimited. This condition is considered as important to the efficiency of a PDP scheme.

B. Related Work

The notion of PDP has first been introduced by Ateniese et al. in [3]. That is, the client divides the file data D into blocks and creates a cryptographic tag for each block b_i as $T_{i,b} = (H(W_i)g^{b_i})^d)modN$, 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 blocks for the verification of data possession. The main drawbacks are computation complexity due to the usage of RSA numbers and the private verifiability with the secret key of the data owner. 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 loss of homomorphism property in the verification procedure.

Juels et al. [17] introduce a method to detect unauthorized changes of stored data by randomly 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 [17], Shacham et al. [19] propose an improved scheme to realize public data possession verification based on bilinear signature. However, the number of authentication tokens stored on the server is proportional to the number of data blocks, and the proposed technique does not prevent from data blocks leakage.

Recently, Xu et al. [23] propose a new concept to prove the server data possession. That is, the client creates tags as polynomials and considers the file blocks as coefficients to polynomials. The proof procedure is based on polynomial commitment and uses evaluation in the exponential instead of bilinear maps. This idea has also been adopted by [18], based on Lagrangian interpolation.

In [9], Bowers et al. explore new economic security models for cloud services. They 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 disadvantage of this scheme is the communication cost which depends on the number of blocks in the challenging request, and the storage cost prohibitively important. In fact, the authors exposed two verification approaches. First, they propose a private verification algorithm to check the exactitude of server's responses based on a local copy stored by the data owner. While this option may efficiently work for some scenarios, it is too much restrictive in many other cases as it undermines much of the benefits of cloud outsourcing. Second, in order to improve storage capacity consumption, they refer to the Merkle Tree signature. Thus, this technique also requires the use of a secret for each outsourced data file.

Considering other challenging concerns to provide remote proof verifications, [21] aims to prove correct data encryption at rest by imposing a time basis protocol. An issue arising in the design of this hourglass protocol is the fact that the client needs an authentic version of the outsourced data file, to verify responses from the server. However, the client's storage needs should be of constant size, otherwise the benefits of data outsourcing decrease. In order to optimize storage cost at the client side, [21] proposes to use additional MACs or merkle tree processes at the client side. This necessitates that the client retrieves the integrity checks during the challenge response protocol which raises the bandwidth consumption. In addition, the verifier must keep a secret for each outsourced data (if MACs are used) or the root of the hash tree.

On the basis of [22], Willians and Sion propose SR-ORAM scheme. It allows a client hiding its data access pattern from an untrusted cloud server in a single round protocol. However, this scheme requires a poly-logarithmic storage cost and does not support public sharing verification.

To evaluate the objectives given in Section III, we compare, in Table I, our proposed protocol with some existing techniques. On the basis of the requirements of a data possession proof system, we choose four different PDP schemes ([3], [10], [11], [19]), that are most closely-related to our context.

Metrics	[3]	[10]	[19]	[11]	prop.
Nb. of chall.	fixed	∞	∞	∞	∞
Public verif	Yes	No	Yes	No	Yes
CSP cmp. cost	$O(1)$	$O(n)$	$O(n)$	$O(logn)$	$O(logn)$
User cmp. cost	$O(1)$	$O(n)$	$O(n)$	$O(logn)$	$O(logn)$
Band. cost	$O(1)$	$O(1)$	$O(l)$	$O(l)$	$O(1)$
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 block data)

Table I shows that none of the presented schemes ([3], [10], [11], [19]) does cover the totality of the fixed requirements, in Section III. In addition, we must notice that these schemes studied the security aspects in a theoretical framework and overlook the issues related to constrained resources user devices.

In this paper, we design a new Zero Knowledge PDP protocol that supports the fixed requirements, and we provide experimental results to highlight the advantage of the application of our scheme in terms of processing, storage and communication overhead.

III. Model Description

In this paper, we present a novel PDP model based on the well-known GPS scheme proposed by Girault et al. in [14]. GPS is a public-key-based zero knowledge protocol that adapts to resource-constrained devices. Hence, we extend the GPS scheme to the verification of the authenticity of files stored on untrusted servers in cloud platforms. In this section, we describe the components of the model used in this paper to develop our PDP scheme.

A. System Model

We consider three participating entities: the client, the user and the cloud service provider. The client has a collection of data files stored on cloud servers after the pre-processing procedure. The user who shares the stored data with the client may challenge the cloud storage server to provide a proof of possession.

The private verification of our proposal contains five probabilistic algorithms:

- *KeyGen* – given a selected security parameter λ, this algorithm outputs the data owner public and secret keys (pk, sk), where pk is a public elliptic curve point.
- *Setup* – given a data file $D \in \{0,1\}^*$ and the public key pk, the setup algorithm generates the data file identifier ID_F and the corresponding public elements (σ_1, σ_2).
- *GenChal* – this algorithm generates a randomized challenge c.
- *ChalProof* – given the challenge c, and the original version of the file data D, the *ChalProof* produces a proof $P = (y_1, y_2)$.
- *Verify* – given the proof P, the public elements and the private key of the data owner, *Verify* checks the data possession and outputs a result as either accept or reject.

Our idea makes use of Zero Knowledge Proofs (ZKP) to provide a data possession verification. That is, our approach is closely based on techniques related to the GPS scheme [14], which is a public key verification protocol. Furthermore, we propose two variants of proof of possession, supporting public and private verifiability. The private verification makes use of a secret stored locally in the client's device, while the public proof check is based on pairing functions.

The choice for adopting the elliptic variant of GPS scheme is motivated by several reasons. First, ZKP joins randomness into exchanged messages. As such, for each verification session, the prover and the verifier generate new pseudo random values and new composition of the considered file data, thus making messages personalized for each session. Consequently, the randomness involved in the server's responses allows resistance to *data leakage attacks* and preservation of data privacy. Second, the GPS scheme is adapted to the required limited storage capacities on tags. So, from this perspective, with the prevalence of wireless communication, the mobile devices start sharing the benefits of on demand cloud storage services. Due to the resource-constrained devices, our scheme

is based on only one secret which is needed for verification of all outsourced data. In addition, the main advantage of our approach is the public verifiability, preserving the privacy of the outsourced data. That is, an authorized verifier makes only use of public elements and does not request the data owner for extra-computation procedures.

B. Security Model

For our technique to be efficient in cloud storage applications, we have to consider realistic threat models. We first point out the case where an untrusted cloud provider has a malicious behaviour.In such cases, the storage server claims that it possesses the data file, even if the file is totally or partially corrupted. To model this situation, our scheme is based on two important requirements proposed by Shacham [19]. On one hand, there exists no polynomial-time algorithm that can roof the verifier with non-negligible probability. On the other hand, there exists no polynomial-time algorithm that can recover the original data files by carrying out multiple challenge response exchanges. 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 (due to the zero-knowledge property of our scheme) makes it possible for malicious clients to gain information about files stored on the untrusted servers.

The proposed protocol must provide the capabilities to the verifier and the service provider to thwart the two threats mentioned above. To this end, the PDP scheme must enforce a mutual verification of the actions conducted by the client and the storage server.

C. Notations and Assumptions

Our proposal is based on the Elliptic Curve Cryptography (ECC) [16]. To support the public verification, the client must first define a set of *public verification elements* (PVE). The client generates the groups $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ and the pairing function $\hat{e}$ from $\mathbb{G}_1 \times \mathbb{G}_2$ in $\mathbb{G}_T$. $\mathbb{G}_1$ and $\mathbb{G}_2$ are additive subgroups of the group of points of an Elliptic Curve (EC). However, $\mathbb{G}_T$ is a multiplicative subgroup of a finite field. $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ have the same order q. In addition, $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ are generated by P, Q and the generator $g = \hat{e}(P, Q)$, respectively. The bilinear function $\hat{e}$ is derived from the Weil or Tate pairing [7].

Moreover, the PDP scheme proposed in this paper makes use of the two following cryptographic assumptions.

Elliptic Curve Discrete Logarithm Problem (ECDLP) – given an additive group $\mathbb{G}$, a subgroup of $E(\mathbb{F}_p)$, which is generated by the point P of prime order n, it is intractable to find a, where $Q = aP$, and P are known.

Computational Diffie Hellman Problem (CDH) – given a cyclic group $\mathbb{G}$ of order p and generator g, there is no polynomial-time algorithm to calculate g^{ab} , where (g, g^a, g^b) are known.

IV. A New-Zero Knowledge PDP Protocol

In this section, we propose two new PDP schemes for cloud storage application. The first scheme applies when the verification is performed using public credentials while the second scheme restricts the verification process to the owner of the verified data. Both of our schemes rely on zero-knowledge challenge-response protocols. Therefore, they do not add any storage overhead on the client side, which is an important feature for applications where the access to mobile resource-impoverished devices is possible. We provide mathematical proofs of the correctness (i.e., the PDP scheme returns a positive feedback if, and only if the file exists on the server and has not been altered) of the proposed schemes based on the properties of the Euclidean Division (ED) and the bilinear functions.

A. Private Data Possession Scheme

In our scheme, we define an elliptic curve EC over an additive subgroup $\mathbb{G}_1$ of a prime order q. Let P a generator of $\mathbb{G}_1$.

When a client wants to store a file data D on the cloud, he first decomposes D into two blocks s and n. n represents the quotient and s is the remainder applying the Euclidean Division (ED) on the file D with the divisor b. Note that b is kept secret by the client and is used in the decomposition of several outsourced file data. That is, b represents the unique secret information that the client should preserve for all its requests for proof of data possession verification. We must note that b is tightly related to the security of our remote verification scheme. As such, the definition of several data divisors can extend our proposition. That is, the data owner may rely on different secrets with respect to the sensitiveness of the data that he intends to share on the cloud.

Then, with regards to the ECDLP, the published elements are bP, nP, sP, denoted by pk, σ_1, and σ_2, respectively. pk is referred to as the public key while σ_1 and σ_2 are the public elements of the file D.
In the following, we use R, B and K that satisfy the requirements fulfilled in [14]. We also denote by $\cdot$ the scalar point multiplication in an additive group and by $\star$ two elements multiplication belonging to a multiplicative group.

Figure 1 shows the general concept of the private data possession scheme. This scheme consists in two phases. During the first phase, the *keyGen Setup* procedures are executed. This phase is performed only when the file is uploaded on the cloud. The second phase occurs when the client wants to verify the authenticity of the file. To this purpose, it generates a new challenge $chal$ in order to obtain a proof of data possession from the cloud server. This latter runs the *ChalProof* algorithm which is a 3 way procedure. In the following, we provide a detailed description of the steps that are conducted in each of the two aforementioned phases.

Phase I consists of the two following procedures:

- *keyGen* The client public and private key are generated by invoking *keyGen*(1^λ) procedure (cf. Algorithm 1).
- *Setup* When the client wants to store a file data D in the cloud, he runs the *Setup* algorithm, in order to generate

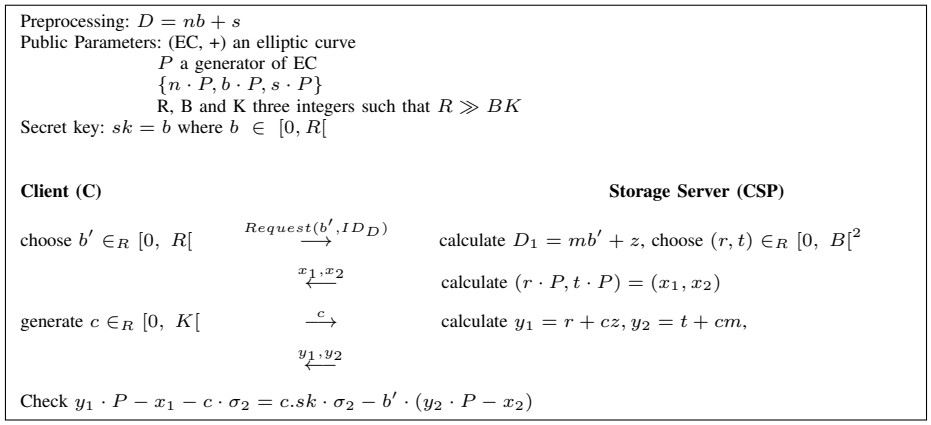

Fig. 1: General Framework of Private Data Possession Scheme

the corresponding public elements (σ_1, σ_2) of D (cf. Algorithm 2).

Algorithm 1 KeyGen Procedure (Private Data Possession)

1: **Input:** System security parameter (λ)
2: **Output:** public key pk and master secret key sk
3: Choose an elliptic curve EC over an additive subgroup $\mathbb{G}_1$ of a prime order n, where *BitLength (n)* $> \xi$ and ECDLP is hard in $\mathbb{G}_1$;
4: Select P a generator of EC;
5: Use a deterministic secure pseudo random number generator (SPRNG) with a random secret *seed* to generate $b \in_R [0, R[$;
6: $sk \leftarrow b$;
7: $pk \leftarrow b \cdot P$;
8: **return** (pk, sk)

Algorithm 2 Setup Procedure (at the data owner side)

1: **Input:** File data (D), pair of secret and public keys (pk, sk) and the point generator P
2: **Output:** File identifier ID_D and the file public elements (σ_1, σ_2)
3: Generate the file identifier ID_D;
4: $ED(D, sk) = (s, n)$; ED: Euclidean Division;
5: $\sigma_1 \leftarrow n \cdot P$;
6: $\sigma_2 \leftarrow s \cdot P$;
7: **return** $(ID_D, \sigma_1, \sigma_2)$

Phase II consists in a challenge-response protocol conducted between the verifier and the storage server. The underlying steps are detailed below.

1) GenChal: The *GenChal procedure* is executed by the client and yields a challenge for the cloud storage server. The client chooses at random a $b' \in_R [0, R[$. In response, the server has to provide a valid new file decomposition using the random divisor b' sent by the client. 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 the secret key of the client sk.

2) ChalProof: The *ChalProof*, executed by the server, has to generate a valid proof of data possession of D. In our construction, the *ChalProof* is a 3 way procedure between the client and the server with a common input (b', P). For the sake of consistency, we suppose that the server possesses a version of the file which is potentially altered. Hereafter, this version is denoted by D_1. The objective of the following steps is to verify whether $D_1 = D$ or not.

- Commitment $(CSP \rightarrow C)$: the storage server calculates the ED of the file using the challenging divisor b' sent by the data owner as:

$$D_1 = mb' + z$$

  Then, he chooses at random two integers $(r, t) \in_R [0, B[^2$ and sends their commitments to the client as $(x_1, x_2) = (r \cdot P, t \cdot P)$.
- Challenge $(C \rightarrow CSP)$: the client chooses a random challenge $c \in_R [0, K[$ and sends it to the server storage, in order to provide the proof. Thereby, the client gets fresh instances indistinguishable from the past results.
- Response $(CSP \rightarrow C)$: the server computes the response as:

$$(y_1, y_2) = (r + cz, t + cm)$$

  Then, the CSP sends (y_1, y_2) to the client.

3) Verify: The client verifies the correctness of the server response. He checks the following equality, using on the secret sk, the divisor b', the challenge c, and the responses (x_1, x_2) and (y_1, y_2) got from the server.

$$y_1 \cdot P - x_1 - c \cdot \sigma_2 = c.sk \cdot \sigma_2 - b' \cdot (y_2 \cdot P - x_2). \quad (1)$$

If the equality holds, the verifier has a proof that the file D exists on the server and that it has not been altered.

Lemma IV.1. Private Verification Correctness *The verification procedure of Equation 1 holds if, and only if the file* $D_1 = D$.

Proof: Having received (y_1, y_2), the client calculates $y_1 \cdot P - x_1 = cz \cdot P$ and $(y_2) \cdot P - x_2 = mb' \cdot P$. Given that

$D = nb+s = mb'+z$, we have $cnb-cmb' = cz-cs$. Taking into consideration that $sk = b$, this writes to the following.

$$c.sk \cdot \sigma_1 - b' \cdot (y_2 \cdot P - x_2) = (y_1 - cs) \cdot P - x_1. \quad (2)$$

This proves the correctness of the verification step (i.e., $D = D_1$). The uniqueness of the quotient and remainder of the ED allows to state that Equation 1 is true if, and only if $D = D_1$. ■

B. Public data possession scheme

An authorized user, different than the client that initially uploaded the file on the storage server, could also verify the authenticity of this file. However, the protocol proposed in the foregoing subsection IV-A cannot be used to this purpose since it supposes that the verifier has the private key of the client, which is not the case of the user. In the following, we demonstrate that the public parameters of the client can also be used in order to implement a PDP scheme between a user, different from the owner of the file, and the storage server.

Thus, procedures presented in Section IV-A cannot apply to the public data possession scenario, as the client uses his secret sk to verify the proof.

As illustrated in Figure 2, the client publishes a set of public verification elements (PVE). As described in Section III-C, these elements are returned by the *KeyGen* procedure as $PVE = \{\mathbb{G}_1, \mathbb{G}_2, P, g, \hat{e}\}$. Note that, for ease of exposition, we used a symmetric pairing function $\hat{e}$. In addition, the client maintains the same public elements, used for the private verification (pk, σ_1, σ_2). The verification condition to state that the file exists on the server and has not been altered is expressed in the following Equation.

$$\hat{e}(c{\cdot}\sigma_1, pk){\star}\hat{e}(c{\cdot}\sigma_2, P) = \hat{e}((y_1{+}b'y_2){\cdot}P, P){\star}\hat{e}(x_1{+}b'x_2, P)^{-1}. \quad (3)$$

Lemma IV.2. Public Verification Correctness *The verification condition of Equation 3 holds if, and only if the file* $D_1 = D$.

Proof: For checking the correctness of the received proof, the authorized user has to verify the equality between the two Equation 3 sides. That is, he has to compare $\hat{e}(c \cdot \sigma_1, pk) \star \hat{e}(c \cdot \sigma_2, P)$ to $\hat{e}((y_1 + b'y_2) \cdot P, P) \star \hat{e}(x_1 + b'x_2, P)^{-1}$.

First, the user executes the following steps, based on the public elements provided by the client:

1) $\hat{e}(c \cdot \sigma_1, pk) = \hat{e}(cn \cdot P, b \cdot P) = \hat{e}(P, P)^{cnb} = g^{cnb}$
2) $\hat{e}(c \cdot \sigma_2, P) = \hat{e}(cs \cdot P, P) = \hat{e}(P, P)^{cs} = g^{cs}$
3) $\hat{e}(P, P)^{cnb} \star \hat{e}(P, P)^{cs} = g^{cnb+cs}$ (4)

We must note that the second side of Equation 3 may be written as follows (cf. Equation 5).

$$\hat{e}((y_1 + b'y_2) \cdot P, P) \star \hat{e}(x_1 + b'x_2, P)^{-1} = \hat{e}(y_1 \cdot P - x_1, P) \star \hat{e}(b' \cdot (y_2 \cdot P - x_2), P) \quad (5)$$

In the sequel, the verifier performs the following steps:

1) $y_1 \cdot P - x_1 = y_1 \cdot P - r \cdot P = cz \cdot P$
2) $b'y_2 \cdot P - b'x_2 = b'y_2 \cdot P - b't \cdot P = (cmb') \cdot P$
3) $\hat{e}(cmb' \cdot P, P) = \hat{e}(P, P)^{cmb'} = g^{cmb'}$
4) $\hat{e}(cz \cdot P, P) = \hat{e}(P, P)^{cz} = g^{cz}$
5) $\hat{e}(P, P)^{cmb'} . \hat{e}(P, P)^{cz} = g^{cmb'+cz}$ (6)

The condition of Equation 3 is equivalent to (4)=(6). Given the uniqueness of the quotient and remainder of ED and the non-singularity of the pairing function, this condition holds if, and only if $D = D_1$. ■

V. Security Analysis

In this security analysis, the cloud service provider is not considered to perform preservation of computation resources by reusing the same pair $(x_1, x_2) = (r \cdot P, t \cdot P)$ from one possession proof session to another. The server is assumed to renew the pair of random numbers r and t and to calculate the elliptic points $x_1 = r \cdot P$ and $x_2 = t \cdot P$ for each session.

In Section V-A, we describe the security of our PDP protocol using a game that captures the data possession property. In fact, this game consists in a fraudulent storage server, as an adversary, that attempts to construct a valid proof without possessing the original data file as follows. When the verifier wants to check the server's possession of data file, he sends a random query b_g to the adversary.

- *ForgeCommit* – without the possession of the data file, the server tries to generate two randoms m^* and z^*, using an iterated hash function. Then, he chooses two integers $(r, t) \in_R [0,\ B[^2$ and sends their commitments to the client as $(x_1, x_2) = (r \cdot P, t \cdot P)$.
- *Challenge* – the verifier requests the adversary to provide a valid proof of the requested file, determined by a random challenge c_g.
- *ForgeProof* – the adversary computes a proof (y_1^*, y_2^*) using his random generation (m^*, z^*) and the challenge c_g.

The adversary wins the data possession game, if the *verify* procedure returns "accept".

In Section V-B, we discuss the resistance of our proposed scheme against a malicious verifier. The verifier attempts to get knowledge about the outsourced data, based on the available public elements and multiple previous exchanges resulting in successful verification sessions with the legitimate storage server.

A. Security and privacy discussion

According to the standard definition of an interactive proof system proposed in [15], our protocol has to guarantee three security requirements: completeness and soundness of verification, and the zero-knowledge property.

1) Soundness of verification: The soundness means that it is infeasible to confound the verifier to accept false proofs (y_1^*, y_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 1 and Equation 3), while

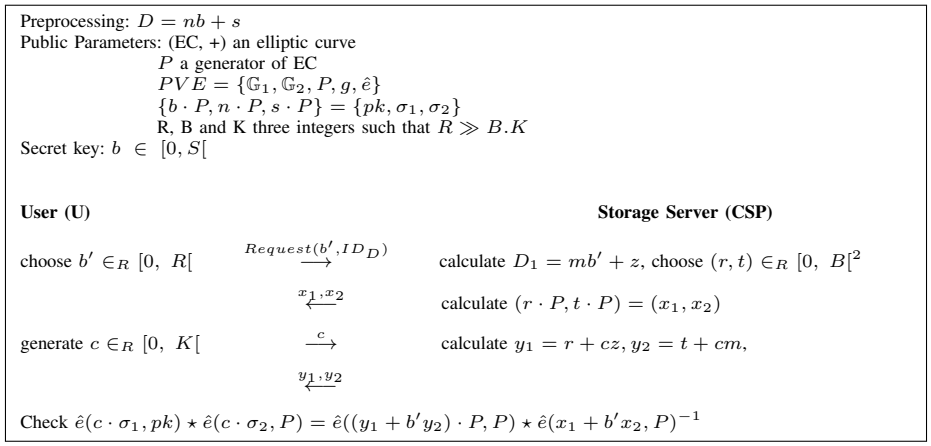

Fig. 2: General Framework of Public Data Possession Scheme

considering the uniqueness of the quotient and remainders of the ED. This property prevents from forging the soundness of verification of our protocol.

In order to the nonexistence of a fraudulent server prover, we assume that there is a knowledge extractor algorithm Ψ ([15]), which gets the public elements as input, and then attempts to break the Elliptic Curve Discrete Logarithm Problem (ECDLP) in $\mathbb{G}$. We state, in Section III, that ECDLP holds in $\mathbb{G}$, if there does not exist a Probabilistic Polynomial Time (PPT) algorithm, with non negligible probability ϵ, that may solve the ECDLP problem.

The Ψ algorithm interacts as follows:

Learning 1– the first learning only relies on the data owner public key $pk = b \cdot P$ 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 $r_i P$. For each r_i, Ψ checks whether the comparison holds between pk and $(r_i \cdot P)$. 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 $(pk, \sigma_1, \sigma_2, PVE)$. The algorithm attempts to extract the secret key sk by performing following steps:

1) $\hat{e}(pk, \sigma_1) = \hat{e}(b \cdot P, n \cdot P) = g^{nb}$
2) $\hat{e}(pk, P) = \hat{e}(b \cdot P, P) = g^{b}$
3) $\hat{e}(n \cdot P, P) = g^{n}$

This learning cannot hold, because of the DDH assumption. In [8], Boneh demonstrates that the DDH assumption is far stronger than the CDH.

2) 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, our proposal is a public verifiable protocol.

Lemma V.1. Completeness of verification *Given the tuple of public elements $(pk, \sigma_1, \sigma_2, PVE)$ and $D = D_1$, the completeness of verification condition implies that Equation 3 holds in $\mathbb{G}_2$.*

Proof: Based on Equation 5 and Equation 6, the completeness of our protocol is performed as follows:

$$\begin{aligned}
\hat{e}(cmb' \cdot P, P) \star \hat{e}(cz \cdot P, P) &= \hat{e}(P, P)^{cmb'} \star \hat{e}(P, P)^{cz} \\
&= g^{cmb'+cz} \\
&= g^{cnb+cs} \\
&= \hat{e}(P, P)^{cnb} \star \hat{e}(P, P)^{cs} \\
&= \hat{e}(cn \cdot P, b \cdot P) \star \hat{e}(cs \cdot P, P) \\
&= \hat{e}(c \cdot \sigma_1, pk) \star \hat{e}(c \cdot \sigma_2, P)
\end{aligned}$$

There exists a trivial solution when $g = 1$. In this case, the above verification could not determine whether the processed file is available, because the equality remains true.
Hence, the completeness of our construction holds, if and only if when $g \neq 1$, giving the uniqueness of the quotient and the non singularity property of the pairing function $\hat{e}$. ■

3) Zero Knowledge property of verification: Compared to original zero-knowledge GPS scheme [14], the prover represents the data storage server and the verifier represents any authorized user. As such, our scheme inherits the zero-knowledge feature from the GPS scheme.

The zero knowledge property ensures the efficiency of a cloud server against malicious attempts to gain knowledge from the outsourced data files. For our construction, this property is achieved thanks to personalized verification session. That is, randomness is required in cloud server's responses, in order to resist to *Data Leakage Attacks* (DLA) and to preserve the confidentiality of data (cf. Section V-B2).

B. Resistance to attacks

In the following analysis, we discuss the resistance of our proposed scheme to classical attacks, when only considering the vulnerabilities over the data file. As such, we suppose a malicious verifier. This latter 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.

1) Resistance to replay attacks: For each verification session, the server storage and the verifier generate new pseudo random values r and t. As presented in [14], the probability of impersonation is $1/K^l$, where l is the number of the protocol rounds, and it depends on the challenge c. That is, a secure pseudo random generator can mitigate to replay attacks. In addition, the public proof of data possession variant of our protocol is secure against MITM attacks. [14] demonstrates that an attacker cannot retrieve the secret key from the exchanged messages between the prover and the verifier.

2) Resistance to DLAs: 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 decomposition of the file by sending the same b' and the same challenge c. As such, using two different sessions $((\alpha),(\beta))$, the attacker computes:

$$2cz = y_1^{(\beta)} + y_1^{(\alpha)} - (r^\beta + r^\alpha)$$

On the other side, the cheating attacker calculates:

$$2cm = y_1^{(\beta)} + y_1^{(\alpha)} - (t^\beta + t^\alpha)$$

Knowing the challenge c, the attacker cannot reconstruct the file data, based on the ECDLP assumption. In fact, the prover sends only the couple $(r \cdot P, t \cdot P)$ to the verifier. As such, it is likely impossible to extract the challenges (r, t) from the server response. Thus, the randomness property is also necessary for the non triviality of the proof system.

VI. Performance Evaluation

In this section, we present a performance evaluation, based on the public verification variant of our construction, in terms of bandwidth, computation and storage costs. In addition, we conduct a number of experiments to evaluate our system performances. As such, we demonstrate that the adopted proof mechanism brings acceptable computation costs.

A. Computation Cost Evaluation

As presented in Section III, our scheme is composed of 5 algorithms: *KeyGen*, *Setup*, *GenChal*, *Chal Proof* and *Verify*. Among these algorithms, *KeyGen* and *Setup* are performed by the data owner. To generate the public key, the client performs one scalar point multiplication in $\mathbb{G}$. In the *Setup* procedure, this latter implements two scalar point multiplication $(n \cdot P, s \cdot P)$ and an Euclidean Division (ED) of the file data, which remains linearly dependent on the data size. Note that, this *Setup* algorithm is one-time cost for the data owner and can be performed apart the other procedures.

For each proof generation, the server applies the ED of the file data, and performs two scalar multiplication (y_1, y_2). Upon receiving the server proof, the verifier conducts 4 pairing computations.

In Section II, we presented a brief comparison between our protocol and the most closely-related schemes ([3], [10], [11], [19]). 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, our construction introduces two scalar additions and two scalar point multiplications, regardless the number of data blocks. Therefore, contrary to the other approaches, our scheme 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 ([10], [11]) which have to centralize all verification tasks to the data owner. In our scheme, the authorized verifier has to generate two random scalars $b' \in [0, R[$ and $c \in [0, K[$, in order to conduct his challenge request. Then, he checks the received proof from the cloud server, while performing four pairing computations and two elements multiplications in a multiplicative group. Thus, the public verifiability introduces a $O(logn)$ processing cost at the verifier side.

B. Bandwidth Cost Evaluation

In our proposed scheme, the bandwidth cost comes from the generated challenge message *GenChal* algorithm and the proof response in each verification request. We neglect the computation cost of algebraic operations because they run fast enough [5], compared with operations in elliptic curve groups and multiplicative groups.

On one hand, the exchanging challenge algorithm consists in transmitting one random element c, where $c \in_R [0,\ K[$ and two elliptic curves points (x_1, x_2). 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 an additive $\mathbb{G}$.

On the other hand, the proof response consists only in two elements $(y_1, y_2) \in \mathbb{Z}^2$. Therefore, the total bandwidth cost becomes constant and the bandwidth complexity of our scheme is $O(1)$.

As shown in Table I, [10] and [19] 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, [3] 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. Based on a private proof, [11] also performs a low bandwidth cost. However, this algorithm supports only private verification. Therefore, along with a public verification, our proposed scheme, allows each verifier to indefinitely challenge the server storage with a constant bandwidth cost.

C. Storage Cost Evaluation

On the client side, our scheme only requires the data owner to keep secret his private key sk, which is a random element $b \in_R [0, R[$, and to store three public elements. These public elements consist in three elliptic curve points $\{pk, \sigma_1, \sigma_2\}$. Thus, the storage size of each client is $3|P|$. We must note that $|P|$ is the size of a group element, 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 contents.

D. Time performance discussion

In this section, we first present the context of the implementation of our proposed scheme. Then, we discuss the computation performances.

1) Context: In an effort to evaluate the performances of our proposal, we build a simulated proof of data possession based on Open Stack Storage system (Swift) [1]. Swift is a cloud based storage system, which stores data and allows write, read, and delete operations on them.

In order to discuss the communication cost and the computation complexity at the client side, we implement several cryptographic operations at the client side of our simulated cloud environment. First, our scheme essentially relies on the Euclidean Division (ED) of the data file and the multiplication of the remainder and the quotient by an elliptic curve point. As such, the effort to evaluate the performance of our solution leads us to study the time performance of scalar and point multiplication operation, at the swift client machine and the computation cost of different symmetric pairing functions. Our tests are conducted in order to understand the execution cost of our proposal on real hardware. That is, on one side, we evaluated the scalar point multiplication durations, of different scalar size of random data for each multiplication. On the other side, we studied the processing cost, due to the computation of bilinear functions, relying on different security levels.
Second, we extend the functions of swift, in order to support our security requirements. That is, we set the security parameter at $\lambda = 80$, and we discuss the communication cost of our proposal, for different content size. For our tests, we use the GNU Multiple arithmetic Precision (GMP) library [12]. We used 1000 samples in order to get our average durations of scalar point multiplication operations, at the swift client machine. That is, we conducted our experiments on an Intel core 2 duo, started on *single mode*, where each core relies on 800 MHz clock frequency (CPU).

2) Implementation Results: Four different scalar sizes and point multiplication are evaluated, in order to present the execution cost of this elementary operation on real hardware. The obtained results are summarized in Table II.

Sec. level (λ) / Scalar size (bits)	80	112	128
10	0.105	4.980	7.123
100	1.813	12.516	28.475
1000	14.522	41.009	79.868
10000	98.001	257.9	677.012

TABLE II: Mathematical operations cost (in ms).

Table II shows that the computation time increases with the scalar size. We must note that the selected scalar presents either the remainder or the quotient, while applying the ED on the outsourced file data.

We also notice that the processing time with large integers is still reasonable. In order to increase the computation performances when larger scalars are needed, our scheme can take advantage of pre-computation tables, by expressing this scalar as a linear decomposition of precomputed scalars used in this table. Therefore, the execution cost of the scalar multiplication becomes much more easier. In addition, Table II shows that the consumed time for multiplication increases, independently from the choice of the scalar size, when we increase the level of security. The latter is recurrent concept in cryptography. It permits to evaluate the hardness of breaking an encryption or a signature algorithm. That is, the longer the level of security is, the harder the cryptanalysis of the algorithm becomes.

In order to show the performances of the public *verify* procedure, we examine the computation duration cost of pairing functions, at the swift client side. That is, for our comparison, we used two symmetric pairing functions from the PBC library [6], including type E pairing *e.param* and type A *a.param*, to examine the impact of different bilinear functions on our proposal, based on three different security levels (cf. Figure 3).

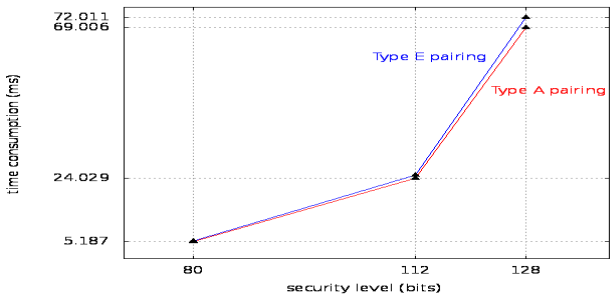

Fig. 3: Computation duration of Type A vs Type E pairing functions (ms)

From Figure 3, we can notice that there is a tiny difference between the two symmetric pairing functions. As such, the type of pairing function should be taken into account, while implementing the proposed procedures. We must note that the type of the pairing function is bound to the choice of the elliptic curve, where the bilinear map is computed.

We also investigate the communication cost of the *GenChal* and the *ChalProof* procedures. For a security parameter λ set to 80, we measure the consumed bandwidth for varying data content size from 1000 bits to 8000 bits (cf. Figure 4). Figure 4 shows that our proposal performs an acceptable bandwidth communication, which remains constant at about 500 bytes for different data sizes.

VII. Conclusion

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

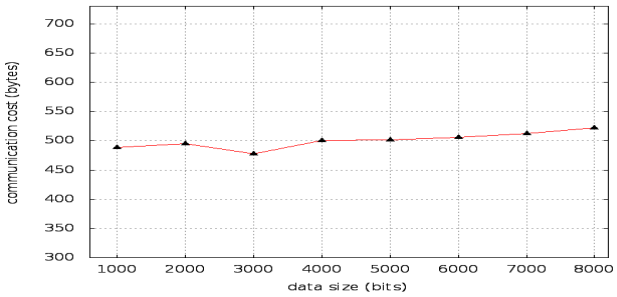

Fig. 4: Communication cost (bytes)

In this paper, we propose a new zero-knowledge PDP protocol that benefits from the elliptic curve variant of the GPS scheme features, namely 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. Thus, we implemented a proof of concept based on the Openstack swift service to demonstrate the feasibility of our proposal and give support to our previous theoretical performance measurements.

Finally, we should state that zero-knowledge PDP schemes present a valuable way to reveal the abstract security assurances sacrificed to cloud based outsourcing issues.

VIII. Acknowledgements

This work is part of ODISEA project and is financially supported by the Conseil Regional d'Ile de France.
The authors would like to thank Prof. Mohamed Hamdi (Higher Communication School, Tunisia) for his helpful suggestions and comments.

References

[1]
[2] G. Ateniese, R. Burns, R. Curtmola, J. Herring, O. Khan, L. Kissner, Z. Peterson, and D. Song. Remote data checking using provable data possession. *ACM Trans. Inf. Syst. Secur.*, 14(1):12:1–12:34, June 2011.
[3] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song. Provable data possession at untrusted stores. In *Proceedings of the 14th ACM conference on Computer and communications security*, CCS '07, pages 598–609, New York, NY, USA, 2007. ACM.
[4] G. Ateniese, S. Kamara, and J. Katz. Proofs of storage from homomorphic identification protocols. In *Proceedings of the 15th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, ASIACRYPT '09, pages 319–333, Berlin, Heidelberg, 2009. Springer-Verlag.
[5] P. S. Barreto, S. D. Galbraith, C. O. Héigeartaigh, and M. Scott. Efficient pairing computation on supersingular abelian varieties. *Des. Codes Cryptography*, 42(3):239–271, Mar. 2007.
[6] L. Ben. On the implementation of pairing-based cryptosystems, 2007.
[7] I. Blake, G. Seroussi, N. Smart, and J. W. S. Cassels. *Advances in Elliptic Curve Cryptography (London Mathematical Society Lecture Note Series)*. Cambridge University Press, New York, NY, USA, 2005.
[8] D. Boneh. The decision diffie-hellman problem. In *Proceedings of the Third International Symposium on Algorithmic Number Theory*, ANTS-III, pages 48–63, London, UK, UK, 1998. Springer-Verlag.
[9] K. D. Bowers, M. van Dijk, A. Juels, A. Oprea, and R. L. Rivest. How to tell if your cloud files are vulnerable to drive crashes. In *Proceedings of the 18th ACM conference on Computer and communications security*, CCS '11, pages 501–514, New York, NY, USA, 2011. ACM.
[10] Y. Dodis, S. Vadhan, and D. Wichs. Proofs of retrievability via hardness amplification. In *Proceedings of the 6th Theory of Cryptography Conference on Theory of Cryptography*, TCC '09, pages 109–127, Berlin, Heidelberg, 2009. Springer-Verlag.
[11] C. Erway, A. Küpçü, C. Papamanthou, and R. Tamassia. Dynamic provable data possession. In *Proceedings of the 16th ACM conference on Computer and communications security*, CCS '09, pages 213–222, New York, NY, USA, 2009. ACM.
[12] T. G. et al. GNU multiple precision arithmetic library 4.1.2, December 2002.
[13] B. J. Gantz and D. Reinsel. Extracting value from chaos state of the universe : An executive summary. *IDC iView*, (June):1–12, 2011.
[14] M. Girault, G. Poupard, and J. Stern. On the fly authentication and signature schemes based on groups of unknown order. *Journal of Cryptology*, 19:463–487, 2006.
[15] O. Goldreich. *Foundations of Cryptography: Basic Tools*. Cambridge University Press, New York, NY, USA, 2000.
[16] D. Hankerson, A. J. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003.
[17] A. Juels and B. S. Kaliski. Pors: proofs of retrievability for large files. In *In CCS 07: Proceedings of the 14th ACM conference on Computer and communications security*, pages 584–597. ACM, 2007.
[18] L. Krzywiecki and M. Kutylowski. Proof of possession for cloud storage via lagrangian interpolation techniques. In *Proceedings of the 6th international conference on Network and System Security*, NSS'12, pages 305–319, Berlin, Heidelberg, 2012. Springer-Verlag.
[19] H. Shacham and B. Waters. Compact proofs of retrievability. In *Proceedings of the 14th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, ASIACRYPT '08, pages 90–107, Berlin, Heidelberg, 2008. Springer-Verlag.
[20] U. States. *A report to Congress in accordance with [section] 326(b) of the Uniting and Strengthening America by Providing Appropriate Tools Required to Intercept and Obstruct Terrorism Act of 2001 (USA PATRIOT ACT) [electronic resource] / submitted by the Department of the Treasury*. Dept. of the Treasury [Washington, D.C.], 2002.
[21] M. van Dijk, A. Juels, A. Oprea, R. L. Rivest, E. Stefanov, and N. Triandopoulos. Hourglass schemes: how to prove that cloud files are encrypted. In *Proceedings of the 2012 ACM conference on Computer and communications security*, CCS '12, pages 265–280, New York, NY, USA, 2012. ACM.
[22] P. Williams and R. Sion. Single round access privacy on outsourced storage. In *Proceedings of the 2012 ACM conference on Computer and communications security*, CCS '12, pages 293–304, New York, NY, USA, 2012. ACM.
[23] J. Xu and E.-C. Chang. Towards efficient proofs of retrievability. In *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '12, pages 79–80, New York, NY, USA, 2012. ACM.