Proof of Possession for Cloud Storage via Lagrangian Interpolation Techniques *

Łukasz Krzywiecki and Mirosław Kutyłowski

Faculty of Fundamental Problems of Technology, Wrocław University of Technology
{firstname.secondname}@pwr.wroc.pl

Abstract. We consider *Provable Data Possesion* (PDP) - a protocol which enables an owner of data stored in the cloud to verify whether this data is still available in the cloud without holding a copy of the file. We propose a PDP framework based on Lagrangian interpolation in the exponent for groups with hard discrete logarithm problem. We reuse properties of this arithmetic exploited so far in broadcast encryption systems.

Keywords: provable data possession, proof of retrievability, cloud computing, Lagrangian interpolation in the exponent, broadcast encryption

1 Introduction

Storing data in a cloud. Recently, storing data in a cloud became a rapidly growing phenomenon. Despite many security problems, it becomes increasingly popular. While there are many reasons for which the users might be tempted to use clouds for storing data, providing guarantees of service quality is a hard challenge.

Clients relying on a remote data storage are interested in data durability and consistency, even if they are not (immediately) interested in the data retrieval. As a minimum condition a data owner should be given an effective way to check that

- his data is still available in the cloud in its original form, and
- once retrieved, the owner can recognize if it has been corrupted by the cloud servers.

Depending on some details a scheme fulfilling these conditions is called *PDP* (*Provable Data Possession*) or *POR* (*Proof of Retrievability*). Designing such schemes is not easy due to the following reasons:

- Typically, the data owner does not keep a copy of data stored in the cloud. So there is no reference data on the owner's side that can be used for a comparison.
- Volume of data exchanged between the user and the cloud should be minimized. In particular, downloading the data from the cloud and integrity checking can be done occasionally, but as a system routine it is practically infeasible.

A naïve solution is that: the owner generates a signature s of payload data D and stores D together with s, in order to check, the owner downloads D and s and verifiers the signature s; so whatever change is done in D the signature will be found invalid. This approach is not much useful - downloading the whole data D for every check is very costly.

* Partially supported by Foundation for Polish Science, programme "MISTRZ"

Provable Data Possession PDP is a two party client-server protocol enabling a client to check whether a file D stored in a cloud on a remote server is available there in the original form. PDP should be efficient: storage overhead and volume of data exchanged between the client and the server should be low. A PDP scheme consists of the following procedures. *Preprocessing* run by a client creates meta-data for a given file D. Generally, the meta-data should have *small volume*, and is to be kept locally by the owner of D. The original *large* file D is transferred to the remote server for a permanent storage, possibly with some meta-data. Checking that D is stored by the server is done according to the following general scenario: In a *challenge phase* the client presents a (randomized) challenge c for a proof that a specific file D is still kept at the server. In *response* the server computes a proof P of possession data of D. Constructing the proof must require possession of the original file D, and should depend on the challenge c (to avoid replay attacks). The proof P is sent back to the client, who verifies it against the meta-data stored locally. In case of POR scheme there is a subsequent retrieval procedure during where the client may distinguish between the original file and a corrupted one retrieved from the cloud server. Perhaps the simplest solution for PDP is based on a cryptographic hash function $\mathcal{H}$. First the client chooses at random challenges $c_1, \ldots, c_k$ and computes corresponding proofs $p_1, \ldots, p_k$, where $p_i = \mathcal{H}(c_i||D)$. During the ith challenge phase the clients sends c_i to the server. The server computes $p'_i = \mathcal{H}(c_i||D)$ and sends p'_i back to the client. If $p'_i = p_i$, then the client assumes that the server holds the original file D. Although the scheme is efficient, it has the obvious drawback that the client can check the server only k times, where k must be fixed during the preprocessing phase.

Previous Work Since in the physical layer removing or destroying data records cannot be prevented by a client of such external systems, a proof of possession becomes very important from the point of view of a data owner. For this reason, PDP/POR techniques were recently analyzed in many papers (see e.g. [1–8, **?**,**?**,12, **?**]). The notion PDP has been introduced by Ateniese et al. in [1]. According to their method, a file is divided into a number of blocks and a cryptographic tag is created for each block. The tag for the ith block m has the form $T_{i,m} = (H(W_i)g^m)^d \bmod N$, where N is and RSA number, d is a secret key of the data owner, and $H(W_i)$ is a pseudorandom value computed with a secret key of the data owner. The scheme from [1] enables to test any number of blocks in one shot without retrieving them. The trick is to compute $T = T_{i_1,m_{i_1}}^{a_1} \cdot \ldots \cdot T_{i_c,m_{i_c}}^{a_c}$ where the block numbers $i_1, \ldots, i_c$ as well as $a_1, \ldots, a_c$ are derived in a pseudorandom way from the challenge, and a sibling value $\rho = H(g_s^{a_1 m_{i_1} + \cdots + a_1 m_{i_1}})$ ($g_s = g^s$ is again a part of the challenge). A verifier can remove the exponent d by raising to power e, and divide the result by all factors $H(W_{i_j})$. Then the result can be raised to power s in order to get $g_s^{a_1 m_{i_1} + \cdots + a_1 m_{i_1}}$ and check it against ρ. The biggest disadvantage of the scheme is usage of RSA numbers; this leads to relatively heavy arithmetic and somewhat long tags, challenge and response values.

A closely related concept - *proof of retrievability (POR)* - has been introduced by Juels and Kaliski in [9]. The scheme is devoted to encrypted files and the main approach is to hide sentinels that can be used to detect file modifications. However, only a fixed number of challenges is allowed and verification has to be done by the data owner.

An important milestone for design of PDP/POR is the paper [10]. It reuses general algorithm design from [1], but the details are much different. Again, the server stores tags that mix information on file blocks with the information that can be retrieved by the data owner. The focus is on the size of the challenge and response data – they are considerably reduced compared to [1]. There are two schemes proposed. The first one is based on pseudo-random functions; its verification procedure requires the secret key of the data owner (so it is not publicly verifiable). However, the main contribution is a rigorous security proof in the standard model. The second scheme is based on BLS signatures and its security is proved under CDH assumption over bilinear groups in the random oracle model. The mechanism of the scheme depends very much on bilinear mappings.

[11] introduces a concept that has been recently substantially improved in [12]. The main idea of [12] is to create tags as commitments to polynomials in an efficient way discovered recently in [13]. Blocks of a file are treated as coefficients to polynomials, the tag for $m_{i,0}, \ldots, m_{i,s-1}$ is created via linear equations:

$$t_i := \mathtt{PRF}_{\mathtt{seed}}(\mathtt{id}, i) + \tau \sum_{j=0}^{s-1} m_{i,j} \cdot \alpha^j \bmod p \tag{1}$$

(τ is a secret value, `id` is the file identifier, `seed` is a secret seed, `PRF` is a pseudo-random function), and α is a fixed parameter, which is represented in the public key by values g^{α^j}. The proof procedure is based on polynomial commitment from [13] and uses evaluation of polynomials in the exponent instead of bilinear mapping. Due to usage of evaluation in the exponent this scheme is related to ours, however this is almost the only common technical element of both schemes.

Our Contribution. We propose a PDP/POR scheme based on techniques closely related to Broadcast Encryption. Apart from the standard goals we aim to fulfill the following conditions:

1. the solution should be based on standard algebraic operations so that it can be used on a wide range of devices,
2. design of the scheme should be as simple as possible and transparent to the users.

These conditions are related to practical implementation issues. For instance, a solution based on exotic algebraic structures is likely to require code that is never used by any other application. This increase implementation cost and reduce usability, especially for resource limited devices (like smart phones and many embedded systems). The second condition is very important for building trust. If the cryptographic mechanism behind the scheme is easy enough so that an average computer professional can understand it and estimate the strength of the proof, we no longer depend on pure trust. Last not least such solutions are better accepted by the users.

Developing a system based on another component (in our case BE) has yet another advantage. First, it might be the case that we re-use a lot of code, and therefore development costs are reduced as well as space requirements for the devices (if some routines serve many purposes). It also reduces effort for product evaluation and audit. Composing a new solution out of known and certified products makes evaluation procedure much easier and more reliable.

We present a scheme based on Lagrangian interpolation in the exponent in a group with hard Discrete Logarithm Problem. It requires neither special properties of RSA arithmetic nor bilinear mappings as the previous techniques getting similar functionalities. Our scheme borrows many ideas from Broadcast Encryption from [14].

2 Preliminaries

Lagrangian Interpolation in The Exponent. Let $L : G \to G$ be a polynomial of degree z, and $A' = \langle (x_0, g^{rL(x_0)}), \ldots, (x_z, g^{rL(x_z)}) \rangle$ be a list of pairs where $x_i \neq x_j$ for $i \neq j$. Then we can reconstruct $g^{rL(x)}$, for an arbitrary x, by *Lagrangian interpolation in the exponent*:

$$LI_{EXP}(x, A') \stackrel{\text{def}}{=} \prod_{i=0,(x_i,.)\in A'}^{z} \left(g^{rL(x_i)}\right)^{\prod_{j=0,j\neq i}^{z} \left(\frac{x-x_j}{x_i-x_j}\right)}.$$

Note that the left right-hand side expression equals

$$g^{r\sum_{i=0,(x_i,.)\in A'}^{z} \left(L(x_i)\prod_{j=0,j\neq i}^{z} \left(\frac{x-x_j}{x_i-x_j}\right)\right)} = g^{rL(x)}.$$

The most useful property of Lagrangian interpolation in the exponent is that given z pairs $(x_1, L(x_1)), \ldots, (x_z, L(x_z))$ and additionally $(x_0, g^{rL(x_0)}), g^r$:

- it is easy to compute $g^{r \cdot L(u)}$ for an arbitrary u,
- it is infeasible to compute the values of the polynomial $\hat{g}^{L(x)}$ for $\hat{g}$ chosen at random, and in particular
- it is infeasible to reconstruct the polynomial L,

provided that group G fulfills standard hardness assumptions for discrete logarithm.

Broadcast Encryption (BE). BE is a scheme in which a single unit, called *broadcaster*, sends data to authorized users via a shared broadcast channel. The broadcast channel is accessible to everyone in its range, so BE has to ensure that:

- a non-authorized user cannot retrieve the data from the message broadcasted,
- each authorized user can retrieve this data,
- the system supports changes of the set of authorized users.

The simplest way to achieve these properties is to broadcast data in an encrypted form, where the decryption key is given to the authorized users only. The hardest design problem of BE is how to change the set of authorized users. Usually we try to encode the new key in a short broadcast message, called *enabling block*, so that only authorized users can derive this key.

Encoding based on Lagrangian interpolation is one of the fundamental techniques for BE revocation [14]. In this system a user i holds its share $(x_i, L(x_i))$ of a secret polynomial L of degree z. The new key K is transmitted in a so called *enabling block*:

$$g^{r \cdot L(0)} \cdot K, g^r, (u_1, g^{r \cdot L(u_1)}), \ldots, (u_z, g^{r \cdot L(u_z)})$$

where $u_1, \ldots, u_z$ are pairwise disjoint. So, holding one additional value $(x_i, L(x_i))$ it is easy to reconstruct $g^{r \cdot L(0)}$ via Lagrangian interpolation in the exponent, and thereby to derive K. On the other hand, if the broadcaster wants to prevent a user i to get K, he can use x_i as one of the values u_j. Then user i has only z values of the polynomial $g^{r \cdot L(x)})$ and therefore is unable to derive it. Moreover, since r is chosen for each enabling block independently at random, the values contained in one enabling block are useless for decrypting the key K from a different enabling block.

3 Outline of the Scheme

In our scheme we use a cyclic group G of a prime order q such that Discrete Logarithm Problem is hard in G. Let g be a generator of G.

A file to be stored in the cloud is divided into *blocks* and each block into z *sublocks*, where z is a scheme parameter. Each subblock is represented by a single element from G. The parameter z is chosen according to operating system requirements (preferably, a block should consist of some number of *pages* in the sense of the operating system).

Single-Block PDP: it is a PDP scheme restricted to a single block. The proofs concern *all* subblocks of a block, however the size of challenges and responses corresponds to the subblock size.

Let a block f consist of subblocks $m_1, \ldots, m_z$. The block f also corresponds to a secret polynomial L_f of degree z known to the data owner but unknown for the cloud. We may assume w.l.o.g. that the subblocks are pairwise different, e.g. each subblock contains its serial number. The tag t_i for m_i is defined as $L_f(m_i)$ and is known to the cloud. Note that the number of values of polynomial L_f given by the tags is insufficient to reconstruct the polynomial L_f – one value is missing for interpolation of L_f.

The verifier requests the cloud to compute the value $g^{rL_f(x_c)}$ when $(x_c, g^{rL_f(0)})$ and g^r are given. The number r is chosen at random, independently for each request. The element x_c is fixed and different from all possible m_i's. The requested value is obtained via Lagrangian interpolation in the exponent, it requires z group exponentiations per block on the server side. At the same time, the cloud server can derive neither $L_f(0)$ nor $L_f(x_c)$ as they are obtained in the exponent and additionally blinded by r.

If at least one m_i is lost, it is impossible to use polynomial interpolation even if $L_f(m_i)$ is available. Moreover, the scheme is strong in the information-theoretic sense: for each potential answer a there is a polynomial L' that satisfies $a = g^{rL'(x_c)}$, $L'(0) = L_f(0)$, as well as $L'(m_j) = L_f(m_j)$ for $j \neq i$, and there is an y such that $L'(y) = L_f(m_i)$. Also, it is infeasible to use values generated for challenge g^r to answer a query with the challenge $g^{r'}$ for $r' \neq r$.

Computational complexity on the server side is z group exponentiations per block. So for verification of k blocks the server has to perform kz group exponentiations.

Table 1 provides a sequence diagram of the scheme. Note that after transferring the tagged block T_f to the cloud server S, the client C can erase f to save the local storage space. Thus C uses only short block identifier $ID(f)$ in subsequent computations. The procedures are described in Definition 1.

Table 1: Sequence diagram of the PDP scheme.

Client C		Server S
setup: $SK_C \longleftarrow \mathsf{Setup}(\xi)$ $T_f \longleftarrow \mathsf{TagBlock}(SK_C, m)$ erases f	$\xrightarrow{T_f}$	stores T_f
challenge: $(K_f, H) \longleftarrow \mathsf{GenChallenge}(SK_C, ID(f))$ store K_f	$\xrightarrow{H, ID(f)}$	locates T_f for $ID(f)$ $P_f \longleftarrow \mathsf{GenProof}(T_f, H)$
$\mathsf{CheckProof}(P_f, K_f)$	$\xleftarrow{P_f}$	

Definition 1. $\mathsf{Single-Block-PDP}$ *scheme is a tuple of procedures* (Setup, TagBlock, GenChallenge, GenProof, CheckProof) *where:*

- *initialization procedure* $\mathsf{Setup}(\xi)$ *is a probabilistic key generation algorithm run by the client. Its input is a security parameter* ξ*. It returns a secret keys* SK_c *of the client.*
- *procedure* $\mathsf{TagBlock}(SK_c, f)$ *is run by the client. It takes as input a secret key* SK_c *and a block* f *and returns* T_f*, a tagged version of* f*.*
- *procedure* $\mathsf{GenChallenge}(SK_c, ID(f))$ *is run by the client in order to generate a challenge for a cloud server holding* T_f*. Its input are the secret key* SK_c *and a block id* $ID(f)$*. It returns a verification value* K_f *and a challenge* H *for the server.*
- *procedure* $\mathsf{GenProof}(T_f, H)$ *is run by the cloud server in order to generate a proof of possession of* f*. Its input are the tagged blocks* T_f *of a block* f *and a challenge* H*. It returns the proof of possession* P_f *for the blocks* T_f*.*
- *procedure* $\mathsf{CheckProof}(P_f, K_f)$ *is run by the client in order to validate the proof of possession. It receives as an input the verification value* K_f *from the local memory of the client and a proof of possession* P_f *from the cloud server. It returns Accept or Reject as evaluation of* P_f*.*

A supplementary procedure $\mathsf{PubChall}(PK_f)$ *can be run by a* public *verifier aiming to check that a cloud server is holding* T_f*. The procedure receives as an input the public key* $PK_f = (K_f, H)$ *which is a chosen output from* GenChallenge *procedure.* PubChall *returns another pair* $(K_{f_{new}}, H_{new})$*, which forms a new verification value and a new challenge for the cloud server holding* T_f*.*

General PDP Scheme: the trick is to use different polynomial L_f for each block f, but with the same value at zero. Then the same challenge $g^{rL_f(0)}, g^r$ can be used for all blocks. The client sends only one challenge altogether with block identifiers to check the possession of them by the server.

The cloud server aggregates the proofs of possession $g^{rL_f(x_c)}$ obtained for different blocks by multiplying them. The client checks that the response equals $(g^r)^{\sum_f L_f(x_c)}$.

Definition 2. $\mathsf{General-PDP-Scheme}$ *consists of procedures* $\mathsf{AggGenChallenge}$ *and* $\mathsf{AggGenProof}$*, where:*

- *the aggregation challenge generating procedure* $\mathsf{AggGenChallenge}(SK_c, F)$ *is run by the client in order to generate an aggregated challenge corresponding to many blocks for a server holding a set. It receives as an input the secret key* SK_c*, and a collection of blocks identifies* $F = \{ID(f_1), \ldots, ID(f_k)\}$*. It returns the verification value* K_f *and the short challenge* H *for the server (*$H = (x_c, g^r, (0, g^{rL(0)}))$ *in our case).*
- *the proof generating procedure* $\mathsf{AggGenProof}(F, H)$ *is run by the server in order to generate an aggregated proof of possession. It receives as an input the list of identifiers* F*, the tagged blocks* T_f *for each* $ID(f) \in F$ *and a challenge* H*. It returns the proof of possession* P_f *for all blocks of* F*.*

It is very important that the data owner does not have to store all polynomials L_f (one polynomial per block). They can be generated in a pseudorandom way from some secret seed chosen by the data owner. Therefore the local storage usage is minimized.

4 PDP via Lagrangian Interpolation

In this section we describe in detail our PDP scheme for a single block. First let us fix some preliminary issues:

- Let p, q be prime numbers, $q|p-1$, and G be a subgroup of order q in $\mathbb{Z}_p^*$, for which Discrete Logarithm Problem (DLP) is hard. Let g be a generator of G.
- C will denote a client, S will denote a cloud server.
- We shall consider storing a block f in a cloud. We assume that f is a tuple of z subblocks: $f = (m_1, \ldots, m_z)$, where each m_i contains its index i. Moreover, each m_i can be regarded as an element of $\mathbb{Z}_q$.
- For $f = (m_1, \ldots, m_z)$, T_f is a tagged version of f. It is a tuple of z pairs: $T_f = ((m_1, t_1) \ldots, (m_z, t_z))$ such that t_i is obtained via $\mathsf{TagBlock}$ described below. We assume that each t_i can be interpreted as an element of $\mathbb{Z}_q$.
- $ID(f)$ is a unique identifier for the block f. Similarly, $ID(S)$ is an identifier for the remote server S.
- $SPRNG(SK, ID(S), i, \mathbb{Z}_q)$ is a secure pseudorandom number generator that generates elements from $\mathbb{Z}_q$ in a way indistinguishable from a uniform random generator. $SK, ID(S), i$ are used as the seed.
- let $a \longleftarrow_R A$ mean that the element a is drawn from A uniformly at random.

4.1 Procedures

Procedure Setup defines system parameters for a user:

Data: system security parameter ξ
Result: group G, the master secret key SK_C of the user
begin
  choose G subgroup of $\mathbb{Z}_p^*$ of order q, such that q, p are prime, $q|p-1$, *BitLength*$(q) > \xi$, and DLP is hard in G
  $SK_C \longleftarrow_R \mathbb{Z}_q$
  return SK_C;

Procedure 1: Setup algorithm $\mathsf{Setup}(\xi)$

Procedure Poly yields a secret polynomial L_f over $\mathbb{Z}_q$ for a given block f, described by the index *ID*(f). The polynomial is derived in a pseudorandom way with *SPRNG* seeded by the secret key of the client, and the block identifier *ID*(f). Thus the client can easily reconstruct L_f without the need to store polynomial L_f or f. L_f must have degree z, where z is the number of subblocks in f.

Data: the secret SK_C, the maximum number of subblocks z, the identifier *ID*(m), secure pseudorandom number generating function *SPRNG*, system security parameter ξ
Result: the polynomial $L_f(x)$
begin
  for $i = 0, \ldots, z$ **do**
    $a_i \longleftarrow$ *SPRNG*$(SK_C, ID(f), i, \mathbb{Z}_q)$
  $L_f(x) \longleftarrow \sum_{i=0}^{z} a_i x^i$
  return $L_f(x)$;

Procedure 2: Polynomial generating sub-procedure Poly

In the above procedure we silently assume that $a_z \neq 0$. If $a_z = 0$, then we run the generator until an element a_z different from 0 is obtained.

Procedure TagBlock is a tag generating procedure performed by the client. After reconstructing the secret block polynomial $L_f(x)$ via Poly sub-procedure, the client computes $t_i = L_f(m_i)$ for each subblock $m_i \in f$:

Data: a block $f = (m_1, \ldots, m_z)$, the secret SK_C generated by Setup Procedure
Result: a tagged block $T_f = ((m_1, t_1), \ldots, (m_z, t_z))$
begin
  $L_f(x) \longleftarrow \mathsf{Poly}(SK_C, z, ID(f), SPRNG)$
  foreach $m_i \in m$ **do**
    $t_i \longleftarrow L_f(m_i)$
  return $T_f = ((m_1, t_1), \ldots, (m_z, t_z))$;

Procedure 3: Tag generating procedure $\mathsf{TagBlock}(SK_c, f)$

Procedure GenChallenge is executed by the client and yields a challenge for the cloud. The client reconstructs polynomial $L_f(x)$ via Poly sub-procedure. The client chooses at random an $r \in \mathbb{Z}_q$. Then, the client chooses at random x_c such that $(x_c \neq 0)$ and $(x_c \neq m_i)$ for each subblock $m_i \in f$. The proof to be returned by the cloud equals to $g^{rL_f(x_c)}$. It is stored locally by the client as K_f. (Storing K_f is only for the sake of efficiency, as it can be recomputed after obtaining the proof from the cloud server S). Finally the user creates the challenge $H = \langle g^r, x_c, g^{rL_f(0)} \rangle$:

Data: the master secret SK_C, the block identifier $ID(f)$
Result: $(K_f, H = \langle g^r, x_c, g^{rL_f(0)} \rangle)$

```
1 begin
2     L_f(x) ⟵ Poly(SK_C, z, ID(f), SPRNG)
3     r ⟵_R Z_q
4     x_c ⟵_R Z_q s.t. (x_c ≠ m_i) for each m_i ∈ f
5     K_f = g^{rL_f(x_c)};
6     H ⟵ ⟨g^r, x_c, g^{rL_f(0)}⟩
7     return (K_f, H)
```

Procedure 4: Generating a challenge via GenChallenge$(SK_c, ID(f))$

Procedure GenProof executed by the cloud server has to generate a proof of possession of f. It takes as an input the challenge $\langle g^r, x_c, g^{rL_f(0)} \rangle$ received from the user. It constructs a set Ψ=$\{(m_i, (g^r)^{t_i}) | i = 1, \ldots, z\}$, from tagged block T_f. Then it constructs interpolation set $\Psi' = \Psi \cup \{(0, g^{rL_f(0)})\}$, interpolates the polynomial L in the exponent at x_c: $P_f = LI_{EXP}(x_c, \Psi')$.

Data: the tagged block T_f, the challenge $H = \langle g^r, x_c, g^{rL_f(0)} \rangle$
Result: the proof P_f

```
1 begin
2     Ψ = ∅
3     foreach (m_i, t_i) ∈ T_f do
4         Ψ ⟵ Ψ ∪ {(m_i, (g^r)^{t_i})}
5     Ψ' ⟵ Ψ ∪ {(0, g^{rL_f(0)})}
6     P_f ⟵ LI_EXP(x_c, Ψ')
7     return P_f;
```

Procedure 5: Proof procedure GenProof(T_f, H)

Procedure CheckProof executed by the user verifies the proof returned by the cloud with the answer K_f retained locally:

Data: P_f returned from the cloud, K_f stored locally
Result: *Accept* if the proof is correct, *Reject* otherwise

```
1 begin
2     if P_f == K_f then
3         return Accept
4     else
5         return Reject
```

Procedure 6: Proof procedure CheckProof(P_f, K_f)

4.2 Supplementary Procedures

Public verification. In this scenario a client publishes a public key PK_f for the block f. The key PK_f, in fact, is a pair (K_f, H) returned by the GenChallenge procedure. PK_f forms an input tuple $\langle g^{rL_f(x_c)}, g^r, x_c, g^{rL_f(0)} \rangle$ for procedure PubChall. This procedure transforms these values by replacing r by $r \cdot r'$ for a random r'. Thereby, the verifier gets fresh instances indistinguishable from the result of GenChallenge. With these values the verifier can proceed in the way already described. Observe that PubChall yields the results with the same probability distribution as GenChallenge.

Data: a public key PK_f for the block f, $PK_f = \left\langle g^{rL_f(0)}, g^r, x_c, g^{rL_f(x_c)} \right\rangle$
Result: $(K_f = (g^{rL_f(x_c)})^{r'}, H = \left\langle (g^r)^{r'}, x_c, (g^{rL_f(0)})^{r'} \right\rangle)$

begin
  $r' \longleftarrow_R \mathbb{Z}_q$
  $K_f \longleftarrow (g^{rL_f(x_c)})^{r'}$
  $H \longleftarrow \left\langle (g^r)^{r'}, x_c, (g^{rL_f(0)})^{r'} \right\rangle$
  return (K_f, H)

Procedure 7: Public challenge generating PubChall

Aggregation of challenges and proofs. Here we present a modified version of algorithms that allow to aggregate challenges and proofs for many blocks (files) stored at server S, as mentioned in Sect. 3 (General PDP Scheme). Now all coefficients but a_0 in polynomials L_f are unique for each block f; while a_0 is the same for all blocks.

Data: the secret SK_C, the maximum number of subblocks z, the identifier *ID*(f), secure pseudorandom number generating function *SPRNG*, system security parameter ξ
Result: the polynomial $L_f(x)$

begin
  $a_0 \longleftarrow$ *SPRNG*$(SK_C, ID(S), 0, \mathbb{Z}_q)$
  for $i = 1, \ldots, z$ **do**
    $a_i \longleftarrow$ *SPRNG*$(SK_C, ID(f), i, \mathbb{Z}_q)$
  $L_f(x) \longleftarrow \sum_{i=0}^{z} a_i x^i$
  return $L_f(x)$;

Procedure 8: Polynomial generating sub-procedure AggPoly

In the modified procedure the challenge and the proof are constructed by interpolating polynomials at $x_c \neq 0$. The value g^{ra_0} is sent to the cloud server as the value $g^{rL_f(0)}$ for every single questioned block. The proof value K_f is now the product of proof values determined separately for questioned blocks.

Data: the master secret SK_C, the block identifiers $F = \{ID(f_1), \ldots, ID(f_k)\}$
Result: $(K_f, H = \left\langle g^r, x_c, g^{rL(0)} \right\rangle)$

begin
  foreach *ID*$(f_i) \in F$ **do**
    $L_{f_i}(x) \longleftarrow$ AggPoly$(SK_C, z, ID(f_i), SPRNG)$
  $r \longleftarrow_R \mathbb{Z}_q$
  $x_c \longleftarrow_R \mathbb{Z}_q$ s.t. $(x_c \neq m)$ for each $m \in f$
  $K_f \longleftarrow g^{rL_{f_1}(x_c)} \cdot \ldots \cdot g^{rL_{f_k}(x_c)}$
  $H \longleftarrow \langle g^r, x_c, g^{rL(0)} \rangle$
  return (K_f, H)

Procedure 9: Challenge generating AggGenChallenge

Data: The list of identifiers F, the tagged blocks T_f for each $ID(f) \in F$, the challenge $\left\langle g^r, x_c, g^{rL(0)} \right\rangle$
Result: The proof P_f

```
1 begin
2     P_f = 1
3     foreach ID(f) ∈ F do
4         Ψ_f = ∅
5         foreach (m_i, t_i) ∈ T_f do
6             Ψ_f ⟵ Ψ_f ∪ {(m_i, (g^r)^{t_i})}
7         Ψ'_f ⟵ Ψ_f ∪ {(0, g^{rL(0)})}
8         P_f ⟵ P_f · LI_EXP(x_c, Ψ'_f)
9     return P;
```

Procedure 10: Proof procedure AggGenProof

5 Security of the Scheme

Due to lack of a space, we provide only a proof for a simplified adversary model. Security of the proposed PDP scheme relies on assumptions: CDH and PRNG, defined formally below. The CDH assumption should allow the client to encode a challenge as $g^{rL_f(0)}$ multiple times for the same $L_f(0)$ but for different parameters r.

Definition 3 (CDH Assumption). *Let $\langle g \rangle$ be a cyclic group generated by element g of order $\operatorname{ord} g = q$. There is no efficient probabilistic algorithm $\mathcal{A}_{\mathrm{CDH}}$ that given (g, g^a, g^b) produces g^{ab}, where a, b are chosen at random from $\mathbb{Z}_q$.*

Definition 4 (PRNG Assumption). *Let SPRNG be a pseudorandom number generator that takes as an input any seed $\{0,1\}^*$ and outputs elements from $\mathbb{Z}_q$. There is no efficient probabilistic algorithm $\mathcal{A}_{PRNG}$ that distinguishes with a non-negligible advantage between two distributions $D_1 = (SPRNG(sk,1), \ldots, SPRNG(sk,k))$ and $D_0 = (r_1, \ldots, r_k)$, where $sk, r_1, \ldots, r_k$ are chosen at random from $\mathbb{Z}_q$. That is, for any efficient probabilistic algorithm $\mathcal{A}_{PRNG}$ the advantage $\mathbf{Adv}(\mathcal{A}_{PRNG})$*

$$\mathbf{Adv}(\mathcal{A}_{PRNG}) = \Pr[\mathcal{A}_{PRNG}(D_1) = 1] - \Pr[\mathcal{A}_{PRNG}(D_0) = 1]$$

is negligible, i.e., $\mathbf{Adv}(\mathcal{A}_{PRNG}) \leq \epsilon_{PRNG}$ for sufficiently small ϵ_{PRNG}.

Let's now analyze security of the PDP scheme from Sect. 4.1.

Definition 5. *We consider the experiment of running a forger algorithm $\mathcal{A}$ by a server S for a tagged block $T_f = ((m_1, t_1), \ldots, (m_z, t_z))$. We consider that server runs correctly through q sessions using the unmodified T_f. Then we assume w.l.o.g. that (m_1, t_1) is erased permanently (the case that only m_1 is erased can be treated similarly). To answer the subsequent challenge S invokes the algorithm $\mathcal{A}$ that takes as an input all previous knowledge S gains (i.e. results of computations and interpolations), and the truncated $T'_f = T_f \backslash \{(m_1, t_1)\}$. Let $I_i = \{(x_c, g^{r_i L_f(x_c)}), (0, g^{r_i L_f(0)}), (m_1, g^{r_i t_1}), \ldots, (m_z, g^{r_i t_z})\}$ denotes results of computations for the ith challenge $H_i = \left\langle g^{r_i}, x_c, g^{r_i L_f(0)} \right\rangle$).*

Experiment $\mathbf{Exp}_{\mathcal{A}}$
  let $f = (m_1, \ldots, m_z)$
  let $T_f = (m_1, t_1), \ldots, (m_z, t_z)$
  For $(i = 1$ to $q)$ collect data I_i, H_i
  Erase (m_1, t_1): $T'_f = T_f \setminus \{(m_1, t_1)\}$
  Let $H = \left\langle g^r, x_c, g^{rL_f(0)} \right\rangle)$
  Run $\mathcal{A}(I_1, H_1, \ldots, I_q, H_q, T'_f, H) \rightarrow P_f$
  if $(P_f == g^{rL_f(x_c)})$ then
    return 1 else return 0

Then we define the advantage $\mathbf{Adv}(\mathcal{A})$ *of the algorithm* $\mathcal{A}$ *in experiment* $\mathbf{Exp}_{\mathcal{A}}$ *as the probability* $\Pr[\mathbf{Exp}_{\mathcal{A}}$ *returns* $1]$.

Theorem 1. $\mathbf{Adv}(\mathcal{A})$ *is negligibly small.*

Proof. According to the framework of security games [15], we construct a sequence of games against the adversary $\mathcal{A}$.
Game 0. We define Game 0 as the original attack game against $\mathcal{A}$ in the experiment $\mathbf{Exp}_{\mathcal{A}}$: Let S_0 be the event that $P_f == g^{rL_f(x_c)}$ in Game 0. Thus we have $\Pr[S_0] = \mathbf{Adv}(\mathcal{A})$.
Game 1. We define Game 1 by a slight modification of Game 0. Namely we replace pseudorandom number generator *SPRNG* in the polynomial L_f setup by a truly random draw. Thus resulting polynomial L is a random polynomial over G. The polynomial L is kept secret by the client. The other procedures utilize this polynomial. The rest of the experiment is unchanged. Let S_1 be the event that $P_f == g^{rL(x_c)}$ in Game 1.

Claim 1. $|\Pr[S_1] - \Pr[S_0]| \leq \epsilon_{PRNG}$, where ϵ_{PRNG} is the advantage of some efficient algorithm distinguishing $SPRNG$ outputs from random choices.

Indeed, for any non negligible difference between Games 0 and 1: $|\Pr[S_1] - \Pr[S_0]|$ the algorithm $\mathcal{A}$ could be used as an distinguisher $\mathcal{D}_{PRNG}$ for *SPRNG*.

Claim 2. $\Pr[S_1] \leq \epsilon_{CDH}$.

Proof of Claim 2. Assume conversely that there exists an algorithm $\mathcal{A}$ that helps to win Game 1 with non negligible probability. Then we use $\mathcal{A}$ to construct an algorithm $\mathcal{A}_{CDH}(g, g^a, g^b))$ which computes g^{ab}, thereby breaking CDH Assumption.

Algorithm $\mathcal{A}_{CDH}(g, g^a, g^b))$
  let $t_0 \longleftarrow_R \mathbb{Z}_q$, use t_0 as $L(0)$
  let $x_c \longleftarrow_R \mathbb{Z}_q$
  use g^a as $g^{L(x_c)}$
  let $f = (m_1, \ldots, m_z)$
  let $T'_f = \{(m_2, t_2), \ldots, (m_z, t_z)\}$
  For $(i = 1$ to $q)$ do:
    $r_i \longleftarrow_R G$
    $A \longleftarrow \{(0, g^{r_i t_0}), (x_c, (g^a)^{r_i}), (m_2, g^{r_i t_2}), \ldots, (m_z, g^{r_i t_z})\}$
    $g^{r_i t_1} \longleftarrow LI_{EXP}(m_1, A)$
    $I_i = \{(x_c, (g^a)^{r_i}), (0, g^{r_i t_0}), (m_1, g^{r_i t_1}), \ldots, (m_z, g^{r_i t_z})\}$
    $H_i = \left\langle g^{r_i}, x_c, g^{r_i t_0} \right\rangle$
  Let $H = \left\langle g^b, x_c, (g^b)^{t_0} \right\rangle$
  Run $\mathcal{A}(I_1, H_1, \ldots, I_q, H_q, T'_f, H) \rightarrow P_f$

return P_f

In the above procedure, the polynomial L is not given explicitly, but $L(0) = t_0$, $L(x_c) = a$ and $L(m_i) = t_i$ for $i = 2, \ldots, z$. Note that this definition cannot be used in the procedure in this form, as the number a is available in the exponent only (and deriving it would mean breaking Discrete Logarithm Problem). Nevertheless, in a tricky way the attacker can construct a history of the protocol that is based on values of L at the points $0, m_2, \ldots, m_z$ and x_c (note that one of these values is available in the exponent only). The problematic case for constructing the history is the value at the point m_1, but the algorithm knows the factors r_i and so the value $g^{r_i L(m_1)}$ can be obtained via Lagrangian interpolation in the exponent.

For the challenge step (when m_1 and $L(m_1)$ are already forgotten), we do not have to derive the values $g^{bL(m_i)}$ as they are not included in the challenge. It is easy to see that if $\mathcal{A}$ works as specified, then for the given inputs it returns $P_f = g^{ab}$.

Combining Claims 1 and 2, we have that $|\Pr[S_0] - \Pr[S_1]| = |\mathbf{Adv}(\mathcal{A}) - \Pr[S_1]| \leq \epsilon_{PRNG}$ and $\Pr[S_1] \leq \epsilon_{CDH}$. Therefore $\mathbf{Adv}(\mathcal{A}) \leq \epsilon_{PRNG} + \epsilon_{CDH}$. □

6 Discussion

Space efficient version. The tags $t_1, \ldots, t_z$ for a block need not to be stored by the cloud. An alternative procedure is that the client determines a secret s_1 such that the cloud derives $t_1, \ldots, t_z$ with a PRNG from a seed consisting of s_1, the file name and the block number. The client derives s_1 from its own secret s_0 and the cloud server name. Additionally, the client determines t_0 with a PRNG from the seed consisting of s_0 and the file name. For computing t_c - which stands for $L(x_c)$ - the client applies Lagrangian interpolation given the pairs $(0, t_0), (m_1, t_1), \ldots, (m_z, t_z)$ and argument x_c. The value g^{t_c} encrypted symmetrically with the private key of the client is the tag of the block to be stored by the cloud server (note that retaining g^{t_c} is necessary, as it cannot be reconstructed by the client after erasing $m_1, \ldots, m_z$ from the local memory).

Performance comparison Below we compare our scheme (denoted by LI) with the scheme from [12] (denoted by CX).

space overhead - cloud server: LI: z group elements per block,
LI modified: 1 group element per block and 1 secret per client,
CX: 1 group element per block and z group elements per client,
space overhead - client: independent of the number of blocks for both LI and CX,
creating tags: no exponentiation for both LI and CX,
challenge & verification: LI: 3 exponentiations per block, CX: 2 exponentiations in total,
creating a proof: LI: z+1 exponentiations per block, CX: z–1 exponentiations in total,

File updates Perhaps the only important disadvantage of scheme [12] is that no data update should be done directly (this problem does not occur for our scheme). Indeed, otherwise equation (1) can be used to derive α: by subtracting we can get rid of the PRF

expression, by dividing such results we get rid of τ as well. However, once α become known to the cloud, then it can modify or erase files but still being able to provide correct proofs of possession.

References

1. Ateniese, G., Burns, R.C., Curtmola, R., Herring, J., Kissner, L., Peterson, Z.N.J., Song, D.X., Song, D.X.: Provable data possession at untrusted stores. In: ACM Conference on Computer and Communications Security. (2007) 598–609
2. Curtmola, R., Khan, O., Burns, R.C., Ateniese, G.: Mr-pdp: Multiple-replica provable data possession. In: ICDCS, IEEE Computer Society (2008) 411–420
3. Chang, E.C., Xu, J.: Remote integrity check with dishonest storage server. In Jajodia, S., López, J., eds.: ESORICS. Volume 5283 of LNCS., Springer (2008) 223–237
4. Ateniese, G., Kamara, S., Katz, J.: Proofs of storage from homomorphic identification protocols. In Matsui, M., ed.: ASIACRYPT. Volume 5912 of LNCS., Springer (2009) 319–333
5. Dodis, Y., Vadhan, S.P., Wichs, D.: Proofs of retrievability via hardness amplification. In Reingold, O., ed.: TCC. Volume 5444 of LNCS., Springer (2009) 109–127
6. Wang, Q., Wang, C., Li, J., Ren, K., Lou, W.: Enabling public verifiability and data dynamics for storage security in cloud computing. In Backes, M., Ning, P., eds.: ESORICS. Volume 5789 of LNCS., Springer (2009) 355–370
7. Erway, C.C., Küpçü, A., Papamanthou, C., Tamassia, R., Tamassia, R.: Dynamic provable data possession. In: ACM Conference on Computer and Communications Security. (2009) 213–222
8. Zhu, Y., Wang, H., Hu, Z., Ahn, G.J., Hu, H., Yau, S.S.: Efficient provable data possession for hybrid clouds. In: Proceedings of the 17th ACM Conference on Computer and Communications Security. CCS '10, New York, NY, USA, ACM (2010) 756–758
9. Juels, A., Kaliski, Jr., B.S.: Pors: proofs of retrievability for large files. In: Proceedings of the 14th ACM conference on Computer and communications security. CCS '07, New York, NY, USA, ACM (2007) 584–597
10. Shacham, H., Waters, B.: Compact proofs of retrievability. In Pieprzyk, J., ed.: ASIACRYPT. Volume 5350 of LNCS., Springer (2008) 90–107
11. Boneh, D., Shen, E., Waters, B.: Strongly unforgeable signatures based on computational Diffie-Hellman. In Yung, M., Dodis, Y., Kiayias, A., Malkin, T., eds.: Public Key Cryptography. Volume 3958 of LNCS., Springer (2006) 229–240
12. Xu, J., Chang, E.C.: Towards efficient provable data possession. IACR Cryptology ePrint Archive **2011** (2011) 574
13. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-size commitments to polynomials and their applications. In Abe, M., ed.: ASIACRYPT. Volume 6477 of LNCS., Springer (2010) 177–194
14. Naor, M., Pinkas, B.: Efficient trace and revoke schemes. In Frankel, Y., ed.: Financial Cryptography. Volume 1962 of LNCS., Springer (2000) 1–20
15. Shoup, V.: Sequences of games: a tool for taming complexity in security proofs (2006). Available from: `http://www.shoup.net/papers/games.pdf`