A Multiple-server Efficient Reusable Proof of Data Possesion from Private Information Retrieval Techniques

Juan Camilo Corena, Anirban Basu, Yuto Nakano, Shinsaku Kiyomoto and Yutaka Miyake
Information Security Group, KDDI R&D Labs Inc. Fujimino, Saitama, Japan

Keywords: Cloud Storage, Proof of Data Possession, Private Information Retrieval.

Abstract: A proof of Data Possession (PDP) allows a client to verify that a remote server is still in possession of a file entrusted to it. One way to design a PDP, is to compute a function depending on a secret and the file. Then, during the verification stage, the client reveals the secret input to the server who recomputes the function and sends the output back to the client. The client can then compare both values to determine if the server is still in possession of the file. The problem with this approach is that once the server knows the secret, it is not useful anymore. In this article, we present two PDP schemes inspired in Multiple-Server Private Information Retrieval (MSPIR) protocols. In a traditional MSPIR protocol, the goal is to retrieve a given block of the file from a group of servers storing identical copies of it, without telling the servers what block was retrieved. In contrast, our goal is to let servers evaluate a function using an input that is not revealed to them. We show that our constructions are secure, practical and that they can complement existing approaches in storage architectures using multiple cloud providers. The amount of transmitted information during the verification stage of the protocols is proportional to the square root of the length of the file.

1 INTRODUCTION

The popularity of cloud-based storage services has fostered the development of primitives to guarantee that the owners of the information can retrieve their data when needed. Given the pay-as-you-go model for storage in cloud providers, it is necessary to perform this task in an efficient way to minimize the cost and resources. Several solutions exist in the literature for this problem, besides the obvious solution of downloading the entire file and performing a computation over it locally. In the literature, the two most relevant solutions for this problem are named Proofs of Data Possession (PDPs) (Ateniese et al., 2007) and Proofs of Retrievability (Shacham and Waters, 2008) (PORs). The difference between these primitives is that even though in both of them blocks of a given file are checked to be stored correctly, in the latter an Erasure Code is applied to guarantee that the file is actually retrievable.

We consider a scenario where there is a set of users U that stores data blocks at a set of remote servers S through a local trustworthy proxy P, similar to an enterprise setting with an in-house proxy, or a website using cloud infrastructure. The requirement for several remote servers (or clouds) comes from a redundancy perspective given that cloud providers also present outages (Raphael, 2013).

A PIR protocol (Chor et al., 1998) allows a client to query a replicated database, in such a way that no server knows what record was retrieved by the client. We wish to apply ideas from MSPIR schemes to the problem of reusing secrets securely in PDP schemes. Even though the idea is very natural, it is usually believed that PIR protocols are too slow to be used in practice (Sion and Carbunar, 2007). For this reason, the approach has not been developed fully and has been deemed only of theoretical interest (Hanser and Slamanig, 2013). However, recent advances in the area (Olumofin and Goldberg, 2012) have made PIR more practical even for the single server scenario.

We show that PIR can be used in a real system in the context of proving data possession, based on the following observations: data storage in cloud servers involves replication, making fast MSPIR schemes such as (Chor et al., 1998) feasible for this scenario. Cloud providers have incentives not to cooperate with each other (e.g. market share). Some efficient PIR schemes can be extended to not just retrieve some blocks form the server, but also to apply a function over the entire file. We present a construction achieving this in Section 4.

Camilo Corena J., Basu A., Nakano Y., Kiyomoto S. and Miyake Y..
A Multiple-server Efficient Reusable Proof of Data Possesion from Private Information Retrieval Techniques.
DOI: 10.5220/0005049803070314
In *Proceedings of the 11th International Conference on Security and Cryptography* (SECRYPT-2014), pages 307-314
ISBN: 978-989-758-045-1

In this work we present two approaches, the first of them is very intuitive: when the proxy has access to a block, it simply stores a hash of the block locally. The verification procedure involves using PIR to download some blocks, in order to verify that their hashes match the ones stored by the proxy. The intuition for security is that by using PIR, the servers have to compute a function involving all the blocks. If a server is storing even a single corrupt block, this will be reflected in the output of the function.

One advantage of this approach is that it requires no additional storage at the server. Processing at the proxy is light since it only involves the computation of a dot product. In addition, it can support dynamic files efficiently. The drawback of this approach is that for the non-retrieved blocks, it does not check that the blocks are those stored by P, but simply that all the servers are storing the same value. It is also possible to reduce the local storage at P by storing hashes of blocks locally with a probability q.

To overcome the previous security drawback, we designed an additional scheme that uses PIR techniques not just for retrieval but also for computing a secret function over the data. The construction can be explained with a toy example: assume we want to perform PIR to recover b_4 over a database $\mathbf{B}$ with five elements b_1, b_2, b_3, b_4, b_5; the database is stored at two servers S_1 and S_2. The client sends to each server the following vectors:

$$\begin{aligned} S_1 &: \mathbf{V}^{[1]} = (-2, 1, -5, 2, -1) \\ S_2 &: \mathbf{V}^{[2]} = (2, -1, 5, -1, 1) \end{aligned} \quad (1)$$

note that $\mathbf{V}^{[1]} + \mathbf{V}^{[2]} = (0,0,0,1,0) = \mathbf{E}^{[4]}$, where $\mathbf{E}^{[i]}$ is a vector consisting of 0s in all coordinates except at coordinate i where it is 1. Now, each server computes the dot product "$\cdot$" between the received vector and its local version of $\mathbf{B}$. Given the properties of the dot product, we have that:

$$\begin{aligned} \mathbf{B}\cdot\mathbf{V}^{[1]} + \mathbf{B}\cdot\mathbf{V}^{[2]} &= \mathbf{B}\cdot(\mathbf{V}^{[1]} + \mathbf{V}^{[2]}) \\ &= \mathbf{B}\cdot\mathbf{E}^{[4]} = b_4. \end{aligned} \quad (2)$$

Therefore, b_4 can be recoverd by adding the response from each server. In this sense, this PIR protocol is computing the dot product of a secret vector $\mathbf{E}^{[i]}$ and the database $\mathbf{B}$. The idea of our PDP is to select a random vector $\mathbf{R}$ and compute $\mathbf{R}\cdot\mathbf{B}$ before uploading the file to the servers. To verify, we select two random vectors such that $\mathbf{V}^{[1]} + \mathbf{V}^{[2]} = \mathbf{R}$. Since each vector $\mathbf{V}^{[i]}$ does not give any information about $\mathbf{R}$, we can verify many times without leaking significant information about our secret vector. Given that all the elements of B are used in the computation, any change or deletion at any of the servers will be detected with high probability. In the current scheme, the client must upload a number of elements equal to the size of the database $|\mathbf{B}|$. However, by representing the database as a square of length $\sqrt{|\mathbf{B}|}$, it is possible to reduce the total transmission to $2|S|\sqrt{|\mathbf{B}|}$, where $|S|$ is the number of servers. The security assumption in the schemes, is that the servers do not communicate among themselves.

1.1 Contributions

The contributions of this work are as follows.

1. We present a novel way to create PDPs, by extending current ideas in multi-server PIR protocols.
2. Our constructions are reusable, can test several servers simultaneously and one of them is very efficient for dynamic files. Even though we are not the first ones to propose a system with these properties (see (Le and Markopoulou, 2012) for a system involving multiple servers), our constructions are simpler and easier to implement for practitioners.
3. We show that our PIR-based constructions are practical given the current trends in remote information storage.

Regarding existing schemes, the drawback of our constructions is that they do not achieve a property called *Public Verifiability*. For this property to hold, anyone should be able to verify the file is stored, regardless of the file's access control policies. Since our proposals may reveal the file contents to the verifier, they should not be used by unauthorized third parties.

The rest of the article is organized as follows: In Section 2 we present the problem scenario; in Section 3 we present existing work related to our proposal; in Section 4 we present our PIR-based constructions and their proof of security; in Section 5 we present the results of the simulation of our proposal and existing constructions; finally, in section 6 we present the conclusions of this work.

2 PROBLEM STATEMENT AND NOTATION

There is a set of users U that connects to a set of remote servers $S = S_1, \ldots, S_s$ through a local proxy P. P forwards all user requests to S and it is assumed to be trustworthy. By trustworthy we mean that the results reported by P about its operations, reflect its view of the system accurately. On the other hand, members of S might want to hide data loss/corruption from members of U. Even though P has storage capabilities,

the amount of storage available at the members of S is significantly larger. We also assume that P is able to perform computations to help members of U to verify that their remote data is stored as intended.

Operations will be performed at S and P in units called blocks. For practical purposes, these blocks are around 4 or 8 KB which is the usual parameter for file systems. It is possible to interact with the storage service using 3 operations namely:

- *write(pos, data, length)*: this operations writes *length* blocks stored in $data$ starting at block *pos*.
- *read(pos)*: reads the block stored at position *pos*.
- *delete(pos)*: deletes the block stored at postion *pos*.

Our goal is to verify the correctness and completeness of the *read*, *write*, *delete* operations in S. By correctness we imply that the information is stored as it was sent by the users. By completeness, we mean that S is returning the requested information in its current state.

The types of attack that can be possible to perform by S are the following:

- A *read* operation returns a random value, or a previous value for the block.
- A *write* operation writes a different data.
- A *delete* operation might not be executed.

We will denote vectors and matrices by bold capital letters (e.g. $\mathbf{V}$, $\mathbf{B}$). Unless otherwise defined, the elements of a given vector will be represented by lower indices, such as $\mathbf{V}_1$. Positions in the matrix will be given by two lower indices enclosed by square brackets and separated by a comma. Thus, $\mathbf{B}_{[k,j]}$ represents position k, j of matrix $\mathbf{B}$. A column j of a matrix will be represented by $\mathbf{B}_{[:,j]}$, conversely the k-th row will be represented by $\mathbf{B}_{[k,:]}$. Upper indices in vectors will be used to denote vectors and matrices that are used or stored by a given server. According to this, $\mathbf{V}^{[i]}$ is a version of vector $\mathbf{V}$ used by server S_i. Similarly, $\mathbf{B}^{[i]}_{[k,j]}$ represents the element k, j of a matrix at server S_i. The need to represent different versions of a given vector arises from possible local variations due tu corruption.

Another use of upper indices is to represent vectors of a given class, such as $\mathbf{E}^{[i]}$ which denotes the i-th row of the identity matrix. The operator $|\mathbf{V}|$ denotes the number of coordinates of a vector. When used on a set (e.g. $|S|$), it denotes the number of elements of the set. Finally, when used on a function, the operator denotes the size of the output of the function.

3 EXISTING WORK

Existing work in this area includes several approaches. On a general perspective there has been significant research on authenticated data structures (Tamassia, 2003). These structures can be used to verify that the elements returned by the remote server contain certain properties, such as being part of a file.

Proofs of Data Possession (PDPs) allow to check a file remotely without downloading it. To this date, many constructions are available, including: trapdoor functions based on discrete logarithms (Ateniese et al., 2007), proofs based on vector operations ans pseudo random functions (Shacham and Waters, 2008), the previous schemes can be set up for public verifiability. Other constructions include: adversarial error correcting codes (Bowers et al., 2009), commitment schemes over linear functions (Xu and Chang, 2012), authenticated encryption (Ateniese et al., 2008) and hardness amplification (Dodis et al., 2009). Other lines of research include: multiuser batch authentication of files (Wang et al., 2010), where a third party can peform tests on behalf of many users simultaneously; audits for dynamic files (Zhu et al., 2013); guaranteeing that multiple encrypted copies can be recovered without additional setup processing (Curtmola et al., 2008); simultaneous public and private verifiability (Hanser and Slamanig, 2013); verification for encoded files (Le and Markopoulou, 2012), (Corena and Ohtsuki, 2013); None of these approaches use PIR techniques to create reusable schemes. Schemes based on Oblivious RAM (ORAM) have also been proposed (Cash et al., 2013), (Apon et al., 2014), their goal is to hide the access pattern of the file, but their overhead is considerable.

A related primitive to proofs of data possession is Private Information Retrieval (PIR) (Chor et al., 1998), where a client wishes to retrieve records from a server without the server knowing what item was retrieved. In particular, we are interested in protocols where there are several servers storing the same database and that are not allowed to communicate among themselves, such as the one from (Chor et al., 1998). This protocol is similar to those described in the introduction.

Efficient single server PIR is also possible. In (Trostle and Parrish, 2011) Trostle and Parrish apply a set of random coefficients to the database to return a single noisy value. The noise can be subtracted in an oblivious way. Single server PIR does not lend itself well for our constructions since the mechanisms used for cancelling the noise works regardless of the correctness of a given block.

4 PROPOSAL

In this section we will present two proposals: the basic one will sample some elements that were stored by P using a PIR protocol. The second one will compute a function over all the blocks.

4.1 Sampling Scheme

The general idea of our construction is to store at P a function of the blocks, and then ask the cloud servers for the blocks to verify them locally. The reason we need PIR to achieve this is twofold. First, PIR protocols apply a function over all the blocks participating in the test. Second, we do not want the cloud servers to know what blocks have been requested by P. Otherwise, a non-persistent attacker that is able to write the same value at a given position for several servers, can reduce the detection capabilities of the scheme. We will now present our approach based on sampling:

Setup: A user sends to P a block $\mathbf{B}$ and a value i denoting the absolute position of this block in the file. P computes $H(s,\mathbf{B}||i)$ and stores it locally. Here, H is a MAC function (e.g. HMAC) using a secret s, over data $\mathbf{B}||i$. The operator $||$ is the concatenation operator. Finally, P uploads $\mathbf{B}$ to S.

Challenge: Each member of S models the L blocks $\mathbf{B}_i$ of the file as a matrix, we call this matrix the *matrix representation of the file*:

$$\mathbf{B} = \begin{bmatrix} \mathbf{B}_1 & \dots & \mathbf{B}_{\sqrt{L}} \\ \vdots & \ddots & \vdots \\ \mathbf{B}_{L-\sqrt{L}+1} & \dots & \mathbf{B}_L \end{bmatrix}. \quad (3)$$

If L is not a square number, then fill the remaining positions of the matrix with a padding scheme over the incomplete columns. Then, P creates $|S|$ random vectors $\mathbf{V}^{[1]},\dots,\mathbf{V}^{[|S|]}$ of length $\sqrt{L}$ such that

$$\sum_{i=1}^{|S|} \mathbf{V}^{[i]} = \mathbf{E}^{[j]} \quad (4)$$

where j is the row of the matrix that wants to be retrieved, and $\mathbf{E}^{[j]}$ is a vector full of 0s except at coordinate j where it is 1. Now P sends $\mathbf{V}^{[i]}$ to $S_i, 1 \le i \le |S|$ and asks them to compute the product of $\mathbf{B}^{[i]}_{[j,k]} = \mathbf{V}^{[i]} \cdot \mathbf{B}_{[:,k]}$ for all the columns $1 \le k \le \sqrt{L}$.

Verification: Once P has received all the responses from each server, it adds them to obtain the returned row j for each column k

$$\mathbf{B}_{[j,k]} = \sum_{i=1}^{|S|} \mathbf{B}^{[i]}_{[j,k]}. \quad (5)$$

For each element $\mathbf{B}_{[j,k]}$, P inverts the mapping function used for the matrix representation, to obtain the real index i' of this block. Next it computes $H(s,\mathbf{B}_{i'}||i')$ and verifies that it has been stored locally. If the previous test does not hold, then we know there is a problem in one of the blocks mapped to the corresponding column in the matrix representation. The column is considered correct otherwise.

Even though it would be tempting not to store the output of H at P for each block, and store it at the servers. That would make the system vulnerable to an attack where a previous version of $\mathbf{B}_{[j,k]}$ is returned by the servers. In such a case, the MAC would verify correctly, without detecting that there is a new version of the block. The purpose of storing the output of H locally, is then to guarantee freshness in the retrieved blocks.

Regarding the detection capabilities of this scheme, it is different from a naive sampling scheme where blocks are retrieved at random without PIR. The reason is that all blocks involved in the proof are included in the computation. If any single part of any block at any of the servers has the wrong value, it will be detected with high probability. In contrast, naive sampling can only detect problems in the blocks that were downloaded by P. We will now formalize this claim

Claim 1. *Our proposed system can detect bit decay or adversarial modification of the file by an adversary who does not corrupt all servers, with significantly higher probability than the naive sampling scheme as the file grows.*

Proof. A scheme that samples random blocks from a file with L blocks, can detect at least one defective block out of d defective blocks with probability:

$$1 - \left(\frac{L-d}{L}\right)^{\tau} \quad (6)$$

where τ is the number of sampled blocks. The expression follows from complementing the probability of always selecting a good element given by $(L-d)/L$ on all the τ tests.

As the file length L grows, the detection probability of the naive sampling approach tends to 0, given that:

$$\lim_{L\to\infty} \left(1 - \left(\frac{L-d}{L}\right)^{\tau}\right) = 0. \quad (7)$$

Now consider our method that retrieves the row j from the file matrix using PIR. Seen from the perspective of a single column k, the block $\mathbf{B}_{[j,k]}$ is retrieved

by computing:

$$\mathbf{B}_{[j,k]} = \mathbf{V}^{[1]} \cdot \mathbf{B}^{[1]}_{[:,k]} + \ldots + \mathbf{V}^{[|S|]} \cdot \mathbf{B}^{[|S|]}_{[:,k]} \quad (8)$$

where $\mathbf{B}^{[i]}_{[:,k]}$ is the k-th column of matrix $\mathbf{B}$ at the i-th server and $\sum_{i=1}^{|S|} \mathbf{V}^{[i]} = \mathbf{E}^{[j]}$. If any element $\mathbf{B}^{[i]}_{[r,k]}$ is different at one of the $|S|$ servers, and assuming that the modified element equals $\mathbf{B}^{[i]}_{[r,k]} = \mathbf{B}_{[r,k]} + \mathbf{N}_{[r,k]}$ where $\mathbf{N}$ is a column noise vector vector full of zeroes except at $\mathbf{N}_{[r,k]} \neq 0$. The contribution of the i-th server to the sum becomes:

$$\mathbf{V}^{[i]} \left(\mathbf{B}^{[i]}_{[:,k]} + \mathbf{N} \right). \quad (9)$$

By combining (9) and (8), it is possible to verify that the result retrieved by the client for the k-th column is:

$$\mathbf{B}_{[j,k]} + \mathbf{V}^{[i]}\mathbf{N}. \quad (10)$$

Since $\mathbf{N}_{[r,k]} \neq 0$, then $\mathbf{V}^{[i]}_j \mathbf{N}$ can take any possible value of the finite field $\mathbb{F}$ where computations are being performed. The probability of getting a value that when applied to H provides the right result, is less than or equal to $c/|H|$. Where c is the maximum number of elements from $\mathbb{F}$ assigned to a single output of H. By selecting a proper size for $|H|$, c can be made close to 1 with overwelming probability. Hence, we can detect random bit decay with probability:

$$\frac{|\mathbb{F}| - c}{|\mathbb{F}|}. \quad (11)$$

This proves that the detection capabilities of our scheme are better than in the naive sampling approach, regardless of the sampling parameters selected for small d in the asymptotic case. □

It is important to note that the previous proof does not imply that the file is safe from an adversary that can modify the information at all the servers. Consider an adversary that can modify $\mathbf{B}^{[i]}_{[r,k]}, \forall i \in S, r \neq j$ with the same value. Since the challenge phase of our construction wishes to cancel the contribution of each of the servers when $r \neq j$, our test is verifying that the non-retrieved rows are storing the same value at all the servers. Therefore, this method can only detect a smart adversary, when the exact row that was modified is retrieved. In terms of detection probability for this scenario, our scheme would be equivalent to the naive sampling scheme over each column of the matrix representation of the file.

4.2 A Scheme Against Smart Adversaries

To address the concern of a smart adversary that modifies blocks using a well defined strategy aimed at fooling the verifier, we can modify the system to return a result that includes information from all the given blocks in a column. The main observation of this scheme is that in a PIR protocol, we want random vectors with this property:

$$\sum_{i=1}^{S} V^{[i]} = \mathbf{E}^{[j]}. \quad (12)$$

However, for our purposes of verifying information, what we want to compute is the result of applying a random vector $\mathbf{V}$ to the columns of the matrix representing the file, hence

$$\sum_{i=1}^{S} \mathbf{V}^{[i]} = \mathbf{V}. \quad (13)$$

This is equivalent to applying a dot product using replication as a way to mask the secret vector $\mathbf{V}$ from the servers.

The modified scheme is as follows:

Setup: P wishes to upload a file which is modeled as a matrix $\mathbf{B}$ in the same way as in the *Challenge* phase of the sampling protocol. Then, P generates a random vector $\mathbf{V}$ and computes the dot product between $\mathbf{V}$ and each of the columns of $\mathbf{B}$ to generate the values:

$$\sigma_k = \mathbf{V} \cdot \mathbf{B}_{[:,k]} \quad (14)$$

where $\mathbf{B}_{[:,k]}$ represents the k-th column of matrix $\mathbf{B}$. Then, P uploads the file to the members of S and stores the σ_k values either locally or encrypted at S.

Challenge: P creates challenge vectors $\mathbf{V}^{[i]}$ for each server such that

$$\sum_{i=1}^{S} \mathbf{V}^{[i]} = \mathbf{V}. \quad (15)$$

Each server applies vector $\mathbf{V}^{[i]}$ to all the columns of its local version of the file $\mathbf{B}^{[i]}$ and returns the values $\sigma^{[i]}_k$ for each column k.

Verification: Once P has received all the responses $\sigma^{[i]}_k$ from all the servers, it adds them in the following way:

$$\sigma'_k = \sum_{i=1}^{|S|} \sigma^{[i]}_k \quad (16)$$

to obtain the result of computing the MAC to the given column. If $\sigma_k = \sigma'_k, 1 \leq k \leq |S|$ then all the servers are correct; otherwise, there is an error.

Similar to the previous scheme, since no information from $\mathbf{V}$ is revealed to any client, the same $\mathbf{V}$ can be reused many times without compromising its security.

Compared to the previous scheme, it is not possible for an adversary to set a random value $\mathbf{B}^{[i]}_{[r,k]} \forall i \in S$, because it would alter the total sum σ'_k. This happens because $\mathbf{V}^{[i]}_r \neq 0$ with high probability. We can avoid the possibility of $\mathbf{V}^{[i]}_r = 0$ by selecting a larger field or sampling a different number from a pseudorandom function if the output in the sequence is 0.

One difference between this scheme and the previous one, is that we are not returning actual blocks from the file, but rather a function of the blocks. For this reason, we need to prove that actually passing the test implies that the servers are storing a copy of the file.

Claim 2. *A set of servers computing the function correctly can recover the file with high probability.*

Proof. From the scheme's description it is possible to see that servers who have the correct file, can compute the function correctly. Now consider a scenario where at least one server is missing one correct block in a column. There are two options for these servers to provide a satisfactory response:

1. Send a random value and expect that the result adds to the correct σ_k. This happens with probability $1/|\mathbb{F}|$ because of the properties of the dot product and the combination procedure performed at P. By selecting a larger field size, the probability of this option succeeding becomes negligible.
2. Use previous responses to infer the right answer. This is possible whenever the new challenge vectors sent by P are linearly dependent to the previous ones. Assume the result for previous vectors $\mathbf{V}^{[i]}, \mathbf{W}^{[i]}$ was:

$$a_k = \mathbf{B}^{[i]}_{[:,k]} \mathbf{V}^{[i]} \text{ and } b_k = \mathbf{B}^{[i]}_{[:,k]} \mathbf{W}^{[i]}. \tag{17}$$

Then, for a given vector $\alpha\mathbf{V}^{[i]} + \beta\mathbf{W}^{[i]}$ where α, β are coefficients, the result is $\alpha a_k + \beta b_k$. This result can be computed by a server even when $\mathbf{B}^{[i]}_{[:,k]}$ is not available. Now, assume $q = |\mathbb{F}|$ and $n = \sqrt{L}$. Then, in order to reply to any query from P, the server needs to have n vector-response pairs. However, if this information is available, the file can be recovered using Gaussian Elimination. If less than n vector-response pairs are available, a copy of the file is still needed to reply to most queries.

To understand why this reasoning is true, consider the best scenario for a cheating server, that is: having the output of $n-1$ linearly independent vector-response pairs. In such a case, there are still $\lambda = q^n - q^{n-1}$ vectors that cannot be produced as a linear combination of the $n-1$ vector-response pairs owned by the server. Here q^n is the total number of vectors of length n over $\mathbb{F}$ and q^{n-1} is the number of linear combinations that can be formed with $n-1$ vectors. If we select a vector at random, the probability of choosing a vector for which the server does not have the necessary information to reply is:

$$\frac{\lambda}{q^n} = \frac{q^n - q^{n-1}}{q^n} = 1 - \frac{1}{q}. \tag{18}$$

Therefore, the probability of selecting a vector for which the server can reply correctly without having a copy of the file is:

$$1 - \left(1 - \frac{1}{q}\right) = \frac{1}{q}. \tag{19}$$

This probability becomes very small as q grows. In addition, the vector-response representation for the file is not advantageous for the server, since it requires more storage than storing the column itself. The previous argument also holds even if the servers uses smaller subvectors for the columns.

□

The drawback of this scheme is that when blocks are being overwritten, we need to subtract the contribution of the previous version of the block $\mathbf{B}_{[j,k]}$ to a given σ_k. Then, we must add the contribution of the new version of the block $\mathbf{B}'_{[j,k]}$. The procedure is illustrated in this equation:

$$\sigma'_k = \sigma_k - \mathbf{V}_j \mathbf{B}_{[j,k]} + \mathbf{V}_j \mathbf{B}'_{[j,k]}. \tag{20}$$

Unfortunately, this procedure involves downloading the current block $\mathbf{B}_{[j,k]}$ from some member of S. For this reason, we believe this construction is more suited for systems where the blocks do not change often, as any update operation involves one additional *read* operation.

Up to this point, both proposals can detect whether at least one of the servers is storing corrupt data, but they do not identify which one exactly. In the next section we will see how to find the corrupt servers from the received responses.

4.3 Finding the Corrupt Servers

The two proposed schemes involve hiding the secret vector by splitting it into several vectors; shares of this vector are sent to the different nodes. This is a particular case of a Threshold Scheme (Shamir, 1979), where n shares are created and at leastt $t+1 \leq n$ of them are needed to recover the secret.

One could try several approaches to find the corrupt server, such as repeating the test with a different subset of them. Unfortunately, this is not efficient. A better approach is presented in (Goldberg, 2007): assume that our secret vector $\mathbf{V}$ has $|\mathbf{V}|$ coordinates $v_1,\ldots,v_{|\mathbf{V}|}$. For each coordinate of $\mathbf{V}$, create a polynomial of degree $t < |S|-1$

$$f_j(x) = a_{j,t}x^t + \ldots + a_{j,1}x + a_{j,0} \tag{21}$$

where $a_{j,0} = v_j$ (the secret to be shared) and the other coefficients $a_{j,k}, k \neq 0$ are selected randomly. Here, $t+1$ denotes the minimum number of servers needed to recover $\mathbf{V}$. Each server receives a vector of the form

$$f_1(c_i),\ldots,f_{|\mathbf{V}|}(c_i) \tag{22}$$

where c_i is a random coefficient used for the server i for this particular test. P receives the dot product of this vector with the column k of the file matrix:

$$\mathbf{B}^{[1]}_{[j,k]} f_j(c_1),\ldots,\mathbf{B}^{[|S|]}_{[j,k]} f_j(c_{|S|}). \tag{23}$$

By selecting any $t+1$ of them, it is possible to interpolate and recover $\mathbf{B}_{[j,k]}a_0$, and given that a_0 is known by P, it is possible to recover $\mathbf{B}_{[j,k]}$. Since we are working in an scenario where we have evaluations of the same polynomial at different points, we need to solve an interpolation with errors problem. This is equivalent to decoding Reed-Solomon codes.

Given the previous presentation of Goldberg's scheme for PIR, it is clear that it can be applied for our sampling scheme. For the scheme considering smart adversaries, the element returned from the k-th column of the i-th server will be of the form

$$\sum_{j=1}^{|\mathbf{V}|} \mathbf{B}^{[i]}_{[j,k]} f_j(c_i) \tag{24}$$

Once we select the result of column k at $t+1$ servers, the result of this sum over the free term once interpolation is performed, is given by:

$$\sum_{j=1}^{|\mathbf{V}|} \mathbf{B}^{[i]}_{[j,k]} a_{j,0} = \sum_{j=1}^{|\mathbf{V}|} \mathbf{B}^{[i]}_{[j,k]} v_j = \sigma_k \tag{25}$$

Which is the same value we expected to obtain in the scheme using vector addition. If there are some nodes returning different values, then a decoding procedure that can find the errors can be applied to identify the corrupt servers.

In terms of bandwidth usage, the advantage of using a more general secret sharing scheme, is that we only need to send $|S|$ vectors to the servers and we can determine what nodes are transmitting correct values to P. In terms of security, it prevents corrupt servers from performing attacks based on P's reaction to wrong responses. One instance of such attack is presented in (Patterson and Sassaman, 2007) where a covert channel for the servers is created. The disadvantage of this scheme is that we now need to evaluate a polynomial of degree $t+1$, $|S|$ times for every coordinate of the secret vector $\mathbf{V}$. This makes the protocol more computationally intensive for P.

5 SIMULATION

The simulation environment consisted of a Windows 8.1 machine with an Intel Core $i7-3770$ CPU running at 3.4 GHz and 16 GB of RAM. The programming environment was the Java JDK 1.7.0 release 21. We used the JDK's BigInteger class for large number arithmetic and set the prime for the field as 170141183460469231731687303715884105757, the first prime number larger than 2^{128}. The purpose of this implementation was to show that the proposal is practical, rather than provide an optimized version of it.

We used a file with 32768 blocks of size 4 KB. For this parameters, the matrix representation had 182 blocks per column. Each column had in total $e = 46592$ elements each one of size 17 bytes. These parameters correspond to a file with a length similar to 128 MB. We experimented with Shamir's scheme involving polynomial evaluation and the scheme where vectors are added. Shamir's scheme was considerably slower, taking 2421 ms for 10 servers and 542 ms for 3 servers. The vector scheme took 990 ms and 400 ms respectively. The evaluation routine for Shamir's scheme was performed using Horner's rule. The dot product step on the matrix took 5172 ms, for an effective throughput of 24.74 MB/s. The total amount of information needed to be transmitted to verify for the 3-server scenario was 4.26 MB corresponding to 3.3% of the total file. The time needed to upload and download the data was not included, since it varies according to the network.

To compare our construction with an existing one, we implemented the private verification scheme from (Shacham and Waters, 2008). We set the transmission overhead to a constant at the cost of increasing the storage at the server to twice the original size of the file. This was done to compare against the most transmission-efficient version of the scheme. The field used for computations was the same. Throughput on the server part was 12.1 MB/s given that the server needs to generate random numbers. Generation of the challenge from the client is significantly faster, since it only involves sending the seeds of a random generator.

6 CONCLUSIONS

We presented two PDPs based on fast multi-server PIR that have several desirable properties and whose complexity is sublinear in the size of the file. We showed that the proposals can detect data corruption due to random failures with high probability. One of the proposals can work with dynamic files and has a very fast setup stage that only involves a hash function. Its drawback, is that it cannot detect corruption when an attacker modifies the servers in a coordinated fashion. This drawback is solved in the second scheme; however, the scheme pays a penalty when it is used for dynamic files, by requiring an additional read operation. The downside of both schemes is the size of the transmitted information and lack of secure public verifiability.

REFERENCES

Apon, D., Katz, J., Shi, E., and Thiruvengadam, A. (2014). Verifiable oblivious storage. In *Public-Key Cryptography–PKC 2014*, pages 131–148. Springer.

Ateniese, G., Burns, R., Curtmola, R., Herring, J., Kissner, L., Peterson, Z., and Song, D. (2007). Provable data possession at untrusted stores. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, CCS '07, pages 598–609.

Ateniese, G., Di Pietro, R., Mancini, L. V., and Tsudik, G. (2008). Scalable and efficient provable data possession. In *Proceedings of the 4th international conference on Security and privacy in communication networks*, page 9. ACM.

Bowers, K. D., Juels, A., and Oprea, A. (2009). Hail: a high-availability and integrity layer for cloud storage. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 187–198. ACM.

Cash, D., Küpçü, A., and Wichs, D. (2013). Dynamic proofs of retrievability via oblivious ram. In *Advances in Cryptology–EUROCRYPT 2013*, pages 279–295. Springer.

Chor, B., Kushilevitz, E., Goldreich, O., and Sudan, M. (1998). Private information retrieval. *Journal of the ACM (JACM)*, 45(6):965–981.

Corena, J. C. and Ohtsuki, T. (2013). Proofs of data possession and pollution checking for regenerating codes. In *Global Communications Conference (GLOBECOM), 2013 IEEE*, pages 2717–2722.

Curtmola, R., Khan, O., Burns, R., and Ateniese, G. (2008). Mr-pdp: Multiple-replica provable data possession. In *Distributed Computing Systems, 2008. ICDCS'08. The 28th International Conference on*, pages 411–420. IEEE.

Dodis, Y., Vadhan, S., and Wichs, D. (2009). Proofs of retrievability via hardness amplification. In *Theory of Cryptography*, pages 109–127. Springer.

Goldberg, I. (2007). Improving the robustness of private information retrieval. In *Security and Privacy, 2007. SP'07. IEEE Symposium on*, pages 131–148. IEEE.

Hanser, C. and Slamanig, D. (2013). Efficient simultaneous privately and publicly verifiable robust provable data possession from elliptic curves. In *SECRYPT 2013*, pages 15–26. SciTePress.

Le, A. and Markopoulou, A. (2012). Nc-audit: Auditing for network coding storage. In *Network Coding (NetCod), 2012 International Symposium on*, pages 155–160.

Olumofin, F. and Goldberg, I. (2012). Revisiting the computational practicality of private information retrieval. In *Financial Cryptography and Data Security*, pages 158–172. Springer.

Patterson, M. L. and Sassaman, L. (2007). Subliminal channels in the private information retrieval protocols. In *Proceedings of the 28th Symposium on Information Theory in the Benelux, NL*.

Raphael, J. (2013). The worst cloud outages of 2013 (so far),. http://www.infoworld.com/slideshow/107783/the-worst-cloud-outages-of-2013-so-far-221831. Accessed: April 9th 2014.

Shacham, H. and Waters, B. (2008). Compact proofs of retrievability. In *Advances in Cryptology-ASIACRYPT 2008*, pages 90–107. Springer.

Shamir, A. (1979). How to share a secret. *Communications of the ACM*, 22(11):612–613.

Sion, R. and Carbunar, B. (2007). On the computational practicality of private information retrieval. In *Proceedings of NDSS*.

Tamassia, R. (2003). Authenticated data structures. In *Algorithms-ESA 2003*, pages 2–5. Springer.

Trostle, J. and Parrish, A. (2011). Efficient computationally private information retrieval from anonymity or trapdoor groups. In *Information Security*, pages 114–128. Springer.

Wang, C., Wang, Q., Ren, K., and Lou, W. (2010). Privacy-preserving public auditing for data storage security in cloud computing. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–9.

Xu, J. and Chang, E.-C. (2012). Towards efficient proofs of retrievability. In *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, pages 79–80. ACM.

Zhu, Y., Ahn, G.-J., Hu, H., Yau, S. S., An, H. G., and Hu, C.-J. (2013). Dynamic audit services for outsourced storages in clouds. *Services Computing, IEEE Transactions on*, 6(2):227–238.