Proofs of Retrievability in 2 server setting

Nishant Kumar, Tapas Jain
Shweta Agrawal, Ragesh Jaiswal

April 1, 2015

Abstract

We start with assuming that there are these 2 servers that are not talking to each other. With this assumption in place, suppose one as a user wants to store his file on both the servers. Then we show that lots of interesting things can be achieved. Like for example, we can develop a completely keyless system, where, by introducing some amount of intelligent error in the files that a user stores on the servers, the 2 servers can be pitted against each other and we can get away without storing any kind of information with the user.

1 Motivation

The primary motivation is to get simpler, more efficient protocols for outsourcing tasks. A hope will be to reduce the number of secret keys and file specific information that a user needs to store in order to benefit from cloud computing. To protect against other users, a user certainly needs to store one password per server, but the hope is to not store a different password per file.

2 Existing Solutions for Proofs of Retrievability

2.1 Juels and Kaliski

The paper discusses the motivation and the formal definitions behind PORs, giving a precise security definition. The authors then move on to present a POR protocol, secure in their definition, whereby they introduce check blocks - so-called "sentinals", randomly in the file. Later the user can query for these sentinal values dispersed randomly throughout the file, and since sentinal values were generated using one-way functions, and the user has the seed, he can cross-check the sentinal value returned by server against the actual one. Some important points worth mentioning are:

- The protocol only queries for the positions of the sentinal blocks, since these values is what the user can generate (using just the key of the one-way function used to generate the sentinal).
- It is important to randomly disperse the sentinels throughout the file, so that the adversarial server cannot find the locations of the sentinals.

- Since the sentinal are just some random bits of data, it is also important that the adversarial server is unable to distinguish between the sentinals and the contents of the file. So, it is necessary to encrypt the file, first before adding the sentinals.
- Hence having the described the need for encrypting and randomly dispersing the sentinals, the protocol first adds some Error Correcting Code(ECC), encrypts the file, generates the sentinals using a suitable one-way function, adds the sentinals and shuffles the blocks using a pseudo-random permutation.
- Once the user creates the file and stores it on the server, the user can simply remove the file and all he needs to store is the key he used in permuting the blocks and generating the sentinals.

2.2 Compact Proofs of Retrievability - Shacham and Waters

The paper focusses on the need to have statelessness and unbounded use to have public verifiability in any scheme. The authors then present 2 schemes - one with public verifiability and one without. We discussed in detail the public verifiability scheme.

- The main overall idea used by SW was to use some kind of heavy machinery based on BLS signature schemes, to store some kind of signature along with the file stored in the server.
- Then to check if the file was being stored correctly, all the protocol does is to ask some function of the data and the signature and check one against the other.
- The signature scheme used is itself so heavy, that it not only helps in verification, but also compresses the responses.
- The authors have also presented an extraction algorithm, which simply keeps collecting information from the verification protocol until enough information has been acquired.

3 Keyless protocol for POR: does not handle retrieval

We assume that a user, say Alice wants to store her file on 2 servers - say Google and Amazon. As we shall see, by introducing intelligent error in the file she stores on the 2 servers, she can get away without storing any kind of key or private information. All the parameters will be publicly known. All this, assuming that the servers - in this case - Google and Amazon - are not colluding with each other.

3.1 The protocol - An example

Suppose Alice has a file - say F, that she intends to store on Google's and Amazon's servers.

- She first introduces some Error Correcting Code(ECC) in the file F - sufficient enough to tolerate the error that will be introduced in the next step. Say the new file is F'.
- Carefully she decides to introduce some f fraction of error in the files.
- She chooses some $l' = f \cdot len(F')$ positions (say P) to introduce error. Choose some error vector e of length l' randomly. Create 2 new files - F_1 and F_2 s.t. $F_1(x) = F'(x) \otimes e_x, x \in P$ and $F_2(x) = F'(x) \otimes \overline{e_x}, x \in P$, where e_x = bit of e corresponding to position x.

  Note: $a \otimes b$ =bitwise xor operation. Also, if say $x \in P$, then $F_1(x) = \overline{F_2(x)}$.
- Now suppose she stores the files - F_1 and F_2 on the 2 servers and does not retain any kind of information about the file with herself. All the other parameters - like the file size, etc. are made publicly available.
- Suppose now some time has passed and she intends to check if the 2 servers are honest and storing the original files correctly. *Important point to note here is that she does not intend to retrieve the file right now - just to check if the 2 servers are honest or any of them is cheating.*
- Since the file size, etc. are publicly known, she randomly samples some bit positions and asks the 2 servers for the corresponding bits.
- She makes q such queries. Now since she initially introduced only f fraction of error in the original file, she should expect some $f \cdot q$ queries to mismatch between the 2 servers. If a lot more queries differ, then she can with some probability (subject to discussion) that one of the 2 servers is cheating. These probability bounds can be found using chernoff bounds.

3.2 The concrete protocol

–Todo–

3.3 Improving Efficiency of Keyless protocol

In the protocol, as presented so far, each individual bit is being queried and the server sends replies for each such query. This can be improved upon using some kind of hashing s.t. the user sends a bunch of positions and the server replies with the hash of the bits at those positions.

- Using *distance preserving hash functions* can also help - it can become easy to figure out from the replies of the 2 servers whether the values hashed differ and if yes, then by what extent. This idea is like thinking about to query the server in blocks and process the responses.
- One idea is to use some kind of light machinery for figuring out if any of the servers cheated and if any of them is found to have cheated, run some heavy machinery to figure out exactly who cheated.

–To explore further– **We do not need aggregate signatures now because we don't need signatures. For efficiency, we can just ask for aggregating functions such as $\sum r_i x_i$. We need to analyze the best case for this.**

4 File Retrieval

In the model that we have been developing so far, anyone can verify for the files stored on the servers. We have not yet talked about extraction. For extraction, either we need some form of authentication for which we again have to store something, or we need to look for areas/applications, where anyone is allowed to extract the files for anyone.

How do existing solutions do file retrieval? Do they need a SK per file or per server? In SW, there is a public τ associated with each file, which is needed for checking as well as retrieval. We want to make the client store as little as possible.

5 Tracing the Culprit

Figuring out which of the 2 servers cheated seems like a difficult question, provided we still maintain the invariant that the user does not store any secret key with himself. So, we talked about introducing the assumption that only one of the servers is adversarial and then trying to figure out who is the one who is cheating. Also, we talked about introducing the assumption on the kinds of tampering the adversarial server is allowed - like practically it is reasonable to assume that such a server will only delete the data blocks and not try to corrupt some specific bits.

- As stated in the previous section, one direction to think about is to run some light machinery like our protocol to figure out if someone has cheated, and if yes, then to figure out who has cheated, run some heavy machinery. The advantage we get is that if no-one is cheating, our protocol is very fast.

The main challenge is to figure out who cheated (assuming that only one of them does), without storing any kind of key with the user.

6 Function computation

So far, we have been talking about storing a file with these 2 servers, figuring out if anyone is malicious, meaning has anyone tampered with the contents of the original file and if yes, then who.

We ended the last section discussing that in a completely keyless system, it might as well be difficult to figure out who exactly cheated - as we donot have any sort of evidence to check either server's validity of claim that they have the right file.

So,we look beyond just figuring out who cheated and look at more practical settings. Like consider the more practical problem of function computation on this data.

- Since the 2 servers have original files with error added to it, the same function computed on the 2 files may give wildly different results.
- So, then one possible direction to look forward to is to figure out for what classes of functions, can we introduce intelligent errors in the original file, so that we figure out the exact function. Like consider this simple example. Suppose we wish to compute $x_1 + x_2$ and the original file is broken into 2 files - one having x_1 and the other having x_2. One possibly intelligent way to introduce errors is then to change one file from x_1 to $x_1 + \epsilon$ and the other one from x_2 to $x_2 - \epsilon$. Then store these 2 new files with the servers. Later ask each server for their data, and then add them up to get $x_1 + x_2$.
- The idea behind the simple example discussed may be extended at a very high level to linear spaces, where we somehow add error in some orthogonal space and store the files. The fact that the error is added in some orthogonal space will later help in cancelling out the errors and recovering the original data.

The point to search further from here may be to search for specific classes of functions, s.t. by introducing apt error in the files stored with the 2 servers, the error can somehow be cancelled.

7 Further directions

Since we are proposing a new setting, there are a lot of things to ponder upon. We briefly also touched upon introducing privacy measures into the scheme. Also, as discussed in the meetings, the memory delegation paper presents some new techniques to get away in sub-linear time computation, delegating the computation to the cloud.

References

- PORs: Proofs of Retrievability for large files - Juels and Kaliski
- Compact proofs of Retrievability - Shacham and Waters - http://cseweb.ucsd.edu/ hovav/dist/verstore.pdf