Cloud Security & Proofs of Retrievability
An Extended Survey

A Thesis
Submitted for the degree of
MSc Digital Systems Security

by
Vasiliki Grimpa

University of Piraeus
dept. Digital Systems
Supervisor: Prof.Konstantinos Labrinoudakis

February 2016

Table of Contents

Abstract

In this paper we survey the most important POR schemes proposed since 2007 along with different methods used for various security requirements. through a comparative study, a consolidated report of the techniques used in POR is presented.

1. Introduction

Cloud computing is essentially a large-scale distributed and virtual machine computing infrastructure. Due to the remarkable attention that Cloud Computing receives from both industrial and academic community, Cloud Storage Systems popularity respectively increases. Amazon S3, Dropbox, Google Drive etc. are extendedly used as means of storing data reliably and making it easily accessible from any location. IDC estimates that by 2020, nearly 40% of the information will be involved in cloud computing. [17]

A cloud storage system includes two parties: *cloud storage servers* maintained by a cloud service provider (CSP) with a huge amount of storage space and storage services provided in a pay-as-you-go manner and *clients* with limited storage but with a large amount of data to be stored. In order for a client to store their data on cloud storage server, they need to divide the data files into blocks, outsource them and then they can delete the local copies keeping only a small amount of metadata. Afterwards, a client can dynamically use the storage service, by performing block-level update operations, such as modify a block, insert a block or delete a block. [15]

A wide variety of virtual and dynamically scalable resources are available today via Internet technologies, including computational power, storage, hardware platforms and applications. There are many advantages for private and public organisations that decide to migrate all or some of their information services to the cloud computing environment. Examples of these benefits include increased flexibility and budgetary savings through minimisation of hardware and software investments. [9] The CSP deals with the complicated and expensive tasks of IT services management and maintenance, allowing the users to productively focus on their core business, quickly adjust resources to their needs, without having to worry about the physical location or configuration of the supporting system.

In spite of the obvious advantages that come with adopting cloud technology, cloud storage also raises security and privacy concerns associated with placing data off-premises, mainly on availability and integrity. After outsourcing the data to an off-site storage system and locally deleting it, clients are free from the burden of storage but lose physical control of their data. A cloud server storing many users' data would most probably be a preferred attack target setting the data at risk. A technical way is necessary for the users to be assured that the cloud storage server is following the service level agreement (SLA). [12]

Threats to integrity of data stored in cloud are indeed realistic. There was a large-scale leak of user data of Google mail in February 2009 and March 2011. A large number of the servers in Amazon cloud data center crashed in April 2011, experiencing significant downtime. There have been already security breach incidents in cloud based services, such as the corruption of Amazon S3, due to an internal failure caused by mismatching files with customers' hashes. [9] There are also plenty of data loss cases that are claimed by individuals but not confirmed officially by the cloud server, e.g. data loss cases in Dropbox. None of today's storage service providers in the cloud (e.g., Amazon Simple Storage Service (S3) and Google's BigTable) guarantee any security in their service level agreements.

It becomes obvious that a CSP cannot be completely trusted as a third party. Normally, the CSP will perform operations correctly, and will not deliberately delete or modify clients' data. However, in some cases CSPs may act unfavorably to the security of the user data. For their own benefits, CSPs might discard portion of rarely accessed data to save storage space. Also, the CSPs may be tempted to conceal the hosted data corruptions caused by management errors, server hacks or Byzantine failures to maintain reputation, or intentional tamper or leak user data for some interest. [25] Security issues related to data integrity and availability are the main obstacles for cloud storage to be fully successfully adopted.

A cloud storage architecture should satisfy multiple security requirements such as data confidentiality, data integrity, data availability, fairness and data freshness. When the CSP is untrusted or semi-trusted (lose or corrupt the hosted data inadvertently), an important challenge is how to offer provable outsourced storage guarantees. In particular, a data owner (client), apart from data correctness (which is equivalent to integrity and freshness), also wishes to obtain retrievability guarantee. The client needs assurance that the server is indeed storing all of the client's data, and that no critical data loss has occurred. [21]

Remotely downloading the entire data and periodically check its integrity would be the naive solution. That solution would have a high communication cost due to the expensiveness in I/O and transmission cost across the network, and deviates from the original appealing features of cloud storage. A better approach is that a client performs an audit of the storage provider. A successful audit verifies that the provider stores the data, without the retrieval and transfer of all the data from the provider to the client. [22] As users no longer physically possess the storage of their data, traditional cryptographic primitives for the purpose of data security protection cannot be directly adopted.

In order to solve remote integrity checking problem in cloud storage, a lot of works have been done focusing on various conditions of application and attempting to achieve different goals. In the past decade, two effective solutions have been proposed under different models; the "Provable Data Possession (PDP)" model and the "Proof of Retrievability (PoR)" model. [25]

2. Data Integrity Mechanisms for outsourced data (PDP vs POR)

Clients either have to accept the burden of regularly verifying their outsourced data, or entrust the CSP to deploy the necessary security mechanisms that ensure data integrity in spite of server failures, exploits, etc. Integrating of such security mechanisms in current clouds often incurs considerable costs on the cloud providers, which explains the reason why none of today's cloud storage services accept liability for data loss in their Service Level Agreements (SLAs) and only guarantee service availability – in spite of the plethora of cloud security and dependability solutions that populate the literature. [24]

The ability of a storage system to generate proofs of possession of client's files, without having to retrieve the whole file, is the so-called "proofs of data possession"(PDP). If the client's data can be extracted from proofs of data procession, then the scheme is developed to be a "Proof of Retrievability" (POR) scheme. [10]

A *PDP* scheme, first presented by Ateniese et al. is an integrity verification scheme [25] for static data that only detects a large amount of corruption in outsourced data. Their scheme utilizes the RSA-based homomorphic authenticators for auditing outsourced data and suggests randomly sampling a few blocks of the file. However, the schemes have to set a prior bound on the number of audits and doesn't support public auditability. Public auditability allows an external party, in addition to the user himself, to verify the correctness of remotely stored data. Erway et al. was the first to propose dynamic PDP scheme. They developed a skip lists-based method to enable provable data possession with dynamic support.

In the PDP model, the clients query the server periodically and the server returns a proof to guarantee that a certain percentage (e.g., 99%) of the file are intact. But if a very small amount of the file is lost or corrupted, the clients may not be able to detect it. In this case, the clients cannot retrieve their data intactly. [15]

A successful PoR audit ensures that the server maintains knowledge of all outsourced data blocks; while a PDP audit only ensures that the server is storing most of the data. With PDP, a server that has lost a small number of data blocks can pass an audit with significant probability. While some PDP schemes achieve full security, they require that the server read all of the client's data during an audit, and thus is impractical. [21]

So a PDP scheme can only verify the integrity of a file on the server, but cannot ensure the retrievability of a file. However, a PoR scheme is a challenge-response protocol. In the protocol, the server can demonstrate to the client that a file is intact and retrievable without any loss and corruption. [18] The main difference between PoR and PDP is the notion of security that they achieve. A PoR audit guarantees that the server maintains knowledge of all of the client data, while a PDP audit only ensures that the server is storing most of the client data. On a technical level, the main difference in most prior PDP/PoR constructions is that PoR schemes store a redundant encoding of the client data on the server. [13] Note that PDP is a weaker security property. Indeed, it is already observed in [13] that accommodating dynamic data in the setting of POR is more challenging than accommodating dynamic data in the setting of PDP. [6]

3. Proof of Retrievability - Definitions & Preliminaries

Proof of retrievability (POR), a concept first presented by Juels and Kaliski in 2007, incorporates a cryptographic challenge-response protocol that enables a prover (CSP) to demonstrate to a verifier (client) that a file *F* is stored correctly and is retrievable (i.e., recoverable without any loss or corruption), along with an extractor that will actually reconstruct the file, given the algorithm of a "prover" who is able to correctly respond to a sufficiently high percentage of challenges. [11, 8] The response can be highly compact (tens of bytes), and the verifier can complete the proof using a small fraction of *F*. [27] This highlights a strong need to seek an effective solution for checking whether their data have been tampered with or deleted without downloading the latest version of data. [5]

To conduct and verify POR, users typically need to be equipped with devices that have network access, and that can tolerate the (non-negligible) computational overhead incurred by the verification process. This clearly hinders the large-scale adoption of POR by cloud users, since many users increasingly rely on portable devices that have limited computational capacity, or might not always have network access. [24]

The benefit of a POR over simple transmission of *F* is efficiency. As a standalone tool for testing file retrievability against a single server though, a POR is of limited value. Detecting that a file is corrupted is not helpful if the file is irretrievable and the client has no recourse. Thus PORs are mainly useful in environments where *F* is distributed across multiple systems, such as independent storage services. In such environments, *F* is stored in redundant form across multiple servers. [27]

POR schemes can be classified into the ones that support public verification and the ones that support private verification. In the case of the public verification schemes, everyone can perform the role of verifier in the POR protocol, thus a third-party auditor can participate, while in the private verification PORs only the data owner (client) is able to

play the role of the verifier. Other attributes examined in POR schemes are the storage cost, the communication cost and the computation cost.

Attack models and security assumptions

As stated before, a CSP cannot be completely trusted as a third party. Whether the CSP is considered untrusted, semi-trusted or curious, it operates as a Storage Service Provider under a service level agreement with the client, specifying properties such as throughput, response time, availability, and recovery-time objectives. The case primarily assumed is that where the SSP stores the file, while the client itself does not retain a copy; if the client did, it could just verify retrievability by sampling and comparing random blocks against its own copy, without the need for the POR *per se.*

The price of the service is set by the SSP at some profit margin above the cost of providing the service at a given level (equipment, maintenance, staff, facilities, etc.). An SSP is thus motivated legitimately to increase its profit margin by reducing cost while maintaining the same service level; in a competitive marketplace this will ultimately reduce the price, which is a benefit to clients as well.

Without a requirement to provide continuous service level assurances, an SSP may also be willing to take the risk of decreasing its cost by not maintaining an agreed service level. For instance, an SSP may move files it considers less likely to be accessed to a lower tier of storage than agreed. These lapses are exacerbated by the possibility that the SSP may itself rely on other SSPs to store files or parts of them. For instance, to meet an agreed availability level, an SSP may replicate data on geographically distributed sites, perhaps employing information dispersal techniques as suggested in section 1.3. Some of these sites may be operated by other SSPs, who in turn may have their own motivations to reduce cost, legitimate or otherwise. If a site knows that its occasional outage will be overlooked (indeed, planned for) due to the presence of its replication peers, it may opt to increase its frequency of "outages" by placing a fraction of files on lower tiers—or not storing them at all. (This is akin to the "freeloading" threat described by Lillibridge et al. for peer-to-peer storage systems.)

Another reason a malicious SSP may corrupt or delete certain files (or portions of them) is their content. Encryption partially mitigates this threat since an SSP does not directly know the content of encrypted files, but it may still be possible for other parties to inform the SSP by back channels of which files to "misplace," or to cause the misplacement themselves by physical attack. E-discovery of documents is one scenario motivating these concerns.

Equipment failures and configuration errors may also result in file placement that does not meet an SLA; the breach of agreement may simply be due to negligence, not malice.

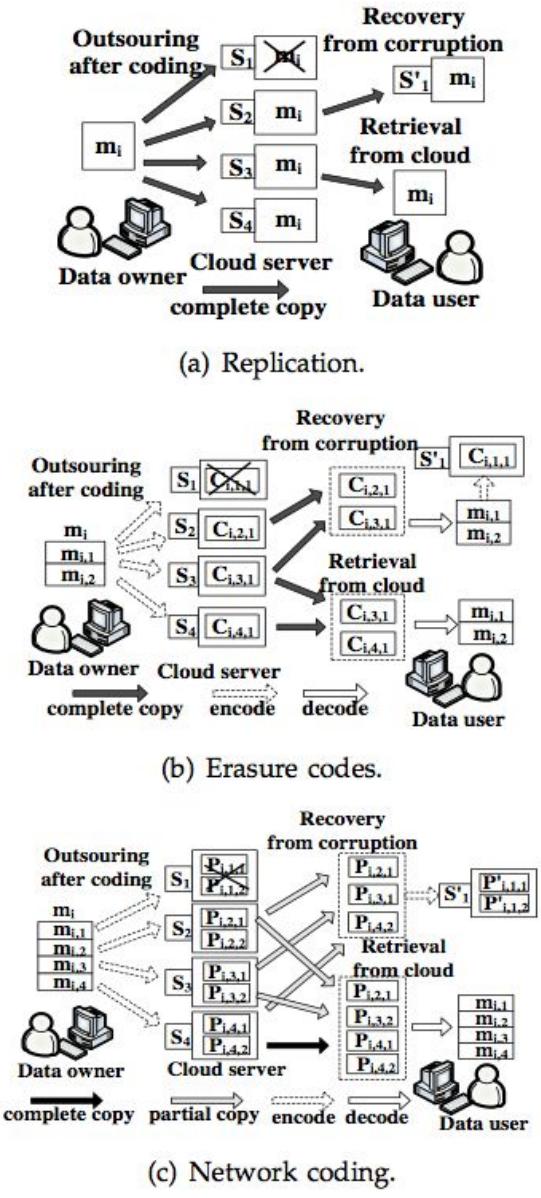
We envision that a POR scheme would be applied to a storage service as follows. As part of its SLA, an SSP would offer periodic, unannounced execution of a POR for selected files. In the POR, a block would be considered to be an erasure if it cannot be read within the agreed response time (after accounting for variations in network latency, etc.). The

client, taking the role of verifier, would thereby obtain (probabilistic) assurance that the agreed service level continues to be met for the file. [1]

If the SSP is trusted to provide file integrity, then a redundancy approach such as an erasure code would be sufficient for error correction, thus data recovery.

Redundancy Approaches for Data Recovery

Data redundancy techniques such as replication, coding (e.g., erasure codes and network coding) are usually employed in the distributed storage systems to improve the data reliability and availability.



(a) Replication.

(b) Erasure codes.

(c) Network coding.

Figure 1. Redundancy approaches [25]

a) **Replication** is the simplest way to generate redundancy. In the replication approach, the original data is completely copied to each of n storage servers. Data users can retrieve the original data by accessing any one of the storage servers. When one server is corrupted, the original data can be recovered by simply copying the entire data from any one of the healthy servers. Obviously, the storage overhead of this redundancy method is very expensive.

b) Using **erasure codes** for obtaining data redundancy, data users can recover the whole m original data blocks by retrieving the same number of encoded data blocks from any k of n servers. As a redundancy-reliability tradeoff each server only needs to store m/k encoded data blocks. However, compared with the replication-based solution, erasure codes may have a higher network overhead for the data recovery. In Fig.(b), the parameters (m, k, n) are set to be (2,2,4), respectively. The original data block m_i is first divided into 2 native data blocks equally. These two native blocks are then encoded into 4 encoded blocks $C_{i,j,l}$ which are stored in 4 cloud servers. Each cloud server stores one encoded blocks. The size of $C_{i,j,l}$ is equal to the size of the native data block.

3) Using **network coding**, a given file object is first divided into equal-size native data blocks. The native data blocks then are encoded by linear combination to form encoded data blocks which are distributed over $n > k$ storage servers. The original data file object can be reconstructed from the encoded blocks contained in any k of the n servers. Thus, it tolerates the failure of any $n - k$ storage servers. The network coding can achieve a tradeoff between storage and communication cost. In Fig.(c) the parameters (n,k) are set to be (4,2). In this example, the original data block mi is first divided into 4 native data blocks with equal size. These native data blocks are then encoded into 8 encoded blocks Pi,j,l and store them in 4 cloud servers, where each cloud server stores two encoded blocks. Here, the size of Pi,j,l is equal to the size of the native data block. [25]

Modern erasure codes, e.g., fountain codes such as Raptor codes, operate in linear time and are amenable in some cases to practical application to fairly large files or file segments without any need for chunking. Additionally, it is possible (if not generally practical) to treat a full file as a single message in an error-correcting code with large minimum distance. In such cases, we can obtain considerably tighter bounds on the security of our POR system.

Erasure codes such as fountain codes by themselves, however, do not provide robust guarantees over an adversarial channel. Their strong resilience bounds apply only to erasures in a stochastic channel. Thus, to achieve the bounds given here in a POR, the encoding steps of encryption and permutation are still essential. These steps effectively reduce an adversarial channel to a stochastic one. Additionally, for very large files, application of a fountain code across the entire file can be challenging, as such codes typically require repeated random accesses across the full file. Thus, chunking may still be desirable, and permutation can then provide enhanced resilience by eliminating chunk structure.

When file blocks are MACed, it is effectively possible to convert an erasure code into an error- correcting code. The decoding process simply discards corrupted blocks. [1]

5. Surveying PoR schemes

The first efficient PoR scheme was introduced in 2007 by Jules & Kaliski and constitutes the first formal, concrete security definition of PORs and the basic notation for most related PoR schemes to be proposed since then. Their scheme use spot-checking and error-correcting codes to ensure both possession and retrievability of remote data files. [11]

According to their scheme, a POR may be viewed as a kind of cryptographic proof of knowledge (POK) specially designed to handle a large file F. In a POR, unlike a POK, neither the prover nor the verifier need actually have knowledge of F. Successfully executed, PoR assures a verifier (client) that the prover (CSP) presents a protocol interface through which the verifier can retrieve F in its entirety. Although the verifier can refuse to release F after participating in a PoR, the protocol provides the strongest possible assurance of file retrievability barring changes in prover's behavior. A POR can be efficient enough to provide regular checks of file retrievability. Consequently, as a general tool, a POR can complement and strengthen any of a variety of archiving architectures, including those that involve data dispersion.

This POR protocol encrypts F and randomly embeds a set of randomly-valued check blocks called sentinels. [1] The use of encryption here renders the sentinels indistinguishable from other file blocks. The verifier challenges the prover by specifying the positions of a collection of sentinels and asking the prover to return the associated sentinel values. If the prover has modified or deleted a substantial portion of F, then with high probability it will also have suppressed a number of sentinels. It is therefore unlikely to respond correctly to the verifier. To protect against corruption by the prover of a small portion of F, error-correcting codes are also employed.

The scheme comprises 6 functions:

keygen$[\pi] \rightarrow \kappa$ generates a secret key κ.

encode$(F; \kappa, \alpha)[\pi] \rightarrow (F\eta, \eta)$

The function encode generates a file handle η that is unique to a given verifier invocation. The function also transforms F into an (enlarged) file $F\eta$ and outputs the pair ($F\eta$, η).

//4 steps: error-correction, encryption, sentinel creation, permutation

extract$(\eta; \kappa, \alpha)[\pi] \rightarrow F$

The function extract is an interactive one that governs the extraction by verifier V of a file from a prover P. In particular, extract determines a sequence of challenges that V sends to P , and processes the resulting responses. If successful, the function recovers and outputs F η .

challenge(η ; κ , α)[π] → c

The function challenge takes secret key κ and a handle and accompanying state as input, along with system parameters. The function outputs a challenge value c for the file η .

respond (c, η) → r *//the only function executed by the Prover*

generates a response to a challenge c. Note that in a POR system, a challenge c may originate either with challenge or extract.

verify ((r, η); κ , α) → b ∈ {0, 1}

The function verify determines whether r represents a valid response to challenge c. The challenge c does not constitute explicit input in our model; it is implied by η and the verifier state α . The function outputs a '1' bit if verification succeeds, and '0' otherwise.

The function **encode** entails four steps:

1. Error correction: We carve our file F into k-block "chunks." To each chunk we apply an (n, k, d)-error correcting code C over GF [2^l]. This operation expands each chunk into n blocks and therefore yields a file F′ = F′ [1],...,F′ [b′], with b′ = bn/k blocks.

2. Encryption: We apply a symmetric-key cipher E to F′ , yielding file F′ ′ . Our protocols require the ability to decrypt data blocks in isolation, as our aim is to recover F even when the archive deletes or corrupts blocks. Thus we require that the cipher E operate independently on plaintext blocks. One option is to use a l-bit block cipher. In this case, we require indistinguishability under a chosen-plaintext attack; it would be undesirable, for example, if an adversary in a position to influence F were able to distinguish the data contents of blocks.2 In practice, an appropriate choice of cipher E would be a tweakable block cipher [27] such as XEX [35]. A second option is to employ a stream cipher E. On decryption, portions of the keystream corresponding to missing blocks may simply be discarded.

3. Sentinel creation: Let f : {0, 1}j × {0, 1}* → {0, 1}l be a suitable one-way function. We compute a set of s sentinels {aw}sw=1 as aw = f(κ ,w). We append these sentinels to F′ ′ , yielding F′ ′ ′ .

4. Permutation: Let $g: \{0,1\}^j \times \{1,...,b' +s\} \rightarrow \{1,...,b' +s\}$ be a pseudorandom permutation (PRP). We apply g to permute the blocks of F′ ′ ′ , yielding the output file F ̃ In particular, we let F ̃[i] = F′ ′ ′ [g(κ , i)].

The **sentinel blocks** are randomly inserted into the encrypted data files for static data corruption detecting. Before encryption, the file is coded by an error correcting code and therefore can be recovered when the server response correctly in a high probability. Unfortunately, once used in the proof, the very part of sentinel blocks would be exposed. Therefore, this scheme can only handle a limited number of queries. [16]

A strongly counterintuitive aspect of this POR scheme is that the sentinels, which constitute the content of a POR proof, are generated independently of the bit-string whose retrievability they are proving. By contrast, in an ordinary proof of knowledge (POK), the content of a proof depends on the values that are the subject of the proof. The step of preprocessing / encoding of the data prior to storage imposes some computational overhead—beyond that of simple encryption or hashing—as well as larger storage requirements on the prover.

In 2008, Shacham & Waters introduced an improved version of PoRs scheme called Compact PoR with full proofs of security against arbitrary adversaries in the model of Juels and Kaliski. Their work consisted of 2 short, efficient homomorphic authenticators;
the first, based on pseudorandom functions (PRFs), gives a PoR scheme secure in the standard model. The second, based on BLS signatures, gives a PoR scheme with public verifiability secure in the random oracle model. Both schemes rely on homomorphic properties to aggregate a proof into one small authenticator value.

While Jules & Kaliski scheme offers a limited number of executions of the Challenge algorithm due to the number of precomputed sentinels embedded into the encoded file. [19](thus limited verification capabilities), Compact PoRs enables an unlimited number of queries which results to lower communication overhead.

Bowers, Juels and Oprea designed a new variant of the Juels-Kaliski scheme in 2009 that achieves lower storage overhead, tolerates higher error rates, and can be proven secure in a stronger adversarial setting. [3] At the same time, Dodis, Vadhan and Wichs formally proved the security of an (optimized) variant of the bounded-use scheme of Juels and Kaliski, without making any simplifying assumptions on the behavior of the adversary. Moreover, they built the first unbounded-use PoR scheme where the communication complexity is linear in the security parameter and which does not rely on Random Oracles, resolving an open question of Shacham and Waters. [4]

In "Fair and Dynamic Proofs of Retrievability" (2011), Zheng and Xu proposed the first dynamic POR scheme, while introducing ***fairness***, which is necessary property and also inherent to the setting of dynamic data because, without ensuring it, a dishonest client could legitimately accuse an honest cloud storage server of manipulating its data. Using two new tools, ***range-based 2-3 trees*** and an incremental signature scheme called ***hash-compress-and-sign*** (HCS for short), they presented FDPOR, a useful extension of the static POR, proven secure in random oracle. Unfortunately, the authors did not consider how to update the redundant encoded data and their scheme does not support public verifiability.

Correctness requires that, for all keypairs (pk, sk) output by Kg, for all files $M \in \{0, 1\}^*$, and for all (M*,t) output by St(sk,M), the verification algorithm accepts when interacting with the valid prover:

$$\left(\mathcal{V}(pk, sk, t) \rightleftharpoons \mathcal{P}(pk, t, M^*)\right) = 1 \ .$$

A proof-of-retrievability protocol is **sound** if any cheating prover that convinces the verification algorithm that it is storing a file M is actually storing that file, which we define in saying that it yields up the file M to an extractor algorithm that interacts with it using the proof-of-retrievability protocol.

An extractor algorithm Extr(pk,sk,t,P′) takes the public and private keys, the file tag t, and the description of a machine implementing the prover's role in the proof-of-retrievability protocol: for example, the description of an interactive Turing machine, or of a circuit in an appropriately augmented model. The algorithm's output is the file $M \in \{0, 1\}^*$. Note that Extr is given non–black-box access to P′ and can, in particular, rewind it.

A proof-of-retrievability scheme is ε-sound if there exists an extraction algorithm Extr such that, for every adversary A, whenever A, playing the setup game, outputs an ε-admissible cheating prover P′ for a file M, the extraction algorithm recovers M from P′ – i.e., Extr(pk, sk, t, P′) = M – except possibly with negligible probability.[2]

Based on the BLS signature, Shacham & Waters aggregate the proofs into a small value and their scheme can support public verifications. [7] However, using their scheme in dynamic scenario is impractical and insecure due to the following two reasons: First, its block signatures contain the indices of blocks. If a client deletes (or inserts) a block with index i, then any block with index j larger than i will have to change its index from j to j – 1 (or j + 1). So the client will need to re-sign all of the blocks whose indices have been changed, which makes this scheme impractical for supporting dynamic updates. Second, using the scheme in dynamic scenario cannot prevent replay attacks.

In order to achieve high error detection probability, the number of challenging data blocks is mainly determined by the fault tolerance rate of erasure coding employed. Although the scheme supported public auditability of static data using publicly verifiable homomorphic authenticators, how to perform data recovery was not explicitly discussed.

Based on the number of verifications they support over the lifetime of the system, we can classify PORs into two main types:

1) PORs that enable unlimited number of verifications, such as the SW [2] scheme, are usually constructed by storing additional integrity values for file blocks.
2) PORs that can verify a limited number of queries, such as the JK [1] scheme, usually pre-compute the responses to a set of challenges and embed them (encrypted) into the file encoding.

In 2012, Mo, Zhou & Chen with their paper "A Dynamic PoR Scheme with O(logn) Complexity" [7] achieved the best communication complexity by proposing a dynamic PoR based on a B+ tree and a merkle hash tree called **Cloud Merkle B+ tree (CMBT)** combined with the **BLS signature**. Their scheme can detect file corruptions with high probability even if the CSP tries to hide them. Moreover, our scheme is **able to support dynamic**

updates while keeps the same detection probability of file corruption. The attack model assumes a semi-trusted CSP.

The scheme uses the following algorithms:

KeyGen(1^k) → (pk; sk)

Prepare(sk; F'; Ftags) →(Φ; sigsk(v(R));CMBT)

GenChallenge(n) → Q

GenProof(Q;CMBT; F'; Ftags; Φ) → P

Verify(pk; Q; P; v(R)) → (TRUE; FALSE)

UpdateRequest() → Request

Update(F'; Ftags; Φ;R) → (Pold; Pnew)

UpdateVerify(Pold; Pnew) → (TRUE; FALSE)

The scheme consists of 3 stages:

1) Preprocess: the client will first encode the file F to F' using an erasure code. Then the client will run the algorithms KeyGen(1k) to create a pair of keys, and use Prepare(sk; F; Ftags) to generate a signature set Φ, a CMBT and the meta data sigsk(v(R))

For each node w in CMBT, six values are stored: *left(w), middle(w) and right(w), rank(w), t(w).*

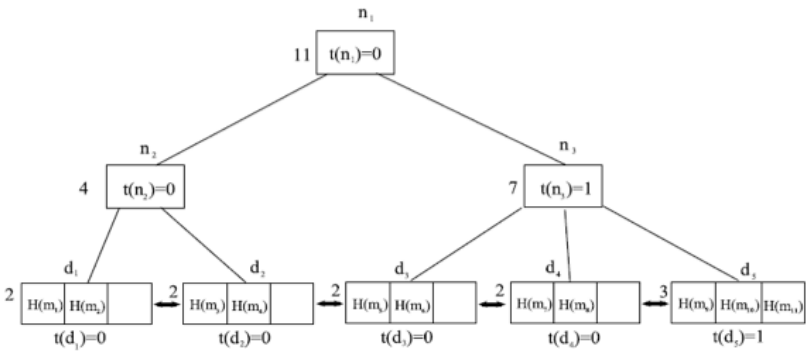


Figure 2. An example of a range-based 2-3 tree [7]

2) Verification: After receiving the proof P from the server, the client will run the algorithm Verify(pk; Q; P; v(R)) to check the integrity of blocks whose indices belong to the set I.

To compute the value of wj(1≤j≤h), the client first determines how many children the node wj contains. Then the client uses a function GetValue who takes the children's values and their location relationship p as inputs, and compute the value of wj

3) Update: client sends the server a request to update the file (update operations: modification, insertion, deletion)

In [8], Paterson et al. provide the first example of a keyed POR scheme with unconditional security , where an adversary has unlimited computational power. In this case retrievability of the file can be modelled as error-correction in a certain code.
Additionally, they show how classical statistical techniques can be used to evaluate whether the responses of the prover are accurate enough to permit successful extraction and prove a new lower bound on storage and communication complexity of POR schemes.

The **components** in a POR scheme and the extractor-based security definition used are the following [8]:

- The Verifier has a message m ∈ (Fq)k which he redundantly encodes as M ∈ (Fq)n.
- M is given to the Prover. In the case of a keyed scheme, the Prover may also be supplied with an additional tag, S.
- The Verifier retains appropriate information to allow him to verify responses. This may or may not include a key K.
- Some number of challenges and responses are carried out by the Prover and Verifier. In each round, the Verifier chooses a challenge c and gives it to the Prover, and the Prover computes a response r which is returned to the Verifier. The Verifier then verifies if the response is correct.
- The computations of the Prover are described in a proving algorithm P.
- The success probability of P is the probability that it gives a correct response when the challenge is chosen randomly.
- The Extractor is given P and (in the case of a keyed scheme) K, and outputs an unencoded message mˆ . Extraction succeeds if mˆ = m.
- The security of the POR scheme is quantified by proving a statement of the form "the Extractor succeeds with probability at least δ whenever the success probability of P is at least ε ". In this paper, we only consider schemes where δ = 1, that is, where extraction is always successful.

Introducing unconditional security, POR schemes are easier to understand and analyse because there is no use of any additional cryptographic primitives or unproven assumptions (e.g., PRFs, signatures, bilinear pairings, MACS, hitting samplers, random oracle model, etc.). Concluding, they perform a comprehensive analysis of the extraction properties of unconditionally secure POR schemes, and established a methodology that is applicable to the analysis of further new schemes. What constitutes "good" parameters for such a scheme depends on the precise application, but our framework allows a flexible trade-off between parameters. [8]

Albeshri, Boyd and Nieto in *"A Security Architecture for Cloud Storage Combining PoR and Fairness"* (2012) obtain the first secure storage cloud computing scheme that furnishes all three properties of availability, fairness and freshness, based on the previous works of Cloudproof (2010) and DPOR (2009) and considering the following entities: Data owner, cloud provider, clients, third-party auditor. [9] Since no mutual trust between parties is considered, several security properties need to be assured when storing the data in the cloud.

- Data Confidentiality
- Data Integrity
- Data Availability
- Public Verifiability
- Freshness
- Fairness

"Towards efficient PoRs" of 2012 [12], Xu and Chang incorporate a construction of constant size polynomial commitment scheme (Kate, Zaverucha and Goldberg, Asiacrypt '10) into Shacham and Waters scheme of private verifiability [2]. The resulting scheme is called EPOR and requires $O(\lambda)$ communication bits (particularly, 920 bits if a 160 bits elliptic curve group is used or 3512 bits if a 1024 bits modulo group is used) per verification and a factor of 1/s file size expansion. Experiment results show that our proposed scheme is indeed efficient and practical. The security proof is based on Strong Diffie-Hellman Assumption. In [2], the size of a response (or proof) is dominated by s group elements where each is λ bits long. They manage to aggregate these s group elements into two group elements, leading to a reduction in proof size from $O(s\lambda)$ bits to $O(\lambda)$ bits, by exploiting an intriguing property of polynomial, which is recently used by Kate, Zaverucha and Goldberg to construct a polynomial commitment. Combining with the result of Dodis, Vadhan and Wichs [4], which reduced the challenge size of SW scheme from $O(\lambda 2)$ to $O(\lambda)$, they reduce the communication cost per verification of SW scheme from $O(\lambda 2 + s\lambda)$ to $O(\lambda)$.

Their construction includes the following algorithms:

$KeyGen(1^{\lambda}) \rightarrow (pk, sk)$

$DEncode(sk, M) \rightarrow (id, M\hat{})$

$Prove(pk, id, M\hat{}, Chall) \rightarrow (y, \psi, \sigma)$

$Verify(sk, id, Chall, (y, \psi, \sigma)) \rightarrow$ *accept or reject*

O(logn) communicational cost is accomplished by Cash et al. in 2013 in their scheme called PORAM (Dynamic PoR via Oblivious RAM). The main idea is to split up the data into small blocks and redundantly encode each block of data individually, so that an update inside any data block only affects a few codeword symbols. Apart from ensuring freshness and retrievability running an efficient audit protocol for dynamic storage, PORAM also sets the server blind by concealing the access pattern using the algorithmic techniques of oblivious RAM.

In this construction, the client starts with data $M \in \Sigma^l$ which splits into small message blocks $M = (m_1,...,m_L)$ with $m_i \in \Sigma^k$ where the block size $k \ll l = Lk$ is only dependant on the security parameter. The client then applies an error correcting code $Enc : \Sigma^k \rightarrow \Sigma^n$ that can efficiently recover n/2 erasures to each message block individually, resulting in the value $C = (c_1, \ldots, c_L) \in \Sigma^{Ln}$ where $c_i = Enc(m_i)$. Finally, the client initializes an ORAM scheme with the initial data D = C, which the ORAM stores on the server in some clever privacy-preserving form, while keeping only a short local state at the client. [13]

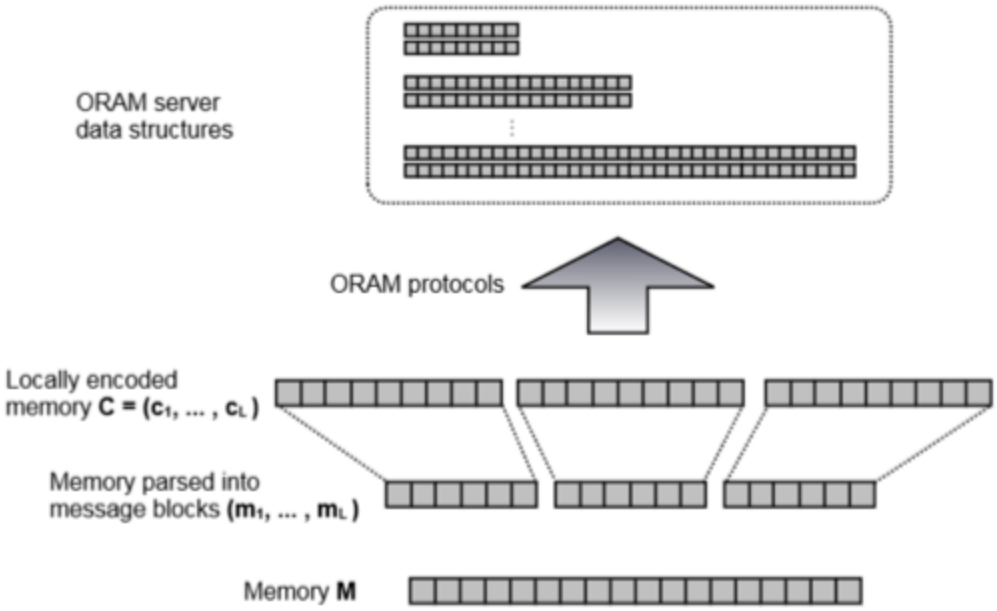


Figure 3. [13] Construction

Another contribution of the scheme is that PORAM introduces and achieves a new notion of security called next-read pattern hiding (NRPH), which considers an adversarial server that first gets to observe many read/write protocol executions performed sequentially with the client, resulting in some final client configuration C_{fin}. The adversarial server then gets to see various possibilities for how the "next read" operation would be executed by the client for various distinct locations, where each such execution starts from

the same fixed client configuration C_{fin}. The server should not be able to discern any relationship between these executions and the locations they are reading. [13]

The scheme constists of the following protocols between the stateful parties of client C and server S.

PInit(1^{λ}, 1^{w}, l)

OInit(1^{λ}, 1^{w}, l_{code})

PRead(i)

PWrite(i, v)

ORead(i_1,...,i_t)

OWrite(i_t,...,i_t;v_1,...,v_t)

Audit

PCPOR by Yuan and Yu published in 2013 [14] was the first POR scheme with public verifiability and constant communication cost, using techniques such as constant size polynomial commitment and homomorphic linear authenticators. The scheme contains the following algorithms:

• KeyGen: Given a selected security parameter λ, the randomized KeyGen algorithm outputs the system public key and private key as (PK, SK).
• Setup: Given a data file $M \in \{0, 1\}*$ and the public-private key pair (PK, SK), the Setup algorithm generates the encoded file M^ as well as the corresponding authentication tag σ, which will be stored on the server.
• Prove: Given the public key PK, encoded file M^ , authentication tag σ and a challenge message Chall, the Prove algorithm produces a proof response Prf.
• Verify: Given the public key PK and the Prf, the Verify algorithm checks the data integrity and outputs result as either accept or reject.

The scheme offers efficient computation performance and releases the data owner from being online in order to audit. It is also secure in the Computational Diffie-Hellman problem (CDH). [20]

In the 2013, “Efficient POR for FHM Data” considers efficiency of the POR scheme, when the data stored in cloud is encrypted under fully homomorphic encryption schemes. The scheme uses a new, efficient homomorphic authenticator, which enables constant-size responses and requests fairly little storage and computational power for the clients. Fully homomorphic encryption (FHE) allows performing permutations on encrypted data without decrypting them and therefore makes many flexible cloud applications possible.

Despite their advantages, the existing FHE schemes share a common flaw of message expansion: the plaintext is encrypted bit by bit, while each bit of plaintext corresponds to a ciphertext of a quite large size. As a result, existing POR schemes would become inefficient when being applied in a cloud storage system which protects its data with FHE scheme. Fully homomorphic encryption consists of four algorithms (Kengen; Enc; Dec; Eva). The additional algorithm Eva_{pk} $(C, \phi 1, ..., \phi t)$ takes as input an arbitrary Boolean circuit C as well as a set of ciphertexts ϕ ,..., ϕ , and outputs a new ciphertext ϕ . The property of homomorphism can be described as the correctness of the algorithm Eva, which guarantees the decryption of ϕ equals with the value for the circuit C being evaluated on the corresponding plaintexts. [16]

A proof of retrievability scheme should be both correct and sound. Correctness requires that, for all keypairs (pk, sk) output by Keygen, for all files $M \in \{0,1\}^*$, and M* output by St(sk, M), the verify algorithm always outputs 1 when interacting with a honest prover. Soundness, on the contrary, requires that any prover who convinces the algorithm Verify that it is storing a file M is actually storing file M and is guaranteed by a game-based security proof, under the security model of Juels and Kaliski's.

The construction includes 5 algorithms:

- Keygen(1^{λ})
- Store(M, sk)
- Challenge
- Prove(c, t, pk, M*)
- Verify(t, sk, proof)
-

Cloud Storage Retrievability Based On 3rd Party Audit (2013) is an authentication scheme of cloud storage security based on third party audit. This model could meet the user's demand of data integrity, data confidentiality, data recovery and extraction and TPA credibility control requirements. The trustiness of TPA is also guaranteed by a mutual authentication protocol. [17]

The framework is shown in figure. There are three entities in this framework: CS (Cloud Server), client and TPA (Third Party Auditor).

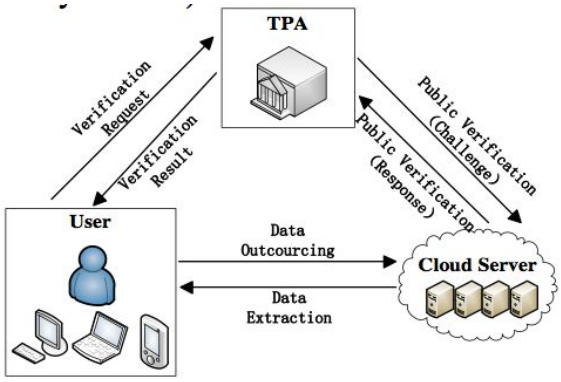


Figure 4. The authentication architecture based on TPA [17]

In addition, this scheme supports multiple servers. The original POR model and PDP model are based only on single server. But the cloud computing is composed of massive servers, so a multiple server environment is necessary. A multiple server support can improve the efficiency, strengthen the safety certification of security

1. *SETUP*
    a. *File Coding*
    b. *Parameter Setting*
    c. *Data Outsourcing*
2. *VERIFICATION*
    a. *User request for verification*
    b. *TPA generate Challenge*
    c. *Server generate proof*
    d. *TPA verify response*
3. *DATA RECOVERY AND EXTRACTION*

Credibility Control of TPA involves 3 aspects. First, the user's sensitive data can not be leaked to TPA, because TPA may leaks user data to an unauthorized or even malicious third party for personal interest. This scheme uses random masking technology based on the same state certification. Second, mutual authentication between user and TPA is been used in this scheme. The mutual authentication is achieved by using public key cryptography technology. Last, user needs to confirm the certified behavior of TPA. User delegates the authentication work to TPA, but TPA may not doing so exactly to reduce cost. To solve this problem, a parameter on the client is suggested in this paper to record the challenging time the user needs TPA to perform.

Shi, Stefanov and Papamanthou presented a dynamic PoR scheme in 2013 [21] with constant client storage $O(\beta \lambda)$, where β is the block size and λ is the security parameter. The scheme requires $\beta + O(\lambda^2 \log n)$ bandwidth for efficient audits and supports public

verifiability. Efficiency is achieved through incrementally constructible code based on Fast Fourier Transform (FFT), forming a butterfly network of levels, as seen in Figure 5.

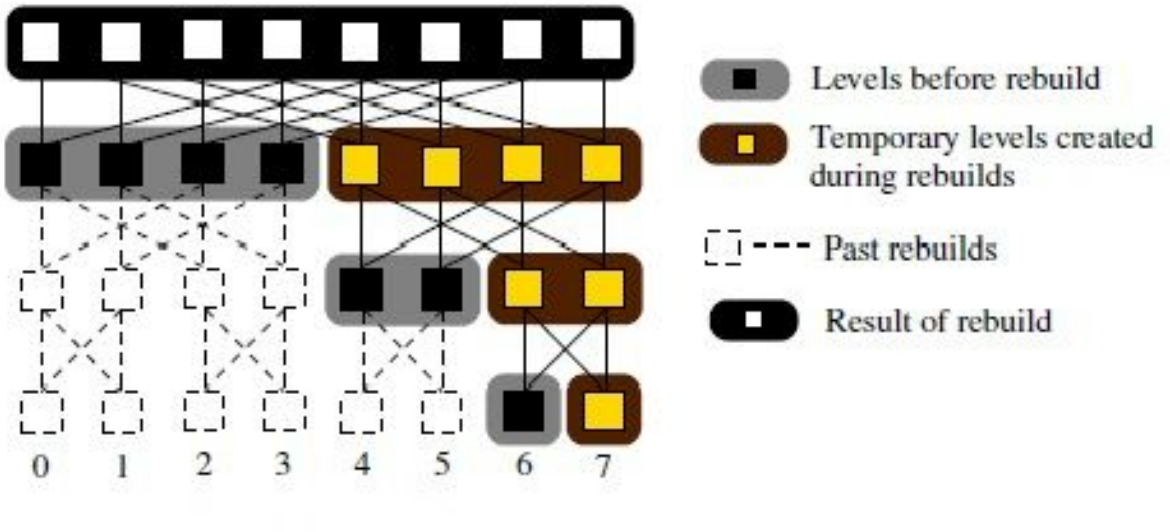


Figure 5. Butterfly network [21]

The level rebuild time achieved is improved compared to the oblivious sorting in ORAM schemes. FFT-based codes also offer implementation simplicity. The scheme idea is to have the server perform the computation on behalf of the client, thus significantly reducing the client computation and bandwidth. However, this concept does not cover all the trust models and does not specify the efficiency in an untrusted storage server scenario.

2014 DPOS, an hybrid Proof of Storage scheme observes that all existing publicly verifiable POS schemes suffer from a serious drawback: It is extremely slow to compute authentication tags for all data blocks, due to many expensive group exponentiation operations. [23] The scheme includes a 3rd party auditor and ensures data leakage prevention. Compared to existing publicly verifiable POS scheme, DPOS improves the authentication tag generation speed. Below the two system model definitions for DPOS**.**

(1) *A Delegatable Proofs of Storage (DPOS) scheme consists of three algorithms (KeyGen, Tag, UpdVK), and a pair of interactive algorithms ⟨P, V⟩.*

- KeyGen(1 λ) → (pk, sk, vpk, vsk)
- Tag(sk,vsk,F) → (ParamF,{(σ i,ti)})
- UpdVK(vpk,vsk,{ti}) → (vpk′ ,vsk′ ,{t′ i})
- ⟨ P(pk, vpk, {Fi }) , V(vsk, vpk, pk, ParamF) ⟩

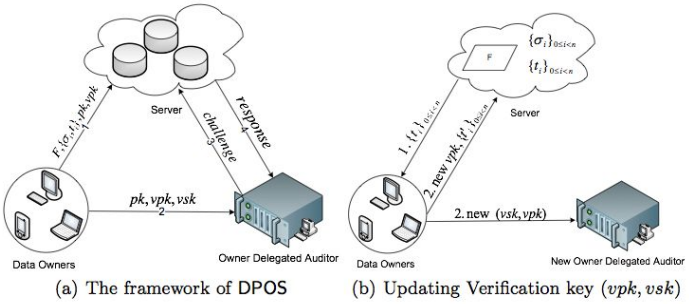


Figure 6. System model of DPOS [23]

(2) A DPOS system among three parties—data owner, cloud storage server and auditor, can be implemented by running a DPOS scheme (KeyGen, Tag, UpdVK,⟨P, V⟩) in the following three phases, where the setup phase will execute at the very begin- ning, for only once (for one file); the proof phase and revoke phase can execute for multiple times and in any (interleaved) order.

- *Setup phase*
- *Proof phase*
- *Revoke phase*

DPOS makes the following trust assumptions:

- The cloud storage server is trusted in data privacy and is not trusted in maintaining data integrity
- Owner-Delegated-Auditor is trusted in performing the delegated auditing task and protecting his/her verification secret key securely (integrity), but is not trusted in data privacy.

The scheme also investigates the scenarios of untrusted CSP and untrusted auditor.

OPOR scheme in 2014 introduces the notion of Outsourced proofs of retrievability (OPOR), in which users can task an external auditor to perform and verify POR with the cloud provider. [24] Evaluation results show that this proposal minimizes user effort, incurs negligible overhead on the auditor (compared to the basic [2] scheme), and considerably improves over existing publicly verifiable POR. Since OPOR extends POR, the authors first introduce POR, adapted from [2] and extended to a formal model for OPOR.

As previously mentioned, a POR scheme consists of four procedures [2]

- setup
- store
- verify, prove

Similar to the traditional POR model, an OPOR consists of a user U, the data owner, who plans to outsource his data M to a service provider S. In addition, U is interested in

acquiring regular proofs that his data is correctly stored and retrievable from S. To this end, an OPOR comprises a new entity A, called the auditor, who runs POR with S on behalf of U . An OPOR scheme comprises five protocols Setup, Store, POR, (that resemble and extend the POR scheme protocols), CheckLog, and ProveLog. One major difference is that the POR protocol not only outputs a decision on whether the POR has been correct, but also a log file which allows the user to *check* (using the CheckLog procedure) if the auditor did his job during the runtime of the OPOR scheme. As the purpose of OPOR is to incur less burden on the user, the verification of the logs by the user should incur less resource consumption on the user when compared to the standard verification of POR directly with the service provider S. Second, logs allow the auditor to *prove* (using the ProveLog procedure) that if some problems occur, e.g., the file is no longer stored by S, the auditor must not be blamed. In what follows, we detail each protocol in OPOR.

The scheme also incorporates the definition of correctness which requires that if all parties are honest, then the auditor always, i.e., with probability 1, accepts at the end of each POR protocol run and likewise the user at the end of each CheckLog protocol run. [24] Regarding the Security Model, OPOR does not consider confidentiality of the file M , but assumes that the user encrypts the file prior to the start the OPOR protocol. It also defines soundness if no honest party aborts (we call this ε -extractability) and soundness for the case that one honest party aborted (liability). Specifically:

- **ε -extractability:** If a scheme is secure with respect to a set of corrupted parties C, it automatically is secure with respect to any subset C' subnet of C. Hence, to show our claim it suffices to consider the three cases where exactly one party is honest.
- **Liability:** An honest auditor can prove the correctness of the log files with high probability while a misbehaving auditor will fail.

FORTRESS is a proposed efficient OPOR that requires that the auditor conducts two POR in parallel with the service provider. Fortress enables the user to efficiently verify in a single batch a number of conducted POR to verify the work of the auditor. This minimizes communication overhead while achieving the same level of security and efficiency as in [2]

To ensure correct parameter generation, Fortress relies on a sub-protocol which guarantees that the parameters computed by the auditor in the beginning have been correctly generated without revealing his secret parameters. Fortress leverages functionality from Bitcoin in order to provide a time-dependent source of pseudo-randomness to sample the parameters of the POR. [24]

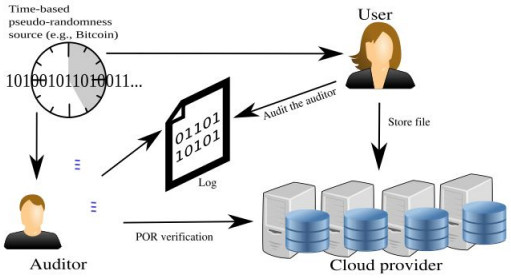


Figure 7. Fortress relies on a time-dependant source of pseudo-randomness [24]

Conclusively, Fortress minimizes communication overhead and incurs minimal overhead on the user. Additionally, it scales well with the number of users and increases the user trust while incurring minimal user interaction. Speedwise, Fortress is 2000 faster and 20% faster in store time than the public scheme [2] and it enhances performance and efficiency of PDP schemes.

More recently, Dynamic Proofs of Retrievability for Coded Cloud Storage Systems scheme proposed by Ren et al. [25], used network coding in addition to erasure codes for redundancy. By utilizing the network coding as the outer code and the erasure codes as the inner code, communication- efficient and effective data recovery can be realized.

The scheme splits up the data into small data blocks and encodes each data block individually using network coding before outsourcing so that an update inside any data block only affects a few codeword symbols and communication-efficient data repair for a breakdown server can be achieved. To eliminate the communication overhead for small data corruptions within a server, each encoded data block is further encoded via erasure codes. An *rb23Tree* is also used to organize the encoded blocks in the leaf nodes to enforce data access sequence and support cheat-proof data dynamic operations. To support public auditability, Ren et al. use the aggregated signature-based broadcast (ASBB) encryption scheme to generate metadata tags of the encoded blocks. The model exploits both within-server redundancy and cross-server redundancy to encode data blocks before outsourcing (with regenerating codes) , thus improving data reliability and availability.

6. Conclusion

By now it is understood that the standard Proof of Retrievability guarantees are storage authentication and retrievability. In detail, this means that an efficient PoR scheme not only ensures freshness and integrity of the outsourced data, which can easily be achieved by using MAC or Merkle trees, but also guarantees that the server is storing all of the client's data and performs efficient data recovery on corrupted data, due to redundancy.

In terms of practicality, which is yet to be efficiently addressed, a PoR scheme must be dynamic and support public verifiability with minimum overhead on the part of the client. Certain constant complexity or O(logn) complexity schemes presented in this survey meet those requirements, but are not being practically used yet. Besides, public verifiability schemes are extremely slow to compute authentication tags for all data blocks, due to many expensive group exponentiation operations.

Bounded-use schemes can often be significantly more efficient in terms of server storage, as they lack the pre-computation process. Intuitively approaching the privacy

matter of PoRs but also proven mathematically in schemes such as [13] , there is the usual tradeoff between privacy and efficiency. Practical PoR schemes are possible when access privacy is not required. As we have already seen, privacy is not a standard requirement of PoRs, and the mechanisms with which it can be achieved add an important overhead to the model. In order to make PoR schemes actually practical on part of the user or even incorporated in future SLAs, one must keep in mind all of the above, as well as optimizing the characteristics that affect the efficiency as seen in the comparative attempt on Table 2, such as the size n and the number β of data block which usually affects the computational cost. A multiple server support scenario like [17] can improve the efficiency and strengthen the security, providing a more realistic model.

Scheme	Technique	Advantages	Disadvantages
PoR for Large files	Sentinel-based scheme Error correcting code	Ensures both possession and Retrievability of files on archive service systems	Computationally cumbersome especially when the data to be encrypted is large. There will also be storage overhead at the server, partly due to the newly inserted sentinels and partly due to the error correcting codes that are inserted. Larger storage requirements on the prover. [1] can only handle limited number of queries
Compact PoRs	Homomorphic authenticators & BLS signatures	Unlimited number of queries and requires less communication overhead	Used only for static data How to perform data recovery is not explicitly discussed
PoRs theory & implementation	Spot-checking	Lower storage overhead, tolerates higher error rates, and can be proven secure in a stronger adversarial setting.	Used only for static data
PoR O(logn)	Cloud Merkle B+ tree , BLS signature	Possible to detect file corruptions with high probability even if the CSP tries to hide them. The scheme is able to support dynamic updates while keeps the same detection probability of file corruption. Worst case performance when compared with other schemes is O (logn).	Less efficient
Towards	Diffie-Hellma	Efficient and secure.	Only supports private verifiability

efficient PoRs	n Assumption security	It requires only a constant number of communication bits per verification.	
PoR via ORAM	ORAM & NRPH	strong privacy and authenticity guarantees	Server's storage remains linear in the size of the client data
Practical Dynamic PoRs	FFT-codes	efficient & implementable	access pattern privacy not guaranteed Not all attack models covered
3rd party audit	TPA	multiple servers	data block size affects computational cost

Table 1. Qualitative comparison of some of the most important PoRs

Table 2

Scheme	Static/ Dynamic	Public Verifiability	Communication Cost	Storage Cost (client)	Server Storage	Read Complexity	Write Complexity	Audit Complexity	Notes
PORs for Large Files [1]	S	No	O(1)	O(λ)					
Compact PoRs [2]	S	Yes	O(1)	O(λ)	|F| / m				|F|: data file size, m: number of sectors in each data block
PoRs Theory & Implementation [3]	S	Yes							
PoRs via Hardness Amplification [4]	S	Yes	O(1)	3λ + 80	(1 + 1/s) |F|				
Fair and Dynamic PoR [6]	D	No							
Dynamic PoR with O(logn) Complexity [7]	D	Yes	O(logn)						
A Security Architecture for Cloud Storage Combining PoR & Fairness [9]	D	Yes							+Freshness +Fairness
Towards efficient PoR [12]	D	No	O(1)	λ + |F|	|F| / s				|F|: data file size, s: block size, λ: security parameter
PoR via ORAM [13]	D	No	O(λ)	O(λ)	O(l)	O(λ * log^2 * l)	O(λ^2 * log^2 * l)	O(λ^2 * log^2 * l)	+ privacy
PCPOR Public and Constant-Cost PoRs [14]	D	Yes	O(1)	λ+4|G|	(1+1/s) |F|				
Practical Dynamic PoRs	D	Yes	O(βλlogn)	O(βλ)	O(βn)		O(β logn)	O(λ logn)	
Cloud Storage Retrievability Based On 3rd Party Audit [17]	D	Yes	O(logn)						multiple servers

References

1. A. Juels and B. S. K. Jr. “Pors: proofs of retrievability for large files”. In ACM Conference on Computer and Communications Security, pages 584–597, 2007.
2. H. Shacham and B. Waters. “Compact proofs of retrievability”. In ASIACRYPT, pages 90–107, 2008.
3. K. D. Bowers, A. Juels, and A. Oprea. “Proofs of retrievability: theory and implementation”. In CCSW, pages 43–54, 2009.
4. Y. Dodis, S. P. Vadhan, and D. Wichs. “Proofs of retrievability via hardness amplification”. In TCC, pages 109–127, 2009.
5. Zhu, Y. · Wang, H., et al. “Zero-knowledge proofs of retrievability”, in Science China Information Sciences (2011)
6. Q. Zheng and S. Xu. “Fair and dynamic proofs of retrievability”. In CODASPY, 2011.
7. Z. Mo, Y. Zhou, and S. Chen. “A dynamic proof of retrievability (por) scheme with o(log n) complexity”. In ICC’12, pages 912–916, 2012.
8. M. Paterson, D. Stinson, J. Udadhyay, “A coding theory foundation for the analysis of general unconditionally secure PoR schemes for cloud storage”, Journal of Mathematical Cryptology. Volume 7, Issue 3, Pages 183–216, ISSN (Online) 1862-2984, ISSN (Print) 1862-2976, DOI: 10.1515/jmc-2013-5002, September 2013
9. A. Albeshri, C. Boyd, J. Nieto, “A Security Architecture for Cloud Storage Combining PoR and Fairness”, CLOUD COMPUTING 2012 : The Third International Conference on Cloud Computing, GRIDs, and Virtualization
10. S. Liu, K. Chem, “Homomorphic linear authentication schemes from o-ASU2 functions for PoR”, Systems Research Institute, Polish Academy of Sciences, Control and Cybernetics 2012, Vol. 41, no. 2, 335-351
11. Wang Shao-hu, Chen Dan-we, Wang Zhi-weiP, Chang Su-qin, “Public auditing for ensuring cloud data storage security with zero knowledge Privacy” College of Computer, Nanjing University of Posts and Telecommunications, China, 2009
12. Xu, J., Chang, E.C. “Towards efficient proofs of retrievability”. In: ASIACCS’12. pp. 79–90. ACM (2012)
13. D. Cash, A. Küpcü, and D. Wichs. “Dynamic proofs of retrievability via oblivious ram”. In Eurocrypt, 2013.
14. Jiawei Yuan and Shucheng Yu. “Proofs of retrievability with public verifiability and constant communication cost in cloud”. In Proceedings of the 2013 international workshop on Security in cloud computing, Cloud Computing ’13, pages 19–26, New York, NY, USA, 2013. ACM.
15. Z. Mo, Y. Zhou, and S. Chen, “An Efficient Dynamic Proof of Retrievability Scheme,” ZTE Communications, vol. 2, p. 008, 2013.
16. Y. Guang, Y. Zhu, C. Gu, Y. Zheng and J. Fei, "An efficient proof of retrievability scheme for fully homomorphic encrypted data", Journal of networks, vol. 8, No. 2, pp. 339-344, 2013. (2013)
17. Qin Zhongyuan, Song Yunyan, Zhang Qunfang, Huang Jie “Cloud Storage Retrievability Based On Third Party Audit”, CCIS-13, Advances in Intelligent Systems Research (October 2013)

18. X Song, H Deng, “Lightweight Proofs of Retrievability for Electronic Evidence in Cloud” - EBSCO Information (2078-2489), 2013
19. B Wang, X Hong, “Multi-file PoRs for cloud storage auditing” - IACR Cryptology ePrint Archive, 2013
20. J Yuan, S Yu, “PCPOR Public and Constant-Cost PoRs in Cloud” - Journal of Computer Security, 2015
21. E. Shi, E. Stefanov, and C. Papamanthou. “Practical dynamic proofs of retrievability”. Technical report, 2013.
22. Han, S., Liu, S., Chen, K., Gu, D.: “Proofs of data possession and retrievability based on MRD codes”. IACR Cryptology ePrint Archive, Report 2013/789 (2013)
23. Jia Xu, Anjia Yang, Jianying Zhou and Duncan S. Wong, “Lightweight and Privacy-Preserving Delegatable Proofs of Storage”, Institute for Infocomm Research, Singapore. (2014)
24. Armknecht, F., Bohli, J.M., Karame, G.O., Liu, Z., Reuter, C.A. “Outsourced proofs of retrievability”. In: ACM CCS, pp. 831–843 (2014)
25. Z. Ren, L. Wang, Q. Wang, M. Xu, “Dynamic Proofs of Retrievability for Coded Cloud Storage Systems”, IEEE Transactions on Services Computing (Volume:PP , Issue: 99), September 2015