Poster: Proofs of Retrievability with Low Server Storage

Michael Hanling

mikehanling@gmail.com

United States Naval Academy
Annapolis, Maryland, United States

Gaspard Anthoine

Gaspard.Anthoine@etu.univ-grenoble-alpes.fr

Laboratoire Jean Kuntzmann
Grenoble, France

Jean-Guillaume Dumas

Jean-Guillaume.Dumas@univ-grenoble-alpes.fr

Laboratoire Jean Kuntzmann
Grenoble, France

Aude Maignan

Aude.Maignan@univ-grenoble-alpes.fr

Laboratoire Jean Kuntzmann
Grenoble, France

Clément Pernet

clement.pernet@univ-grenoble-alpes.fr

Laboratoire Jean Kuntzmann
Grenoble, France

Daniel S. Roche

roche@usna.edu

United States Naval Academy
Annapolis, Maryland, United States

ABSTRACT

Proof of Retrievability (PoR) and Provable Data Possession (PDP) schemes have been proposed to ensure the integrity of stored data on untrusted servers. A successful PoR audit ensures, with high probability, that every piece of stored data is recoverable by the server. Most PoR schemes proposed have focused on bandwidth and computation cost, but in some deployment scenarios the *size of remote storage* can be the most expensive factor.

We propose a simple PoR scheme which is a variant on some existing PDP work. Compared to existing audit routines, the server computation cost and bandwidth are higher, but the server storage cost is minimal. Our preliminary work indicates that deploying this scheme may be less costly in commercial cloud settings, depending on the cost structure and frequency of audits.

CCS CONCEPTS

• Security and privacy → Database and storage security.

KEYWORDS

Proof or retrievability, data integrity

ACM Reference Format:

Michael Hanling, Gaspard Anthoine, Jean-Guillaume Dumas, Aude Maignan, Clément Pernet, and Daniel S. Roche. 2019. Poster: Proofs of Retrievability with Low Server Storage. In *2019 ACM SIGSAC Conference on Computer and Communications Security (CCS'19)*, November 11–15, 2019, London, United Kingdom. ACM, New York, NY, USA, 3 pages. <https://doi.org/10.1145/3319535.3363266>

1 INTRODUCTION

Cloud storage has become the common answer to store and maintain large data repositories. However, this solution presents a new problem to answer: security of one's data without physical access to the server. Though the client believes that the Cloud server will protect the integrity of its data, Cloud providers do not provide any mechanism to verify this. A typical user may not know if their data has been lost or corrupted until they need to use it (if even then).

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

CCS '19, November 11–15, 2019, London, United Kingdom

© 2019 Copyright held by the owner/author(s).

ACM ISBN 978-1-4503-6747-9/19/11.

<https://doi.org/10.1145/3319535.3363266>

Two closely-related lines of work, Proof of Retrievability (PoR) and Provable Data Possession (PDP), provide mechanisms to *audit*, or check the integrity of, data at rest. The audit process is an interactive protocol whereby the client gains some probabilistic guarantee that their data has not been lost or modified. The various protocols proposed in the literature vary in terms of:

- (1) the strength of the probabilistic guarantee,
- (2) the amount of communication (*bandwidth*) per audit,
- (3) the amount of server and client computation per audit, and
- (4) the amount of persistent overhead storage incurred.

In this preliminary investigation, we present and evaluate a protocol which trades increased server computation (3) and bandwidth (2) for decreased persistent storage (4).

1.1 Prior Work

PDP schemes, first introduced by [1] in 2007, audit data by randomly selecting a fraction of storage blocks to verify, typically using some hash-based methods. The original scheme was later adapted to allow dynamic updates by [3] and has since seen numerous performance improvements. However, PDPs only guarantee (probabilistically) that a *large fraction* of the data was not altered; a single block deletion or alteration is likely to go undetected in an audit.

PoR schemes, first introduced at the same CCS conference in 2007 by [5], provide a stronger guarantee of integrity: namely, that any small alteration to the data is likely to be detected by an audit. Later improvements by [2, 8, 9] allow for dynamic updates and reduce the asymptotic complexity. However, these techniques rely on computationally-intensive tools, such as locally decodable codes and Oblivious RAM (ORAM), and incur at least a 1.5x, or as much as 10x, overhead on the size of remote storage.

1.2 Our Contributions

We present a simple protocol which supports dynamic updates, has the stronger integrity guarantees of PoRs, but incurs only minimal additional overhead in server storage. Specifically, our scheme uses only $n + o(n)$ persistent server storage for an n -bit database, as opposed to the $O(n)$ storage in previous PoR schemes.

The disadvantage of our approach is that the server computation time and bandwidth are $O(n)$ and $O(\sqrt{n})$ respectively, as opposed to the state of the art in PoR which are closer to $O(\log n)$. However, our preliminary experiments indicate that *the tradeoff of increased computation for decreased persistent storage is advantageous in cloud deployments*. Specifically, performing an audit every three hours

with our protocol, from our tests, incurs less cost than that of using a PoR with 2x persistent storage overhead.

2 MATRIX BASED APPROACH

The basic premise of our particular PoR is to treat the data, consisting of n machine words, as a square matrix $\mathbf{M} \in \mathbb{R}^{m \times m}$, where $m = \lceil \sqrt{n} \rceil$ and $\mathbb{R}$ is a suitable finite ring. Crucially, the choice of ring $\mathbb{R}$ detailed below does not require any modification to the raw data itself; that is, any element of the matrix $\mathbf{M}$ can be retrieved in $O(1)$ time. At a high level, our audit protocol follows the matrix multiplication verification technique of [4].

At initialization time, the client chooses a random vector $\mathbf{r} \in \mathbb{R}^m$ and computes a second secret vector $\mathbf{s} \in \mathbb{R}^m$ according to

$$\mathbf{s}^\top = \mathbf{r}^\top \mathbf{M}. \quad (1)$$

Later, to perform an audit, the client chooses a random challenge vector $\mathbf{c} \in \mathbb{R}^m$, and asks the server to compute a response vector $\mathbf{v} \in \mathbb{R}^m$ according to

$$\mathbf{v} = \mathbf{M}\mathbf{c}. \quad (2)$$

Upon receiving the response $\mathbf{v}$, the client checks two dot products for equality, namely

$$\mathbf{s}^\top \mathbf{c} \stackrel{?}{=} \mathbf{r}^\top \mathbf{v}. \quad (3)$$

The proof of retrievability relies on the fact that observing m successful audits allows, with high probability, recovery of the matrix $\mathbf{M}$, and therefore the entire database.

In addition to the original database, logically treated as the matrix $\mathbf{M}$, the server also stores a Merkle tree in which every leaf node is labelled with the hash of a data block, and every non-leaf node is labelled with the cryptographic hash of the labels of its child nodes [6]. This blocking is independent of that used for $\mathbf{M}$, and, if using a block size of $\Omega(\log n)$, the total size of the Merkle tree is $O(n/\log n)$, which is $o(n)$ as required.

Table 1 lists the asymptotic complexity of this approach.

Table 1: Complexity Analysis of Matrix Based PoR

	Init	Audit	Update
Client Work	$O(n)$	$O(\sqrt{n})$	$O(\log n)$
Communication	$O(n)$	$O(\sqrt{n})$	$O(\log n)$
Server Work	$O(n)$	$O(n)$	$O(\log n)$
Client Storage		$O(\sqrt{n})$	
Server Storage		$n + O(n/\log n)$	

2.1 Operations Provided

2.1.1 Initialization. At initialization, the client computes $\mathbf{r}$ and $\mathbf{s}$ as above and also computes the root hash of the Merkle tree. The server computes and stores the entire Merkle hash tree.

2.1.2 Audit. The client initiates an audit by generating a fresh random challenge vector $\mathbf{c}$ and sending this to the server. The server computes the matrix-vector product $\mathbf{v} = \mathbf{M}\mathbf{c}$ and sends this back, and finally the client checks the two dot products from (3) above. If these dot products are equal, the client knows with high probability that the server can recover the entire database.

The audit protocol's cost is mostly in the server's matrix-vector product. The client's dot products are much cheaper in comparison. The communication cost is bounded by $O(\sqrt{n})$ as each vector has $\sqrt{n}$ values. We trade this infrequent heavy computation for no additional persistent storage, justified by the significantly cheaper cost of computation versus storage space.

A similar technique was used by [7] for checking integrity. However, their security relies on the difficulty of integer factorization. Implementation would therefore require many modular exponentiations at thousands of bits of precision. Our approach is much simpler and independent of computational hardness assumptions.

2.1.3 Retrieve. While the audit operates on the data in 8-byte chunks, retrieving data is done at byte level with support for retrieving any range of bytes (that is legal with the size of the data). The Merkle tree structure is used here to ensure that the server sends an honest answer back, incurring only $O(\log n)$ overhead.

2.1.4 Update. Dynamic updates are supported in our PoR scheme and can be performed in the same time as a retrieval. Updating is performed at the byte level with support for any range of byte within the data, and the logarithmic overhead comes from updating a path of the Merkle hash tree.

The client must also update her copy of the secret vector $\mathbf{s}$, but this is possible in constant time due to the mechanics of matrix multiplication. The server need only update the byte value in the data, and send back the old byte value that was in that index. The client can then compute the difference between the old and new values, and then update the secret vector $\mathbf{s}$ accordingly. Only a single entry of $\mathbf{s}$ is affected since only a single column in the data matrix is affected: $s'_j \leftarrow s_j + (m'_{i,j} - m_{i,j}) \cdot r_i$.

2.2 Two Prime Calculations

The protocol description above treats the database as a square matrix over a (finite) ring $\mathbb{R}$. In order to avoid any conversion which might incur a $O(n)$ overhead in server space, we break the data into word-size (8 byte) blocks, and choose the ring $\mathbb{R} = \text{GF}(p) \times \text{GF}(q)$ as the direct product of two finite fields, each of large prime order. If $pq \geq 2^{64}$, this ensures unique recovery of the database from images in $\mathbb{R}$ via Chinese remaindering, and also allows efficient computation without extended precision.

For the experiments below, we chose $p = 2^{31} - 1$ and $q = 2^{36} - 5$. That p is a Mersenne prime makes computations with it particularly efficient, but a second Mersenne prime of similar size does not exist.

This causes all computations to be performed twice, but at high efficiency, as we demonstrate in the next section, and (importantly) no increase in server storage.

3 EXPERIMENTAL RESULTS

To see how our theoretical design works in practice, we first focused on the audit protocol. Our costs of initialization, retrieval, and update are comparable to the state of the art in PoR protocols, but our audit algorithm is much more costly, linear-time rather than poly-logarithmic.

Rather than directly comparing the time for audit to existing PoR or PDP protocols, where our $O(n)$ time will scale much more

quickly, we instead wanted to address the question of how “heavy” the hidden constant in the $O(n)$ is.

For this, we compared the cost of performing a single audit, on databases of various sizes, to the cost of computing a cryptographic checksum of the entire database using MD5 or SHA256. In a sense, a cryptographic checksum is another means of integrity check that requires no extra storage, albeit without the malicious server protection that our PoR protocol provides. Therefore, having an audit cost which is comparable to that of a cryptographic checksum indicates the $O(n)$ theoretical cost is not too heavy in practice.

3.1 Experimental Design

The experiment was run between two VMs created on Google Cloud’s Compute Engine. The client machine was a basic f1-micro instance, 1 vCPU with 0.6GB memory. The server machine was an n1-standard-2, 2 vCPU with 7.5 GB memory. The client and server processes communicated over TCP connection. The data itself was stored on an attached 1.2TB standard persistent disk. Test files of size 100MB, 1GB, 10GB, 100GB, and 1TB were generated with random bytes from `/dev/urandom`. The function used to compute the run times throughout the experiment was `/usr/bin/time`.

The experiment took place in 4 stages. First, each file was run through the initialization protocol. Then, each file was run through the audit protocol. Third, each file an MD5 digest was calculated for each file. Finally, a SHA256 digest was computed for each file. A Merkle tree was also created over each file. Instead of scaling the block size to each file, a block size of 8KiB was chosen for practical performance. The results are organized into Table 2.

Per operation, the timings report the CPU time from the server side, and the total wall time from the client side. The difference is due mostly to I/O overhead; even for the audit, the client-side work to compute the two dot products is minimal.

Table 2: Run Time Test Results (seconds)

Operation		1GB	10GB	100GB	1TB
Init	Server	0.81	47.38	483.16	5236.65
	Wall	11.17	102.49	1055.46	11437.00
Audit	Server	5.53	54.26	463.25	5510.80
	Wall	12.75	117.36	1080.66	11495.00
MD5	Server	2.46	23.92	251.31	2848.21
	Wall	8.91	88.47	914.91	9234.00
SHA256	Server	6.30	62.67	639.08	6428.22
	Wall	11.49	112.72	1553.74	11969.00

3.2 Conclusions

There are two main conclusions to draw from the experiments. The first deals with the audit protocol following the theoretical bounds that were expected, and the second deals with how the run time compares to that of the hash functions.

Because the server computation time for an audit is $O(n)$, we expect the times to scale linearly, and our results support this. The large disparity between the Server Time and the Wall Time was not predicted in the theory. This disparity is found to be I/O based, caused by the CPU waiting for the reads to the external drive.

When analyzing the performance in comparison to MD5 and SHA256, the Wall time is best comparison as it is the time the client waits for a response. As shown by the table, the run time of the audit protocol falls between MD5 and SHA256. This justifies our hope that the $O(n)$ time audit protocol, while more costly than other PoR and PDP schemes, is comparable to that of computing a cryptographic checksum.

Again, we emphasize that the main benefit of our approach is the vastly decreased persistent storage compared to existing PoR schemes. Including the Merkle tree with block size of 8KiB, the total storage overhead is only 1.0684x file size. Using Google Cloud with a 1TB database, the cost for a 1.069TB Standard Persistent Disk per month is \$42.76. The cost for a single audit on a standard 2-core VM is \$0.14498. Running our audit protocol every three hours, would still be financially favorable compared to using a PoR scheme with just 2x storage overhead.

4 ONGOING WORK

Work is being being to decrease the disparity between the Server and Wall Times. Since the data is always read sequentially, parallelizing this process is a source of optimization. Testing shows better results when using the Message Passing Interface (MPI) to distribute reads over multiple, cheaper 1 core VMs than when parallelizing between multiple cores on one, more expensive VM.

Another area of potential improvement is considering the matrix dimensions. As of now, the audit protocol described treats the data as a square $m \times m$ matrix. However, by imposing instead the logical structure of a $m_1 \times m_2$ matrix, with $m_1 < m_2$, provides the possibility of achieving decreased communication during audits, at the expense of increased client-side computation with the dot products. Finding the right balance here will depend on the relative costs involved.

REFERENCES

- [1] Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary Peterson, and Dawn Song. 2007. Provable Data Possession at Untrusted Stores. In *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS '07)*. ACM, New York, NY, USA, 598–609. <https://doi.org/10.1145/1315245.1315318>
- [2] David Cash, Alptekin Kupcu, and Daniel Wichs. 2012. *Dynamic Proofs of Retrievability via Oblivious RAM*. Technical Report 550. <http://eprint.iacr.org/2012/550>
- [3] C. Chris Erway, Alptekin Küpcü, Charalampos Papamanthou, and Roberto Tamassia. 2015. Dynamic Provable Data Possession. *ACM Trans. Inf. Syst. Secur.* 17, 4, Article 15 (April 2015), 29 pages. <https://doi.org/10.1145/2699909>
- [4] Rūsiņš Freivalds. 1979. Fast Probabilistic Algorithms. In *Mathematical Foundations of Computer Science 1979 (LNCS)*, J. Bečvář (Ed.), Vol. 74. Springer-Verlag, Olomouc, Czechoslovakia, 57–69. https://doi.org/10.1007/3-540-09526-8_5
- [5] Ari Juels and Burton S. Kaliski. 2007. PORs: proofs of retrievability for large files. In *Proceedings of the 14th ACM conference on Computer and communications security - CCS '07*. ACM Press, Alexandria, Virginia, USA, 584. <https://doi.org/10.1145/1315245.1315317>
- [6] Ralph C. Merkle. 1988. A Digital Signature Based on a Conventional Encryption Function. In *Advances in Cryptology — CRYPTO '87*, Carl Pomerance (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 369–378.
- [7] F. Seb, J. Domingo-Ferrer, A. Martinez-Balleste, Y. Deswarte, and J. Quisquater. 2008. Efficient Remote Data Possession Checking in Critical Information Infrastructures. *IEEE Transactions on Knowledge and Data Engineering* 20, 8 (Aug 2008), 1034–1038. <https://doi.org/10.1109/TKDE.2007.190647>
- [8] Hovav Shacham and Brent Waters. 2013. Compact Proofs of Retrievability. *J. Cryptol.* 26, 3 (July 2013), 442–483. <https://doi.org/10.1007/s00145-012-9129-2>
- [9] Elaine Shi, Emil Stefanov, and Charalampos Papamanthou. 2013. Practical Dynamic Proofs of Retrievability. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security (CCS '13)*. ACM, New York, NY, USA, 325–336. <https://doi.org/10.1145/2508859.2516669>