

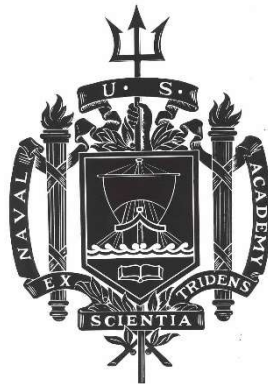
A TRIDENT SCHOLAR PROJECT REPORT

NO. 494

Proofs of Retrievability with Low Server Storage

by

Midshipman 1/C John M. Hanling, USN



UNITED STATES NAVAL ACADEMY
ANNAPOLIS, MARYLAND

This document has been approved for public
release and sale; its distribution is unlimited.

USNA-1531-2

REPORT DOCUMENTATION PAGE				Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.					
1. REPORT DATE (DD-MM-YYYY) 7/6/20		2. REPORT TYPE		3. DATES COVERED (From - To)	
4. TITLE AND SUBTITLE Proofs of Retrievability with Low Server Storage				5a. CONTRACT NUMBER	
				5b. GRANT NUMBER	
				5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S) Hanling, John M.				5d. PROJECT NUMBER	
				5e. TASK NUMBER	
				5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)				8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) U.S. Naval Academy Annapolis, MD 21402				10. SPONSOR/MONITOR'S ACRONYM(S)	
				11. SPONSOR/MONITOR'S REPORT NUMBER(S) Trident Scholar Report no. 494 (2020)	
12. DISTRIBUTION / AVAILABILITY STATEMENT This document has been approved for public release; its distribution is UNLIMITED.					
13. SUPPLEMENTARY NOTES					
14. ABSTRACT We investigate a novel approach to Proofs of Retrievability (PoR), protocols that allow a client to audit the cloud server storing its data remotely. These protocols allow a means of efficiently ensuring that all of the data the client believes to be stored in the cloud is still able to be retrieved by the cloud server, instead of relying on trust alone in the current model. Past PoR approaches have worked toward computational optimization for the audit; however, this requires a large amount of overhead persistent storage (up to 10x the actual database size). Our new approach instead trades higher computation for significantly decreased persistent storage. As all major cloud providers charge markedly more for storage than for computation, our new protocol offers practical efficiency. Our approach rests on treating the data as a square matrix, comparing randomized linear algebra identity tests over the matrix at the time of last check and at the current time. Honest retrieval of data, enforced through a Merkle hash tree requiring negligible extra persistent storage, and dynamic updates are supported in our approach. While audit computation now scales linearly, the required persistent storage is only 1.068x the size of the data. We demonstrate its efficiency in practice with a deployment on Google Cloud Compute Engine with test case data size of 1TB. Our approach costs \$42.72 per month for storage, and an audit costs \$0.23 taking 16 minutes. Previous state of the art requiring 6x storage of the data size costs \$240 per month. This is a 82% cost savings from storage while hosting the data in the cloud. We parallelized the computation of the audit across multiple virtual machines using MPI in order to increase the I/O-bound run time performance, which resulted in a near-linear speed up. We are investigating further optimizations on client-side storage and communication costs, as well as how to deploy our approach over an entire block device.					
15. SUBJECT TERMS Cloud storage, data integrity, audit protocol					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES 51	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			19b. TELEPHONE NUMBER (include area code)

**PROOFS OF RETRIEVABILITY
WITH LOW SERVER STORAGE**

by

Midshipman 1/C John Michael Hanling
United States Naval Academy
Annapolis, Maryland

(signature)

Certification of Adviser Approval

Associate Professor Daniel S. Roche
Computer Science Department

(signature)

(date)

Acceptance for the Trident Scholar Committee

Professor Maria J. Schroeder
Associate Director of Midshipman Research

(signature)

(date)

Abstract

We investigate a novel approach to Proofs of Retrievability (PoR), protocols that allow a client to audit the cloud server storing its data remotely. These protocols allow a means of efficiently ensuring that all of the data the client believes to be stored in the cloud is still able to be retrieved by the cloud server, instead of relying on trust alone in the current model. Past PoR approaches have worked toward computational optimization for the audit; however, this requires a large amount of overhead persistent storage (up to 10x the actual database size). Our new approach instead trades higher computation for significantly decreased persistent storage. As all major cloud providers charge markedly more for storage than for computation, our new protocol offers practical efficiency. Our approach rests on treating the data as a square matrix, comparing randomized linear algebra identity tests over the matrix at the time of last check and at the current time. Honest retrieval of data, enforced through a Merkle hash tree requiring negligible extra persistent storage, and dynamic updates are supported in our approach. While audit computation now scales linearly, the required persistent storage is only 1.068x the size of the data. We demonstrate its efficiency in practice with a deployment on Google Cloud Compute Engine with test case data size of 1TB. Our approach costs \$42.72 per month for storage, and an audit costs \$0.23 taking 16 minutes. Previous state of the art requiring 6x storage of the data size costs \$240 per month. This is a 82% cost savings from storage while hosting the data in the cloud. We parallelized the computation of the audit across multiple virtual machines using MPI in order to increase the I/O-bound run time performance, which resulted in a near-linear speed up. We are investigating further optimizations on client-side storage and communication costs, as well as how to deploy our approach over an entire block device.

Keywords: Cloud storage, data integrity, audit protocol

Acknowledgements

I would first and foremost like to thank my Trident Advisor, Dr. Daniel Roche. I have learned more from him than I could have imagined; it is because of his mentorship that I feel I have truly earned my degree. I thank all of my external advisors from the Laboratoire Jean Kuntzmann at the Université Grenoble Alps in France: Professor Jean-Guillaume and Associate Professors Gaspard Anthoine, Aude Maignan, and Clément Pernet. The work we present is truly a team effort, and I have learned much from our conversations and collaboration regarding this work. I thank my roommates, two brothers who have put up with the brightness of my monitors and sound of my keyboard through the night far too many times without complaint. Their support, whether they know it or not, allowed me to accomplish all that I have. I would also like to thank all of the teachers that I had in my life, from grade school through my time at USNA. Their guidance in academics and life have shaped me into the person that I am; without the help I have received along the way, I would not have risen to the caliber expected of the Trident Scholar Program. My parents and siblings have supported me in all of my endeavors, and I thank them for giving me the strong foundation that I have fallen back on many times. To all those that have had to approve my acceptance and/or progression in this program, your endorsement does not go unappreciated. Finally, I thank my Drum & Bugle Corps family. They have given me the outlet I loved, the support I wanted, and the challenge I needed.

Contents

1	Introduction	7
1.1	Motivation	8
1.2	Report Organization	9
2	Previous Work	10
2.1	Naive Solution	10
2.2	Provable Data Possession	10
2.3	Proof of Retrievability	12
2.4	Dynamic PoR	13
3	Security Model	15
3.1	Our Security Model	15
3.2	Correctness	16
3.3	Authenticity	16
3.4	Retrievability	17
4	Our Contribution	18
4.1	Overview of Matrix-Based Approach	18
4.2	Initialization	19
4.3	Audits	21
4.4	Merkle Tree Retrieval and Updates	24
4.4.1	Merkle Tree Protocols	24
4.4.2	Updates	25
5	Implementation	27
6	Experimentation	29
6.1	Parameter selection	29
6.2	Two Prime Calculations	30
6.3	Experimental Design	30
6.4	Audit compared to checksums	31
6.5	Parallel audits using MPI	33
6.6	Communication and Client Computation Time	34

7	Use-Based Optimizations	36
7.1	Structured Vectors	36
7.2	Non-Square Matrix Dimensions	36
7.3	Externalization	37
8	Comparison to State of the Art	38
8.1	Low storage overhead	39
8.2	Fast audits but large extra storage	39
9	The Way Ahead	42
9.1	Applications	42
9.2	Future Work	42
	References	44
A	Formal Security Proof	47
A.1	Formal Protocol Descriptions	47
	A.1.1 Ensuring security assumptions via multiple checks	47
A.2	Security Proof	47

List of Figures

1	Relavent Run Time Complexities Compared. From Most Efficient to Least: $O(N)$ - Linear, $O(\sqrt{N})$ - Square Root, $O(\log N)$ - Log, $O(1)$ - Constant	11
2	Creation of one MAC through Hashing the Concatenation of a Key and a Data Chunk	12
3	Data Converted to 4×4 Matrix	20
4	Random and Secret 1×4 Row Vectors Constructed by Client	21
5	Summary of Storage after Initialization	21
6	Challenge 4×1 Column Vector Constructed Randomly by Client	22
7	Result 4×1 Column Vector Calculated by Server	22
8	Final Client Side Computation for Audit Process. In the case of this example, $\mathbf{s}^\top \mathbf{v}$ does equal $\mathbf{r}^\top \mathbf{v}$, which means that the server passes the challenge.	23
9	Proof that Proposed Audit Check Is Valid	23
10	Updating Secret Vector	26
11	Updating Secret Vector Efficiently	27
12	Run Times of the Audit Protocol for Varying File Sizes (Log-Log Scale)	32
13	Relative Audit Time Improvements with MPI Parallelization (Log-Log Scale)	34

List of Tables

1	Client/server PoR protocol with low storage server	19
2	Proof of retrievability via square matrix verifiable computing	30
3	Run Time Test Results (seconds)	31
4	Multiple Machine Parallelization Results	33
5	Amount of Communication Per Audit with 16 VMs	34
6	Attributes of some selected schemes	38
7	S-PDP on f blocks : The file M is composed of N/b blocks of bit-size b . The computation is made mod Q , where Q is the product of two large prime numbers.	39
8	Storj-V2: The file M is composed of N/b blocks of bit-size b . K is the number of salts.	40
9	Shi et al. [1]: The file M is composed of $\frac{N}{b}$ blocks of bit-size b	40
10	Comparison with Shi et al. [1]: The audit complexity depends on the number of blocks.	41
11	Externalized PoR	51

1 Introduction

As computers have gained computing power, there is always a concern for how to best store data. Individuals used to store all of their data on the hard disk of a computer, or potentially on an external drive. Large companies would need to invest in their own servers and large data storage solutions for their own use. This is no longer the case. Individuals and large companies alike can now pay for their data to be stored by separate entity. This separate entity offers them access to their data that they store in the “Cloud”, or large data centers. These companies that offer Cloud storage and computing allow for data to be accessed from anywhere, with no need for local disk space to be used by the client. The most important difference is that the owner of the data has granted responsibility of their data to a separate company.

Storing data “in the Cloud” is not only commonplace, but indeed necessary for many people and businesses to succeed in the current day. This stems from an increasingly mobile-minded society. Powerful devices capable of accessing huge stores of data are increasing expected to be smaller, thinner, and more portable. This creates a need for offloading data storage to Cloud servers. However, the convenience gained creates a different problem: security. The data that is now stored by the Cloud provider (Amazon S3, Google Cloud, Google Drive, Microsoft DropBox, Microsoft Azure, etc.) is not controlled by the original owner, or client. The client expects the Cloud provider to safeguard its data; however, the Cloud provider is motivated to store as little data as is possible, since storage capacity is ultimately a finite resource. A dishonest server may choose not to store all of the client’s data if it believes that the client will not notice. The server could delete all or part of the data, modify it, swap it with another’s data, and the list goes on. The server should be able to prove to the client at any time that its data is still fully intact.

A secure system is one defined by Availability, Confidentiality, and Integrity. Availability means that one can access one’s data at the times agreed upon (in this case, at all times). Cloud providers are most motivated to achieve this property as it is the service that they explicitly state to the consumer, and they succeed in this area because servers can run nonstop and redundancy allows for maintenance and failures to occur seamlessly. Confidentiality is the property that one’s data cannot be readily understood by another, and Cloud providers work to achieve this as a means to sell security to their clients. Integrity, however, is the property that one’s data is ensured to be fully intact. This is simply not provided by Cloud providers, as they have full control of the client’s data without ever truly proving it. Without this property guaranteed, Cloud providers cannot truly offer a secure system to their clients.

1.1 Motivation

Integrity, as stated, is the property that the data is not changed by anyone other than the client. The client should be confident that no change will be made to its data until the client explicitly makes the change. The server is responsible for upholding this property, but it does not truly prove it until the client downloads that data back off of the Cloud. Data that is accessed very infrequently is most vulnerable, as its integrity is just as infrequently verified. The Cloud providers are motivated to ensure its clients' data is intact, but they are also motivated to store data in the most efficient manner. Perhaps a server would delete data that it knows is infrequently accessed in order to make more room for other clients' data. This causes a security concern when dealing with the Cloud because there is no mechanism implemented by the Cloud provider to verify integrity to the client on demand.

The Department of Defense has even more reason to be interested in this topic. The DoD's data systems must be secure due to the sensitive nature of the data that is transmitted. One prime example of this would be the storing of satellite imagery. A great deal of high resolution photos are taken everyday, and they must be stored in order to be utilized as intelligence. There is also a need for these photos to be disseminated around the world. Some of these photos, by their very nature, are highly sensitive data, possibly depicting locations and action of both our forces and the adversaries'. A Cloud platform would certainly offer a solution, but it introduces the concern of integrity. It would be quite advantageous for an adversary, then, to gain access to the Cloud server housing these photos and delete or modify the photos to portray a different narrative. With the sheer quantity of photos, they are not all being utilized. Thus, it is possible to imagine that photos could be tampered with without the knowledge of the intelligence community.

In 2013, the United States Central Intelligence Agency (CIA) contracted Amazon Web Services (AWS) for a Cloud that would be able to handle classified data, which has very stringent protocols that must be met for transporting and sharing. More recently, the DoD awarded the Joint Enterprise Defense Infrastructure (JEDI) contract to Microsoft. This billion-dollar contract is predicted to revolutionize the abilities of the modern warfighter. With greater access to intelligence in real time, the warfighter will be able to make more educated decisions.

However, this inherently brings great risk into the battlefield if the data is not stored properly. If adversaries could also change it, delete it, or see who is accessing it, then our warfighters would be put in grave danger. Having a very efficient protocol that tests the integrity of the system at any time would only better the Cloud system that the DoD is already about to implement. Without an effective solution to the Pillar of Integrity, the system cannot be secure making it a liability. While confidentiality is important for ensuring that data cannot be easily read by the adversary, integrity is needed in order to ensure that any

data needed can be retrieved from the server on demand.

Whether it is large stores of medical data, legal records, or intelligence stores, massive databases are stored on the Cloud due to its convenience of access and ease of storage requirements for the client. However, in all of these situations, the entire database is rarely being accessed. In other words, there is data that must be stored, but there is no need to access it for long periods of time. Due to the lack of a mechanism from the Cloud provider to efficiently check the integrity of one's database, there is a risk of data being removed, modified, or injected without the knowledge of the client by either a malicious actor, or the Cloud server itself.

1.2 Report Organization

We begin this report by detailing the progression of previous work in this field. We continue by defining the security model under which we working. Our contributions, namely our matrix-based approach and accompanying protocol, are then presented. We overview the implementation that we deployed in the cloud to test our theory. A discussion of the setup, execution, and analysis of the experimentation follows. We cover use-based optimizations that are possible to make to our protocol presented, which trade efficiency in one area for decreased performance in another. Then, a comparison of our work to that of the state of the art in the field is given. We close by presenting the most likely applications where our work will be useful and where work on this project could continue in the future.

2 Previous Work

Protocols to check the Integrity of one’s data on a Cloud server have not been widely deployed in practice. However, this topic has been researched in academia. Two main philosophies have emerged: Proofs of Retrievability and Provable Data Possession. Both approaches have their strengths and weaknesses. There are two overarching desires of an integrity check: reliably determining if the data is correct and doing so efficiently. Proofs of Retrievability generally perform better in the former, while Provable Data Possession perform better on the latter. When analyzing the performance and efficiency, the computer science community often uses a notation known as Big-O notation. It represents an asymptotic that bounds the amount of computation or storage required. As an example, if a program simply prints “GO NAVY!” one time, its run time is considered to be $O(1)$ or constant time. If instead, the program prints every item in a list of N items, its run time would be considered $O(N)$ or linear time. This notation can aid in classifying the complexity or practicality of potential solutions to problems. Figure 1 shows how the most common bounds in this report relate. Specifically is shows how $O(\sqrt{N})$ is a huge improvement over linear time. Another important point to consider is what all is considered in a bound. Taking linear time, any coefficient with a linear complexity is still considered linear. This means that a run time of N , $7N$, $\frac{1}{2}N - 1000$, and $\frac{N}{2020}$ are all considered the same. Of course, these run times in practicality are quite different. In many cases, the simplest or most naive solution is considered first as a starting point for an optimal solution.

2.1 Naive Solution

For proving the retrievability of data on a Cloud server, a naive solution would be for the client to simply download all of their data. The problems with this solution are two fold. First, the client would not have gained anything from using the Cloud service because it would have to download everything back into its own memory, which it wanted to free up. Second, the client would have to store a duplicate of everything in order to check that what was sent back from the server was unaltered. This naive solution does not work in the defined purpose of the system. It requires $O(N)$ storage for both the client and Cloud server. Certainly the Cloud server will have $O(N)$ storage, but using Cloud storage altogether becomes redundant if the client must store $O(N)$ as well. It also requires N communication since the Cloud server must send everything back to the client.

2.2 Provable Data Possession

One issue with the naive solution is its N communication, or having to send all of the data. Provable Data Possession (PDP) is a model for checking the integrity of data while greatly decreasing the amount of

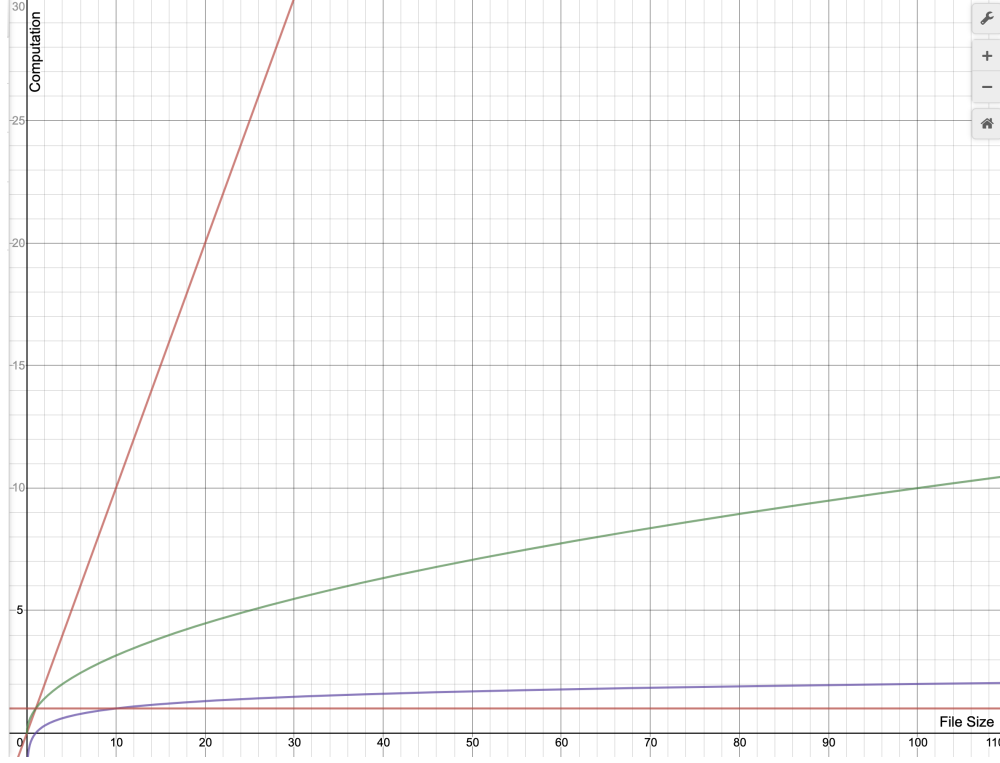


Figure 1: Relevant Run Time Complexities Compared. From Most Efficient to Least: $O(N)$ - Linear, $O(\sqrt{N})$ - Square Root, $O(\log N)$ - Log, $O(1)$ - Constant

communication needed between the client and Cloud server [2]. The client separates the data into blocks, such as individual files. The client then uses a Message Authentication Code (MAC) to give each file a unique identifier. The client must store the MAC key. It then sends all of the data with the unique identifiers to the Cloud server. A MAC is similar to a hash but with two key differences. One, a MAC is considered to be more secure because it has more stringent requirements on its input and functionality than a hash function does by definition. Two, only one MAC key needs to be stored for all of the blocks. The creation of a MAC for one chunk is shown in Figure 2. The “Client Key” in the figure is all that the client must store. Hashing each of the files would require the client to store each hash. Thus, using a MAC allows client storage to be $O(1)$. PDP keeps the server storage at $O(N)$, though storing the MAC information for each file does mean that it requires slightly more server storage than the naive solution.

This increase in storage on both sides allow for the communication to decrease from $O(N)$ to $O(1)$. To check the integrity, or audit the server, the client asks the server to send back a set number of blocks. The server then combines those blocks and identifiers into a single block and identifier and sends those two to the client. The client then determines what the identifiers of the blocks it request should be from the MAC key, combines them in the same fashion the server did, and then checks it against the server’s response. If

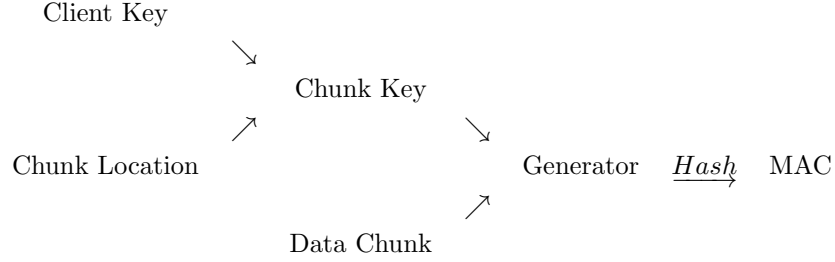


Figure 2: Creation of one MAC through Hashing the Concatenation of a Key and a Data Chunk

they match, then it is probable that all of the data is intact, as the server never knows which blocks will be audited.

PDP sought to decrease the amount of client storage needed to check the integrity as well as decrease the communication necessary. PDP accomplishes this by having the client and server store a small amount of extra information at all times.

2.3 Proof of Retrievability

A Proof of Retrievability (PoR) takes the idea of the PDP model, but introduces a higher level of integrity. In the PDP model, the server could have deleted a block without the client knowing so long as the client does not ask for that block. In addition, there would be no way of retrieving the deleted data. PoR uses two methods in conjunction in order to know with high probability that the server has retained all of the data and provide a way to recover some amount of deleted data [3].

The PoR model has the client first take the data and separate into *chunks*. An Error Correcting Code is then applied to each chunk independently. The Error Correcting Code is what allows for a certain amount of data recovery to happen. The Error Correcting Code encodes each chunk with extra data. The amount of extra data determines how much of the chunk could be erased while still being able to piece together the chunk afterwards. For example, if the chunk had size 10, and extra data of size 5 was added by the Error Correcting Code, then the chunk could be recovered with up to any erasure of size 5. Next, the error corrected chunks are each encrypted. This is done so that the server will not be able to distinguish any chunk from another.

Extra error corrected chunks are now added into the data. These are called *sentinels*, and they are indistinguishable from the other error corrected chunks. The client stores the value of the sentinels and their locations in the data, instead of the MAC key from the PDP model. The final step is to permute the error corrected chunks and sentinels, in order to randomly disperse the sentinels across the data. The data is now sent to the server to store.

During the audit process, the client requests a constant number of sentinels to be returned. The server combine the sentinels in a way similar to how it is done in the PDP model and send that back to the client. The client verifies the response with the values it stored locally. If it matches, then there is a high probability that the server has all of the data. This may seem just like PDP in that the server could delete data that is not in a sentinel and the client would be none the wiser. If this is true, then the deletion is most likely small compared to the size of a chunk, meaning that the Error Correcting Codes should be able to recover the data when retrieved. The PoR model sought to provide a better degree of integrity in its audit process by adding the Error Correcting Codes that can recover small erasures of the stored data.

2.4 Dynamic PoR

The PoR model yields a higher degree of integrity checking; however, it still lacks a way to efficiently update a part of the data. In order to edit any of the data, the process would have to start over to keep the high degree of probability. A Dynamic Proof of Retrievability (DPoR) was developed to address this issue.

The static approach will not work here, because a dynamic model should allow any part of the data to be changed or erased without affecting a large part of the data as a whole. One solution that was developed uses an Oblivious RAM (ORAM) scheme. The idea of ORAM is essentially that any read or write that the client wants applied to a source of memory will be converted into a set of reads and writes, such that there is no recognizable pattern to what type of data was retrieved or changed. This ultimately obscures any patterns in which the data is accessed or changed over time. For the DPoR model, the ORAM's source of memory is the Cloud server. The server must store more than just the client's data in order to allow for the redundant reads and writes to occur. The amount of server storage is bounded by $O(N)$, but it is substantially larger than the unique identifiers or the "sentinels" as with the previous approaches.

An audit in DPoR consists of the client requesting the value at a constant number of locations from the server. Each of these reads are converted through the ORAM scheme, and thus a number a redundant reads and writes occur. The number of redundant reads and writes per client action has been found to be bounded by $O(\log^2(N))$ [4]. This means that both client and server work for audits is also bounded by the same. Since each of the reads and writes must be relayed from client to server, the communication required for an audit is also $O(\log^2(N))$.

Updating in DPoR is writing a new value to the location of the data to be changed. This write action will go through the ORAM scheme, and thus a number a redundant reads and writes occur. From the same reasoning about the efficiency of audits, client work, server work, and communication for an update in DPoR is $O(\log^2(N))$. DPoR through ORAM checks the integrity with a high probability and allows for efficient

updates to the data being stored.

It should be stated that a dynamic model of PDP has been developed [5]. 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. It uses a tree like structure of hashes in order to ensure that the newest version is still present. This proposal deals primarily with the DPoR model, and as such the specifics of DPDP will not be expanded upon.

Later improvements by [1, 6] allow for dynamic updates in addition to [4] 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.

3 Security Model

When constructing these audit protocols, it is imperative to clearly define the security provided. In order to have a standard platform from which to begin, work in this field is based off a game between client and server. At its core, this game is a conversation between the client and the server. In a scenario where both parties are trusted and honest, the client would query the server whether all of the database in question is present. The server would respond in the affirmative or the negatory, and the client would have the answer. However, without direct control of the server, it is not assumed that the server is honest. The client's query must be one that can only be truthfully answered if all of the database is reachable. In this scenario, the client asks the server to do certain computations and respond with the answers. The client is then able to check those computations in some fashion determining the status of the database.

In order to win this game, the server must be able to break a given security model. With respect to PoRs, the server will generally win if the server can pass an audit without being able to retrieve every bit of the database in question. To be sure, any audit protocol in which the server can win this game is broken, meaning that it does not provide the security it claims. Any PoR using the security model we define below must provide correctness, authenticity, and retrievability. Correctness implies that a failure of an audit should not happen, as all queries are fulfilled honestly. Authenticity refers to the ability of the client to detect any communication that is not from an honest server due. Retriveability, intuitively, is the requirement that the server is indeed able to reach the entirety of the database the client stores in the cloud.

3.1 Our Security Model

We define a dynamic PoR scheme as consisting of the following five protocols between a client $\mathcal{C}$ with state $st_{\mathcal{C}}$ and a server $\mathcal{S}$ with state $st_{\mathcal{S}}$. The security parameter λ is the number of bits of security that the protocol depends upon. This is based in the number of bits used for randomness. As λ increases, computational efficiency decreases, but security increases. In practice, $\lambda = 128$ is a reasonable balance. Our definition is the same as given by [1], except that we follow [3] and include the **Extract** algorithm in the protocol explicitly.

The client may use random coins for any protocol; at a minimum, the **Audit** protocol *must* be randomized in order to satisfy retrievability non-trivially.

- $(st_{\mathcal{C}}, st_{\mathcal{S}}) \leftarrow \text{Init}(1^\lambda, b, D)$: On input of the security parameter λ and the database D , consisting of N bits arranged in blocks of b bits, outputs the client state $st_{\mathcal{C}}$ and the server state $st_{\mathcal{S}}$.
- $\{m_i, \text{reject}\} \leftarrow \text{Read}(i, st_{\mathcal{C}}, st_{\mathcal{S}})$: On input of an index $i \in 1..[N/b]$, the client state $st_{\mathcal{C}}$ and the server

state st_S , outputs $m_i = D[i]$ or **reject**.

- $\{(st'_C, st'_S), \text{reject}\} \leftarrow \text{Write}(i, a, st_C, st_S)$: On input of an index $i \in 1..\lceil N/b \rceil$, data a , the client state st_C and the server state st_S , outputs a new client state st'_C and a new server state st'_S or **reject**.
- $\{\pi, \text{reject}\} \leftarrow \text{Audit}(st_C, st_S)$: On input of the client state st_C and the server state st_S , outputs a succesful transcript π or **reject**.
- $D \leftarrow \text{Extract}(st_C, \pi_1, \pi_2, \dots, \pi_e)$: On input of independent **Audit** transcripts $\pi_1, \dots, \pi_e$, outputs the database D . The number of required transcripts e must be a polynomially-bounded function of N , b , and λ .

3.2 Correctness

A correct execution of the protocols by honest client and server results in audits being accepted and reads to recover the last updated value of the database. More formally, Correctness is:

Definition 1 (Correctness). *For any parameters λ, N, b , there exists a predicate **IsValid** such that, for any database D of N bits, $\text{IsValid}(D, \text{Init}(1^\lambda, b, D))$. Furthermore, for any state such that $\text{IsValid}(D, st_C, st_S)$ and any index i with $0 \leq i < \lceil N/b \rceil$, we have*

- $\text{Read}(i, st_C, st_S) = D[i]$;
- $\text{IsValid}(D', \text{Write}(i, a, st_C, st_S))$, where $D'[i] = a$ and the remaining $D'[j] = D[j]$ for every $j \neq i$;
- $\text{Audit}(st_C, st_S) \neq \text{reject}$;
- For e audits $\text{Audit}_1, \dots, \text{Audit}_e$ with independent randomness, with probability $1 - \text{negl}(\lambda)$, $\text{Extract}(st_C, \text{Audit}_1(st_C, st_S), \dots, \text{Audit}_e(st_C, st_S)) = M$.

Note that, even though $\mathcal{C}$ may use random coins in the protocols, a correct PoR by this definition should have no chance of returning **reject** in any **Read**, **Write** or **Audit** with an honest client and server.

3.3 Authenticity

The authenticity requirement stipulates that the client can always detect (except with negligible probability) if any message sent by the server deviates from honest behavior. We use the following game between an observer $\mathcal{O}$, a malicious server $\bar{\mathcal{S}}$ and an honest server $\mathcal{S}$ for the adaptive version of authenticity, with the same game as [1]:

1. $\bar{\mathcal{S}}$ chooses an initial memory D . $\mathcal{O}$ runs **Init** and sends the initial memory layout st_S to both $\bar{\mathcal{S}}$ and $\mathcal{S}$.

2. For a polynomial number of steps $t = 1, 2, \dots, \text{poly}(\lambda)$, $\bar{\mathcal{S}}$ picks an operation op_t where operation op_t is either **Read**, **Write** or **Audit**. $\mathcal{O}$ executes the operations with both $\bar{\mathcal{S}}$ and $\mathcal{S}$.
3. $\bar{\mathcal{S}}$ is said to win the game, if any message sent by $\bar{\mathcal{S}}$ differs from that of $\mathcal{S}$ and $\mathcal{O}$ did not output **reject**.

Definition 2 (Authenticity). *A PoR scheme satisfies adaptive authenticity, if no polynomial-time adversary $\bar{\mathcal{S}}$ has more than negligible probability in winning the above security game.*

3.4 Retrievalability

Intuitively, the retrievalability requirement stipulates that whenever a malicious server can pass the audit test with high probability, the server must know the entire memory contents D . To model this, [4] use a *black-box rewinding access*: from the state of the server before any passed audit, there must exist an extractor algorithm that can reconstruct the complete correct database. As in [1], we insist furthermore that the extractor does not use the complete server state, but only the transcripts from successful audits. In the following game, note that the observer $\mathcal{O}$ running the honest client protocols may only update its state $st_{\mathcal{C}}$ during **Write** operations, and hence the **Audit** protocols are independently randomized from the client side, but we make no assumptions about the state of the adversary $\bar{\mathcal{S}}$.

1. $\bar{\mathcal{S}}$ chooses an initial database D . $\mathcal{O}$ runs **Init** and sends the initial memory layout $st_{\mathcal{S}}$ to $\bar{\mathcal{S}}$;
2. For $t = 1, 2, \dots, \text{poly}(\lambda)$, the adversary $\bar{\mathcal{S}}$ adaptively chooses an operation op_t where op_t is either **Read**, **Write** or **Audit**. The observer executes the respective protocols with $\bar{\mathcal{S}}$, updating $st_{\mathcal{C}}$ and D according to the **Write** operations specified;
3. The observer runs $2e$ **Audit** protocols with $\bar{\mathcal{S}}$ and records the outputs $\pi_1, \dots, \pi_{e'}$ of those which did not return **reject**, where $0 \leq e' \leq 2e$.
4. The adversary $\bar{\mathcal{S}}$ is said to win the game if $e' \geq e$ and $\text{Extract}(st_{\mathcal{C}}, \pi_1, \dots, \pi_{e'}) \neq D$.

Definition 3 (Retrievalability). *A PoR scheme satisfies retrievalability if no polynomial-time adversary $\bar{\mathcal{S}}$ has more than negligible probability in winning the above security game.*

4 Our Contribution

We first present a simple version of our PoR protocol. This version contains the main ideas of our approach, namely, using matrix-vector products during audits to prove retrievability. It also makes use of Merkle hash trees during reads and updates to ensure authenticity.

This protocol uses only $N + o(N)$ persistent server storage, which is an improvement to the $O(N)$ persistent storage of existing PoR schemes, and is the main contribution of this work. The costs of our **Read** and **Write** operations are similar to existing work, but we incur an asymptotically higher cost for **Audit** operations, namely $O(\sqrt{N})$ communication bandwidth and $O(N)$ server computation time. We demonstrate in the Section 6 that this tradeoff between persistent storage and **Audit** cost is favorable in cloud computing settings for realistic-size databases.

A natural question is if this tradeoff is needed in a perfect solution; in other words, is there a PoR solution that can have minimized overhead storage *and* minimized audit computation. However, this solution cannot be. There exists an inverse relationship between extra storage and audit computation required for a protocol to be provably secure. By *extra storage* here we mean exactly the number of extra bits of persistent memory, on the client or server, beyond the bit-length of the original database being represented. For any PoR scheme with sub-linear audit cost, we have

$$(\text{extra storage size}) \cdot \frac{\text{audit cost}}{\log(\text{audit cost})} \in \Omega(\text{data size}).$$

This relationship defines the lower bound of what is possible, clearly showing the inherent tradeoff that is central to our work. None of the previous schemes, nor those presented in this report, make this lower bound tight. Nonetheless, it demonstrates that a “best of all possible worlds” scheme with, say, $O(\sqrt{N})$ extra storage and $O(\log N)$ audit cost to store an arbitrary N -bit database, is impossible.

4.1 Overview of Matrix-Based Approach

A summary of our four protocols is shown in Table 1, where dashed boxes are the classical, Merkle hash tree authenticated, remote read/write operations. The **Init** protocol is needed for initial set up, but it only need be run the once. Arbitrary calls to **Read**, **Write**, and *importantly* **Audit** can then be made in any order requested by the client.

Our idea is to use verifiable computing schemes as, e.g., proposed in [7]. Our choice for this is to treat the data as a square matrix of dimension roughly $\sqrt{N} \times \sqrt{N}$. This allows for the matrix multiplication verification described in [8] to be used as a computational method for the audit protocol.

Crucially, this does not require any additional metadata; the database M is stored as-is on disk, our algorithm merely treats the machine words of this unmodified data as a matrix stored in row-major order. Although the computational complexity for an **Audit** operation is asymptotically $O(N)$ for the server, this entails only a single matrix-vector multiplication, in contrast to some prior work which requires expensive RSA computations [2].

Even with the audit protocol testing if the server can retrieve all of the data, we need to ensure the server's response to each **Read** and **Write** is honest. To ensure authenticity also during **Read** and **Write** operations, we combine this linear algebra idea above with a standard Merkle hash tree. Without this, the server, while able to access all of the database correctly, could return client queries with incorrect or corrupted data.

Table 1: Client/server PoR protocol with low storage server

	Server	Communications	Client
		$N = mn \log_2 q$	$\mathbf{r} \xleftarrow{\$} \mathcal{R}_q^m$
			$\mathbf{s}^\top \leftarrow \mathbf{r}^\top \mathbf{D}.$
Init		$\mathbf{D}, T_{\mathbf{D}} \leftarrow \text{MTInit} \xleftarrow{\quad} \lambda, b, \mathbf{D}$ $\xrightarrow{\quad} r_{\mathbf{D}}$	
	Stores $\mathbf{D}$ and $T_{\mathbf{D}}$		Stores $\mathbf{r}, \mathbf{s}$, and $r_{\mathbf{D}}$
Read		$\mathbf{D}, T_{\mathbf{D}} \xrightarrow{\quad} \text{MTVerifiedRead} \xleftarrow{\quad} i, j, r_{\mathbf{D}}$ $\xrightarrow{\quad} \mathbf{D}_{ij}$	Returns $\mathbf{D}_{ij}$
Write		$\mathbf{D}, T_{\mathbf{D}} \xrightarrow{\quad} \text{MTVerifiedWrite} \xleftarrow{\quad} i, j, \mathbf{D}'_{ij}, r_{\mathbf{D}}$ $\mathbf{D}', T'_{\mathbf{D}} \xleftarrow{\quad} \xrightarrow{\quad} \mathbf{D}_{ij}, r'_{\mathbf{D}}$	$\mathbf{s}'_i \leftarrow \mathbf{s}_i + (\mathbf{D}'_{ij} - \mathbf{D}_{ij})\mathbf{r}_j$ Stores updated $r'_{\mathbf{D}}, \mathbf{s}'$
	Stores updated $\mathbf{D}', T'_{\mathbf{D}}$		
Audit		$\xleftarrow{\quad} \mathbf{c}$ $\xrightarrow{\quad} \mathbf{v}$	$\mathbf{c} \xleftarrow{\$} \mathcal{R}_q^n$ $\mathbf{r}^\top \mathbf{v} \stackrel{?}{=} \mathbf{s}^\top \mathbf{c}$
	$\mathbf{v} \leftarrow \mathbf{D}\mathbf{c}$		

4.2 Initialization

The basic premise of our particular PoR is to treat the data, consisting of N bits organized in machine words, as a matrix $\mathbf{D} \in \mathcal{R}_q^{m \times n}$, where $\mathcal{R}_q$ is a suitable finite ring of size q . Crucially, the choice of ring $\mathcal{R}_q$ detailed below does not require any modification to the raw data itself; that is, any element of the matrix $\mathbf{D}$ can be retrieved in $O(1)$ time. At a high level, our **Init** and **Audit** protocols follows the matrix multiplication verification technique of [8].

The data that the client wishes to store on the server is first converted into a matrix. If the size of

the data is N , then the square root of the size of the data serves as the dimensions for its square matrix representation, $\mathbf{D}$ has dimensions $\sqrt{N} \times \sqrt{N}$. The data is read in order and inserted into the matrix. The construction of the matrix is shown in Figure 3. A random vector $\mathbf{r}$ is then created. This vector has the same number of columns as the data matrix, thus giving it dimensions $1 \times \sqrt{N}$. The matrix product $\mathbf{r}^\top \mathbf{D}$ is taken, resulting in a new vector $\mathbf{s}$, still of dimensions $1 \times \sqrt{N}$. The construction of these two row vectors is shown in Figure 4. More formally, the Client chooses a secret random control vector $\mathbf{r} \in \mathcal{R}_q^m$ and computes a second secret control vector $\mathbf{s} \in \mathcal{R}_q^n$ according to

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

Note that $\mathbf{r}$ and $\mathbf{s}$ are held constant for the duration of the storage, which does not compromise security because no message which depends on $\mathbf{r}$ or $\mathbf{s}$ is ever sent to the server. In particular, this means that multiple clients could use different, independent, control vectors $\mathbf{r}$ as long as they have a way to synchronize **Write** operations (modifications of their shared database) offline.

This implementation can be adapted to use any type of data as a element, such as files, numbers, or collections of bits. For simplicity, this example uses integers.

Data: {3, 11, 18, 2, 24, 55, 63, 42, 38, 9, 27, 14, 46, 59, 13, 7}

Size of Data: $N = 16$

Matrix Dimension: $\sqrt{N} = 4$

$\mathbf{D} =$

3	11	18	2
24	55	63	42
38	9	27	14
46	59	13	7

Figure 3: Data Converted to 4×4 Matrix

The matrix data $\mathbf{D}$ is sent to the Cloud server to be stored. Since the Cloud server does not need to store anything further, Cloud storage is strictly N , the data itself. While the DPOR technically only requires the Cloud server to store $O(N)$, the true amount has a coefficient of nearly 10 due to the extra space needed to hide the operations, which in reality would be costly for the client. For this proposed solution, the client is expected to store an amount in the order of $O(\sqrt{N})$, comprised of the random vector $\mathbf{r}$ and the secret vector $\mathbf{s}$. This initial communication does require the client to perform $O(N)$ work, as performing the matrix multiplication requires touching every element of the data, as well as a communication cost of $O(N)$ due

$$\begin{aligned}
\mathbf{r} &= \begin{bmatrix} 1 & 10 & 11 & 21 \end{bmatrix} \\
\mathbf{s} = \mathbf{r}^T \mathbf{D} &= \begin{bmatrix} 1 & 10 & 11 & 21 \end{bmatrix} \times \begin{bmatrix} 3 & 11 & 18 & 2 \\ 24 & 55 & 63 & 42 \\ 38 & 9 & 27 & 14 \\ 46 & 59 & 13 & 7 \end{bmatrix} \\
&= \begin{bmatrix} 1(3) + 10(24) + 11(38) + 21(46) & 1(11) + 10(55) + 11(9) + 21(59) & 1(18) + 10(63) + 11(27) + 21(13) & 1(2) + 10(42) + 11(14) + 21(7) \end{bmatrix} \\
\mathbf{s} &= \begin{bmatrix} 1627 & 1899 & 1218 & 723 \end{bmatrix}
\end{aligned}$$

Figure 4: Random and Secret 1×4 Row Vectors Constructed by Client

to sending all of the data across the wire. Both of these costs can be justified because it is logical for the client to process each piece of its data and send all of it to be stored in the Cloud. The state of server/client storage after initialization is shown in Figure 5.

It is important to point out that data does not need to consist of a perfect square number of elements. The ceiling of the square root of the data size, $\lceil \sqrt{N} \rceil$, would be used for all matrix dimensions, and any indices not filled would be set to zero for all future math.

Client $\mathbf{r}^T = \begin{bmatrix} 1 & 10 & 11 & 21 \end{bmatrix}$ $\mathbf{s}^T = \begin{bmatrix} 1627 & 1899 & 1218 & 723 \end{bmatrix}$	Server $\mathbf{D} = \begin{bmatrix} 3 & 11 & 18 & 2 \\ 24 & 55 & 63 & 42 \\ 38 & 9 & 27 & 14 \\ 46 & 59 & 13 & 7 \end{bmatrix}$
--	--

Figure 5: Summary of Storage after Initialization

4.3 Audits

The audit process must be able to check that all of the client's data is still present on the Cloud server. The first step for this solution is for the client to create a challenge vector $\mathbf{c} \in \mathcal{R}_q^n$. As defined, this vector will consist of randomly chosen values; however, this time it will have as many rows as $\mathbf{D}$ yielding dimensions $\sqrt{N} \times 1$ as shown in Figure 6. The challenge vector $\mathbf{c}$ will be sent to the Cloud server. The Cloud server will then take the matrix product $\mathbf{Dc}$. The resulting $\sqrt{N} \times 1$ vector $\mathbf{v} \in \mathcal{R}_q^m$ is sent back to the client.

$$\mathbf{v} = \mathbf{Dc} \tag{2}$$

The example calculation for $\mathbf{v}$ is shown in Figure 7.

$$\mathbf{c} = \begin{array}{|c|} \hline 2 \\ \hline 21 \\ \hline 12 \\ \hline 20 \\ \hline \end{array}$$

Figure 6: Challenge 4×1 Column Vector Constructed Randomly by Client

$$\mathbf{v} = \mathbf{D}\mathbf{c} = \begin{array}{|c|c|c|c|} \hline 3 & 11 & 18 & 2 \\ \hline 24 & 55 & 63 & 42 \\ \hline 38 & 9 & 27 & 14 \\ \hline 46 & 59 & 13 & 7 \\ \hline \end{array} \times \begin{array}{|c|} \hline 2 \\ \hline 21 \\ \hline 12 \\ \hline 20 \\ \hline \end{array} = \begin{array}{|c|} \hline 3(2) + 11(21) + 18(12) + 2(20) \\ \hline 24(2) + 55(21) + 63(12) + 42(20) \\ \hline 38(2) + 9(21) + 27(12) + 14(20) \\ \hline 46(2) + 59(21) + 13(12) + 7(20) \\ \hline \end{array}$$

$$\mathbf{v} = \begin{array}{|c|} \hline 493 \\ \hline 2799 \\ \hline 869 \\ \hline 1627 \\ \hline \end{array}$$

Figure 7: Result 4×1 Column Vector Calculated by Server

At this time, the client has the initial random vector $\mathbf{r}$, the secret vector $\mathbf{s}$, the current challenge vector $\mathbf{c}$, and the server's resulting vector $\mathbf{v}$. This is enough information to know if the server has upheld its integrity with high probability. The client takes the dot product $\mathbf{s}^\top \mathbf{c}$. The product will be a single value because the operation takes a $1 \times \sqrt{N}$ matrix and a $\sqrt{N} \times 1$, which yields a 1×1 matrix or a single value. The client will also compute the dot product $\mathbf{r}^\top \mathbf{v}$, which will yield a single value following the same reasoning. Figure 8 shows the both dot products that the client performs for the example. If the server has indeed upheld integrity of the client's data, then two products will be the same because each of the operations is effectively taking the matrix product $\mathbf{r}^\top \mathbf{D}\mathbf{c}$, as shown in Figure 9.

In this fashion, the client can check every element of the data stored on the Cloud server because the server must have each value correct in data matrix $\mathbf{D}$ to send back the correct vector to be used in testing. The proof of retrievability will rely on the fact that observing several successful audits allows, with high probability, recovery of the matrix $\mathbf{D}$, and therefore of the entire database. It is important to point out that server never knows the value of $\mathbf{r}$ or $\mathbf{s}$. The server must know all of $\mathbf{D}$ in order to response to the client's challenge correctly. It is highly improbable that the server would be able to guess all of the values correctly, and thus this solution claims to check the Integrity of the system. This audit protocol is an improvement over the original PoR scheme because there is no interval of audits before there must be a complete refresh of the system. The challenge vector $\mathbf{c}$ is randomly created for each audit, and the use of any value does not

$$\begin{aligned}
\mathbf{s}^\top \mathbf{c} &= \begin{array}{|c|c|c|c|} \hline 1627 & 1899 & 1218 & 723 \\ \hline \end{array} \times \begin{array}{|c|} \hline 2 \\ \hline 21 \\ \hline 12 \\ \hline 20 \\ \hline \end{array} = 1627(2) + 1899(21) + 1218(12) + 723(20) \\
&\mathbf{s}^\top \mathbf{c} = \mathbf{72209} \\
\mathbf{r}^\top \mathbf{v} &= \begin{array}{|c|c|c|c|} \hline 1 & 10 & 11 & 21 \\ \hline \end{array} \times \begin{array}{|c|} \hline 493 \\ \hline 2799 \\ \hline 869 \\ \hline 1627 \\ \hline \end{array} = 1(493) + 10(2799) + 11(869) + 21(1627) \\
&\mathbf{r}^\top \mathbf{v} = \mathbf{72209}
\end{aligned}$$

Figure 8: Final Client Side Computation for Audit Process. In the case of this example, $\mathbf{s}^\top \mathbf{v}$ does equal $\mathbf{r}^\top \mathbf{v}$, which means that the server passes the challenge.

give information to the server. The original solution gave away the position of a sentinel. Once all of the sentinels were used, the client would have to restart the process to ensure that the integrity could be checked accurately.

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

$$\begin{aligned}
\mathbf{s}^\top \mathbf{c} &\stackrel{?}{=} \mathbf{r}^\top \mathbf{v} \\
\mathbf{s} &= \mathbf{r}^\top \mathbf{D} \implies \mathbf{s}^\top \mathbf{c} = \mathbf{r}^\top \mathbf{D} \mathbf{c} \\
\mathbf{v} &= \mathbf{D} \mathbf{c} \implies \mathbf{r}^\top \mathbf{v} = \mathbf{r}^\top \mathbf{D} \mathbf{c} \\
\therefore \mathbf{r}^\top \mathbf{D} \mathbf{c} &= \mathbf{r}^\top \mathbf{D} \mathbf{c}
\end{aligned}$$

Figure 9: Proof that Proposed Audit Check Is Valid

Some similar ideas were used by [9] 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 for audits is much simpler and independent of computational hardness assumptions.

4.4 Merkle Tree Retrieval and Updates

Thus far, this solution works well for static data. Retrieval of data is a simple access, as the matrix structure keeps the data in order. However, in today's age data is normally being updated and altered constantly. This solution proves to work efficiently in a dynamic environment as well. The mechanics of matrix multiplication can be used in order to update single elements of data at a time without losing any ability to check the integrity through an audit. First, we present the details of the Merkle Tree interface. The mathematical theory of updating values in $\mathbf{D}$ are given afterwards.

4.4.1 Merkle Tree Protocols

While the audit operates on the data in word-size chunks as members of a finite ring $\mathcal{R}_q$, retrieving and updating data is done at the byte level with support for retrieving any range of bytes (that is legal with the size of the data). A Merkle hash tree with block size b is used here to ensure authenticity of individual **Read** operations. This is a binary tree, stored on the server, consisting of $O(N/b)$ hashes, each of size 2λ for collision resistance.

The Client stores only the root hash, and can perform, with high integrity assurance, any read or write operation on a range of k bytes in $O(k + b + \log(N/b))$ communication and computation time. When the block size is large enough, the extra server storage is $o(N)$; for example, $b \geq \log N$ means the hash tree can be stored using $O(N\lambda/\log N)$ bits.

Merkle hash trees are a classical result, commonly used in practice, and we do not claim any novelty in our use here [10, 11]. To that end, we provide three protocols to abstract the details of the Merkle hash tree.

These are all two-party protocols between a Server and a Client, but without any requirement for secrecy. A vertical bar $|$ in the inputs and/or outputs of a protocol indicates Server input/output on the left, and Client input/output on the right. When only the Client has input/output, the bar is omitted for brevity.

The **MTVerifiedRead** and **MTVerifiedWrite** protocols may both fail to verify a hash, and if so, the Client outputs **reject** and aborts immediately.

Our three Merkle tree protocols are as follows:

- **MTInit** $(1^\lambda, b, D) \mapsto (D, T_D \mid r_D)$.

The Client initializes database D for storage in size- b blocks. The entire database D is sent to the Server, who computes hashes and stores the resulting Merkle hash tree T_D . The Client also computes this tree, but discards all hashes other than the root hash r_D . This is done as a part of **Init**.

The cost in communication and computation for both parties is bounded by $O(|D|) = O(N)$.

- **MTVerifiedRead** $(D, T_D \mid range, r_D) \mapsto D_{range}$.

The Client sends a contiguous byte range to the server, i.e., a pair of indices within the size of D . This range determines which containing range of blocks are required, and sends back these block contents, along with left and right boundary paths in the hash tree T_D . Specifically, the boundary paths include all left sibling hashes along the path from the first block to the root node, and all right sibling hashes along the path from the last block to the root; these are called the “uncles” in the hash tree.

Using the returned blocks and hash tree values, the Client reconstructs the Merkle tree root, and compares with r_D . If these do not match, the Client outputs **reject** and aborts. Otherwise, the requested range of bytes is extracted from the (now-verified) blocks and returned. **Read** calls **MTVerifiedRead** every time to ensure honesty.

The cost in communication and computation time for both parties is at most $O(|range| + b + \log(N/b))$.

- **MTVerifiedWrite** $(D, T_D \mid range, D'_{range}, r_D)$
 $\mapsto (D', T'_D \mid D_{range}, r'_D)$.

The Client wishes to update the data D'_{range} in the specified range, and receive the *previous value* of that range, D_{range} , as well as an updated root hash r_D .

The protocol begins as **MTVerifiedRead** with the Server sending all blocks to cover the range and corresponding left and right boundary hashes from T_D . After the Client retrieves and verifies the old value D_{range} with the old root hash r_D , she updates the blocks with the new value D'_{range} and uses the same boundary hashes to compute the new root hash r'_D .

Separately, the Server updates the underlying database D' in the specified range, then recomputes all affected hashes in T'_D . **Write** call **MTVerifiedWrite** every time to ensure honesty.

The asymptotic cost is identical to that for the **MTVerifiedRead** operation.

4.4.2 Updates

For this update to be successful, the server must have the updated element, and the client needs to have an updated secret vector $\mathbf{s}$. The first can be achieved quite easily due to the natural addressing that comes with a matrix. The client knows which element it wishes to update. The client sends the server the new data value and the location (i, j) that the server should update. This only requires $O(1)$ communication and server work. The client must now update $\mathbf{s}$ to be able to check the integrity in the future. Since $\mathbf{s} = \mathbf{r}^\top \mathbf{D}$, one value within $\mathbf{s}$ will need to be recomputed. Given that the element to be updated in is column j , then the $\mathbf{r}$ will need to be multiplied by $\mathbf{D}_j$, the entire j th column of the data matrix to compute the corresponding

element $\mathbf{s}_j$, the j th element of the secret vector. Upon updating the new location, the server would return $\mathbf{D}_j$, so that the client can recompute. This would mean that both communication and client work become $O(\sqrt{N})$ with server work remaining at $O(1)$.

To continue the example, imagine that the client wants the value at $\mathbf{D}_{2,1}$ to be updated to 7, resulting in the updated data matrix $\mathbf{D}'$. Figure 10 details how the client's $\mathbf{s}$ vector can be updated.

$$\mathbf{s} = \begin{array}{|c|c|c|c|} \hline 1627 & \mathbf{1899} & 1218 & 723 \\ \hline \end{array}$$

Every element of $\mathbf{s}$ is computed as follows:

$$\mathbf{s}_j = \sum_{i=0}^{\sqrt{N}} \mathbf{r}_i(\mathbf{D}_{i,j})$$

Since all elements remain intact except $\mathbf{s}_1$, only one summation is necessary.

$$\begin{aligned} \mathbf{s}'_1 &= \sum_{i=0}^4 \mathbf{r}_i(\mathbf{D}'_{i,1}) \\ &= 1(11) + 10(55) + \mathbf{11(7)} + 21(59) \\ &= \mathbf{1877} \end{aligned}$$

$$\mathbf{s}' = \begin{array}{|c|c|c|c|} \hline 1627 & \mathbf{1877} & 1218 & 723 \\ \hline \end{array}$$

Figure 10: Updating Secret Vector

$O(\sqrt{N})$ for communication and client work is better than the naive solution, but it can be improved to even further to $O(1)$. The process by which the client updates the secret vector $\mathbf{s}$ can be refined. Analyzing the mechanics of matrix multiplication again, we find a way to update in a completely $O(1)$ fashion. The value $\mathbf{s}_j$ will only differ due to the change in the one specific, updated element $\mathbf{D}_{i,j}$. Since $\mathbf{s}$ is a sum of products, by calculating the difference of the value strictly at $\mathbf{D}_{i,j}$, the difference of $\mathbf{s}_j$ is calculated. Figure 11 details how $\mathbf{s}$ can be updated in this more efficient manner.

When updated in this fashion, the client's work becomes $O(1)$ because it need only perform the math above. The communication also becomes $O(1)$ because the server does not need to send anything back after the client sends the new value and its location. When executing the update in practice, **MTVerifiedRead** and **MTVerifiedWrite** are used as the interface to guarantee that values queried are honest.

$$\begin{aligned}
\mathbf{s}_j' &\stackrel{?}{=} \mathbf{s}_j + \mathbf{r}_i(\mathbf{D}_{i,j}' - \mathbf{D}_{i,j}) \\
\mathbf{s}_1 &= \sum_{i=0}^4 \mathbf{r}_i(\mathbf{D}_{i,1}) \\
&= 1(11) + 10(55) + 11(9) + 21(59) \\
&= 1899 \\
\mathbf{s}_1' &= \mathbf{s}_1 + \mathbf{r}_2(\mathbf{D}_{2,1}' - \mathbf{D}_{2,1}) \\
&= 1899 + 11(7 - 9) \\
&= 1877
\end{aligned}$$

Figure 11: Updating Secret Vector Efficiently

5 Implementation

One improvement that this work brings to the field over previous work is an implementation of the theory. We have paired this novel approach to PoRs with a functioning codebase, in order to better understand its complexities and tests its performance in the real world. As we detail in Section 6, we deploy the codebase developed to the cloud in order to test all aspects of performance. Due to the tradeoff of computation for minimal overhead storage that we claimed would be beneficial in a cloud environment, this implementation allows for a definitive method of supporting that claim.

This project was programmed in C from the ground up. The decision to use C was based in the language’s ability to run more efficiently than other languages and means to control more aspects of the control sequence. C is less intuitive than languages such as Python; however, it allows (and forces) the programmer to decide on the low level functionality of the program and control minute details of how to interact with data. This was necessary due to the need for efficiency in this project.

Programs were made that would be executed by either the client or the server. The client’s program would only be executed when the client decided it wanted to run an audit, retrieve data, or update data. All operations that the client could do are accessed through the single program, creating an simple interface for the client to do so. The server’s program was designed to always be running, though. This is so that it would be waiting for an incoming connection from the client at any time, which is expected by a client storing data remotely. The server program acts only as a point of connection for the client. Once the client connects, another process is started to handle the desired operation specified.

Passing information between the client and server was imperative (the challenge and response vectors, Merkle-tree hashes, specified data chunks, etc.). Thus, a custom communication scheme was necessary to design. This was another benefit of programming in C, as controlling exactly how much data flows, when

it should be sent and recieved, and how much to store on the other end are all able to be controlled in detail. The communication scheme was kept as simple as possible in order to minimize the time not spent on computation for an audit.

For reasons detailed in Section 6.5, we required the use of the Message Passing Interface (MPI). This was used to allow multiple computers (not multiple cores on one computer) to divide the computational work between them. Controlling the access of data needed between them to run efficiently was again the greatest concern.

CMake was used allow for an easy build process for others. Required codebases are checked, source code is linked as needed, and executable files are created from one command. This allows for any computer with any operating system to build the codebase to run on that computer. This was necessary for allowing the public to access and use the implementation which we have developed. We are happy to share the implementation of our work here: <https://github.com/dsroche/la-por>.

This codebase is about 1800 lines of code. While this metric does not directly show quality or effort, the order of magnitude shows that there has been significant work done in creating this project from the ground up. This project incorporates numerous outside libraries and code bases to increase efficiency, meaning there is substantial work in managing said dependencies. Our implementation was developed over more than a year, solely focused on accurately displaying the theory presented.

6 Experimentation

As we have seen, compared to other dynamic PoR schemes, our protocol aims at achieving the high security guarantees of PoR, while trading near-minimal persistent server storage for increased audit computation time.

In order to address the practicality of this tradeoff, we implemented and tested our PoR protocol using virtual machines and disks on the Google Cloud Platform service, a commercial competitor to Amazon Web Services, Microsoft Azure, and the like.

Specifically, we address two primary questions:

- What is the monetary cost and time required to perform our $O(N)$ time audit on a large database?
- How does the decreased cost of persistent storage trade-off with increase costs for computation during audits?

Our experimental results are summarized in Tables 3 to 5. For a 1TB data file, the $O(\sqrt{N})$ communication cost of our audit entails less than 12MB of data transfer, and our implementation executes the $O(N)$ audit for this 1TB data file in less than 16 minutes and for a monetary cost of less than \$0.25 USD.

By contrast, just the extra persistent storage required by other existing PoR schemes would cost at least \$40 USD or as much as \$200 USD per month, not including any computation costs for audits. These results lead to two tentative conclusions:

- The communication and computation costs of our audit protocol are not prohibitive in practice despite their unfavorable asymptotics; and
- Our solution is the most cost-efficient PoR scheme available when few audits are performed per day.

We also emphasize again that a key benefit to our PoR scheme is its *composability* with existing software, as the data file is left in-tact as a normal file on the Server’s filesystem.

The remainder of this section gives the full details of our implementation and experimental setup. The full code of our implementation will be released open-source, and can be made anonymously available to the reviewers upon request.

6.1 Parameter selection

To balance the bandwidth (protocol communications) and the client computation costs, we chose to represent $\mathbf{D}$ as a square matrix with dimensions $m = n = \sqrt{N/8}$, where the 8 comes from our choice of $\mathcal{R}_q$

corresponding to 64-bit words (see Section 6.2). The resulting asymptotic costs for these parameter choices are summarized in Table 2.

Table 2: Proof of retrievability via square matrix verifiable computing

		Server	Communication	Client
Storage		$N + o(N)$		$O(\sqrt{N})$
Comput.	Init	$O(N)$	N	$O(N)$
	Audit	$O(N)$	$O(\sqrt{N})$	$O(\sqrt{N})$
	Read/Write	$O(\log(N))$	$O(\log(N))$	$O(\log(N))$

6.2 Two Prime Calculations

In order to leave the data file unmodified in persistent storage, while allowing constant-time random access to individual matrix elements, we break the data into word-sized (8 byte) blocks, and choose a finite ring $\mathcal{R}_q$ with $q \geq 2^{64}$.

One possibility would be to set q as a prime larger than 2^{64} , but this would entail costly multiple-precision computations for the modular arithmetic. Instead, we chose the ring $\mathcal{R}_q = \mathbb{F}_{p_1} \times \mathbb{F}_{p_2}$ as the direct product of two finite fields, each of large prime order. When $q = p_1 p_2 \geq 2^{64}$, this ensures unique recovery of the database from images in $\mathcal{R}_q$ via Chinese remaindering, and also allows efficient computation without extended precision.

In our implementation, we chose $p_1 = 2^{31} - 1$ and $p_2 = 2^{36} - 5$. That p_1 is a Mersenne prime makes computations with it particularly efficient, but a second Mersenne prime of similar size does not exist. For the actual arithmetic we used the low-level routines provided by the open-source high performance number theory library Flint [12].

This two-prime setup is equivalent to storing two databases $\mathbf{D}_1$ and $\mathbf{D}_2$ in finite fields $\mathbb{F}_{p_1}$ and $\mathbb{F}_{p_2}$ respectively, and so the formal security proof of Theorem 2 applies as long as the smaller prime p_1 is larger than the column dimension n of the database matrix $\mathbf{D}$. This means our implementation parameters satisfy the security proof requirements for sizes up to $N = 144\text{PB}$.

6.3 Experimental Design

Our implementation provides the **Init**, **Read**, **Write**, and **Audit** procedures as described in the previous section, including the Merkle hash tree implementation for read/write integrity. As the cost of the first three of these are comparable to prior work, we focused our experiments on the **Audit** protocol.

We ran two sets of experiments, using virtual machines and disks 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 7.5GB memory. In the second set of experiments, the 4/16 parallel VMs running MPI were all n1-standard-1 instances with 1 vCPU and 3.75GB memory. Further information and specifications of the machines can be found at <https://cloud.google.com/compute/docs/machine-types>. 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 1GB, 10GB, 100GB, and 1TB were generated with random bytes from `/dev/urandom`. The server time in Table 3 measures CPU time only; all other times are “wall time” in actual seconds for operation completion.

6.4 Audit compared to checksums

For the first set of experiments, we 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.

Table 3: 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

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.

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, 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 3.

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

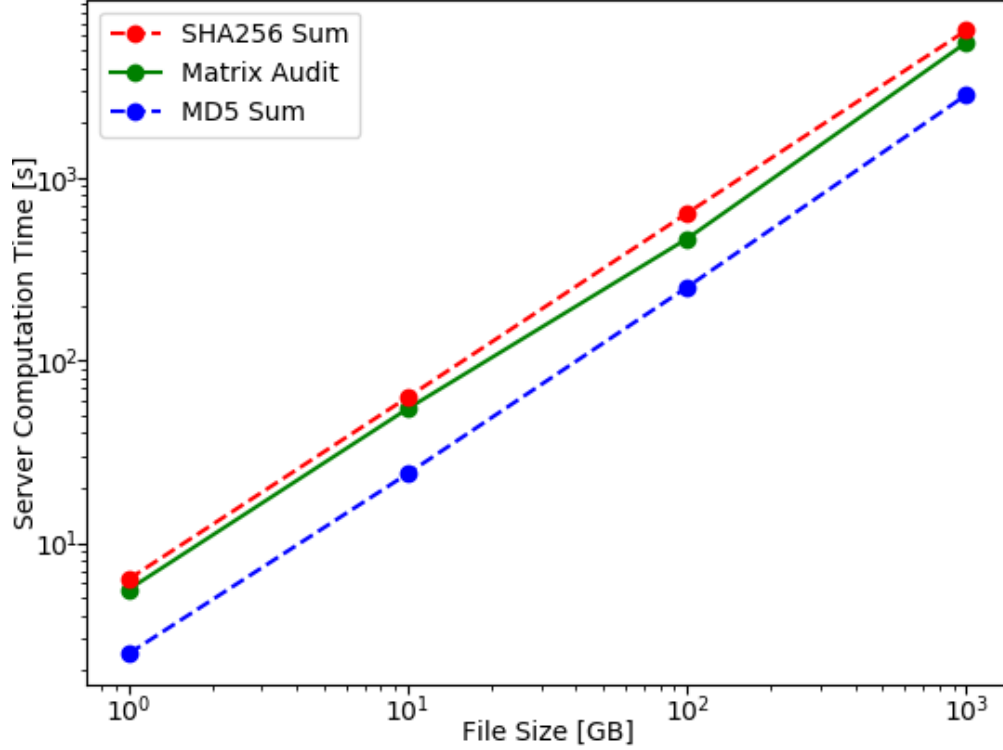


Figure 12: Run Times of the Audit Protocol for Varying File Sizes (Log-Log Scale)

the two dot products is minimal.

There are two main conclusions to draw from the experiments. The first deals with our 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. Figure 12 clearly depicts that the running time is consistently between that of MD5 and SHA256 checksums, both in wall time and CPU time. This justifies the fact 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.

We were surprised by the large disparity between the Server Time and the Wall Time in these experiments, both for our own Audit protocol and for the checksum comparisons. We determined that this disparity is mostly due to I/O within the cloud datacenter, caused by the CPU waiting for the reads to the external drive.

6.5 Parallel audits using MPI

Our first round of experiments indicated that our **Audit** algorithm on the server was I/O bound, despite the favorable linear access pattern of the matrix-vector product computation. It seems that Google Cloud Platform throttles disk-to-VM I/O on a per-VM basis, so that even with many cores, the situation did not improve.

However, we were able to achieve good parallel speedup when running the **Audit** algorithm over multiple VMs in parallel using MPI. In this setup, the server VM waits for a connection from a client, who requests an audit, which is in turn performed by some number of VMs running in parallel, after which the results are collected and returned to the client. The simplicity of our **Audit** protocol makes it trivially parallelizable, where each parallel VM performs the matrix-vector product on a contiguous subset of rows of $\mathbf{D}$, corresponding to a contiguous segment of the underlying file.

Because the built-in MD5 and SHA256 checksum programs do not achieve any parallel speedup, we focused only on our **Audit** protocol for this set of experiments using MPI. The results are reported in Table 4. Our parallel speedup is not quite linear, but was sufficient to gain a significant improvement in the audit time, to just under 16 minutes in the case of a 1TB file using 16 VMs. Figure 13 shows how the parallelization affected the run times of **Audit**. As stated, the comparison here with MD5 and SHA256 are not meaningful as MD5 and SHA256 have not been altered to be run in parallel; however, the graph is used as a means of comparison against the original test results.

Table 4: Multiple Machine Parallelization Results

VMs	Metric	1GB	10GB	100GB	1TB
1	Audit (s)	12.75	117.36	1080.66	11495.00
	Speedup	1x	1x	1x	1x
	Cost	\$0.0004	\$0.0031	\$0.0285	\$0.3033
4	Audit (s)	3.99	46.28	430.29	3512.33
	Speedup	3.19x	2.54x	2.51x	3.27x
	Cost	\$0.0003	\$0.0037	\$0.0341	\$0.2781
16	Audit (s)	3.63	14.34	117.92	946.74
	Speedup	3.51x	8.18x	9.16x	12.14x
	Cost	\$0.0009	\$0.0034	\$0.0280	\$0.2249

We also used these times to measure the total cost of running each audit in Google Cloud Platform, which features per-second billing of VMs and persistent disks, as reported in Table 4 as well. Note that the monetary cost for increasing parallel VMs is slightly decreasing for larger file sizes, indicating that even higher levels of parallelization may decrease the running time even further with no extra monetary cost.

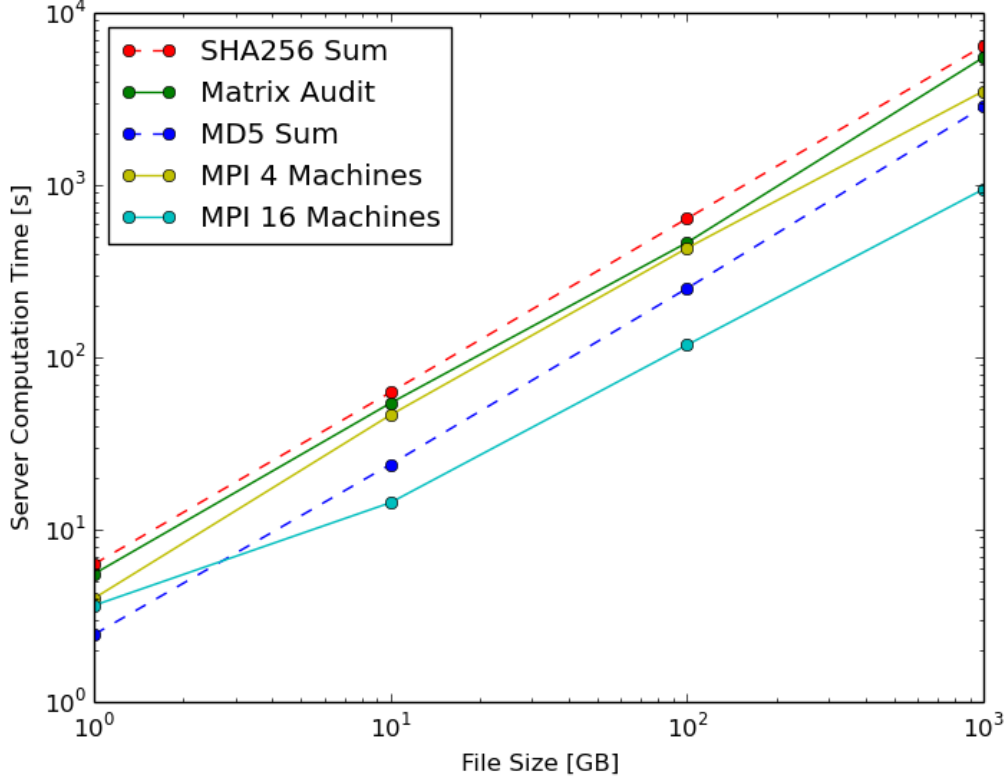


Figure 13: Relative Audit Time Improvements with MPI Parallelization (Log-Log Scale)

6.6 Communication and Client Computation Time

Besides the $O(N)$ complexity for server computation during an audit, the $O(\sqrt{N})$ cost of client computation and communication bandwidth in our scheme is also asymptotically worse than existing PoR schemes. However, our experiments suggest that in practice these are not significant factors.

The time it took the client to compute the two dot products to finish the audit never took more than 0.12 seconds in any case tested. This indicates that even low-powered client machines should be able to run this audit protocol without issue.

The time spent communicating the challenge and response vectors, $\mathbf{c}$ and $\mathbf{v}$, becomes insignificant in comparison to the server computation as the size of the database increases. In the case of our experiments, Table 5 summarizes that communication time of both $\mathbf{c}$ and $\mathbf{v}$ remains under five seconds. The amount of data communicated is also given to confirm the square root scaling.

Table 5: Amount of Communication Per Audit with 16 VMs

Metric	1GB	10GB	100GB	1TB
Comm. (kB)	358	1131	3578	11314
Time (s)	2.05	1.68	3.87	4.04

Our experiments had both the client and server (with associated added VMs) co-located in the us-central1-a zone (listed geographically as Iowa), meaning that the communication is likely within the same physical datacenter. We hope to address this in future work with geographically diverse clients; however, the small amount of data being transferred indicates that this will still not have a significant affect on the overall audit time.

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 \$42.72. Previous state of the art requiring 6x storage of the data size costs \$240 per month in this scenario. This is a 82% cost savings from storage. Each audit at this size costs \$0.23, taking only 16 minutes running on 16 VMs. Even if we consider that audits for the previous state of the art are free, there are clear savings for using our protocol, as the client determines the number of audits necessary.

7 Use-Based Optimizations

While this work, as seen in the theory and confirmed by the experimentation, shows improvement in the field of PoR, we have identified modifications to the protocols that could serve beneficial in certain situations. Each institution, company, or individual that may deploy this work has different goals due to different network layouts and varied levels of dispersion across the globe. Some users may prioritize having as little stored on the client as possible, while others may prioritize efficient communication. We present the following as possible configurations to aid in optimizing individual situations.

7.1 Structured Vectors

Though client storage is $O(\sqrt{N})$, there may be situations where this needs to be even less. In addition, communication cost of the same may need to be optimized further. One way in which to accomplish this is to redefine how $\mathbf{r}$ and $\mathbf{c}$ is formed. As of now, it is m distinct random values that are created with `Init`. Instead of using uniformly random vectors $\mathbf{r}$ and $\mathbf{c}$, one can impose a structure on them, in order to reduce the amount of randomness needed, and the cost of communicating or storing them. We propose to apply Kimbrel and Sinha's modification of Freivalds' check [13]: select a single random field element ρ and raise the power by 1 to determine the next value in $\mathbf{c}$. Thus, reducing the communication volume for an audit from $m + n$ to $m + 1$ field elements. Similarly, we can reduce the storage of $\mathbf{r}$. This reduces the storage on the client side from $m + n$ to only $n + 1$ field elements.

$$\mathbf{c}^\top = [\rho, \rho^2, \dots, \rho^n] \in \mathbb{F}_q^n$$

$$\mathbf{r}^\top = [s, s^2, \dots, s^m] \in \mathbb{F}_q^m$$

7.2 Non-Square Matrix Dimensions

Thus far, we have used a square matrix to represent the data, or $n = m$. Instead, using a rectangular matrix yields interesting effects. Though in this day and age most networks are plenty fast, if there is a concern of download versus upload speed, changing the matrix dimensions can optimize one over the other. In an audit, n elements are uploaded to the server, and m elements are downloaded from the server. Setting $n < m$ would limit the upload needed.

Using the above scheme of structured vectors, uploads would be minimized to one element, allowing to test $n > m$. Limiting the length of $\mathbf{s}$, through a small m , would optimize the communication cost. This could be targeted to any communication cost required.

This does adversely affect the cost of client storage and client computation, both of which are $n + m$.

For a matrix, we substitute $m = \frac{N}{n}$, making the cost $n + \frac{N}{n}$. This cost is minimized when $n = m = \sqrt{N}$; thus, when $n \neq m$ the cost increases.

7.3 Externalization

In some situations, it may be advantageous to not have the client store the vectors $\mathbf{r}$ and $\mathbf{s}$ at all. This can be done by storing those vectors, as well as $\mathbf{D}$ and the Merkle tree, on the server, or externalizing the local storage. The client storage can be reduced to $O(\lambda)$ by externalizing the storage of the block-vector $\mathbf{s}$ at the expense of increasing the volume of communication. Clearly $\mathbf{s}$ must be stored encrypted, as otherwise the server could answer any challenge without having to store the database. Any IND-CPA symmetric cipher works here, with care taken so that a separate IV is used for each column; this allows updates to a column of $\mathbf{s}$ during a **Write** operation without revealing anything about the updated values.

We will simply assume that the client has access to an encryption function and a decryption function. In order to assess the authenticity of each communication of $\mathbf{s}$ from the Server to the client, we will use another Merkle-Hash tree certificate for it: the client will only need to keep the root of a Merkle-Tree built on the encryption of $\mathbf{s}$.

Since this modification reduces the client storage but increases the overall communication, both options (with or without it) should be considered, and we will state the algorithms for our protocol with a *Strategy* parameter, deciding whether or not to externalize the storage of $\mathbf{s}$.

8 Comparison to State of the Art

PDP schemes, first introduced by [2] in 2007, originally only considered static data storage. The original scheme was later adapted to allow dynamic updates by [5] 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 [3], provide a stronger guarantee of integrity: namely, that any small alteration to the data is likely to be detected. In this paper, we use the term PoR to refer to any scheme which provides this stronger level of recoverability guarantee.

PoR and PDP are usually constructed as a collection of phases in order to initialize the data storage, to access it afterwards and to audit the server's storage. Dynamic schemes also propose a modification of subsets of data, called write.

Since 2007, different schemes have been proposed to serve different purposes such as data confidentiality, data integrity, or data availability, but also freshness and fairness. Storage efficiency, communication efficiency and reduction of disk I/O have improved with time. Some schemes are developed for static data (no update protocol), others extend their audit protocol for public verification, still others require a finite number of Audits and Updates. For a complete taxonomy on recent PoR schemes, see [14] and references therein.

For our purpose, we have identified two main storage outsourcing type of approaches: those which minimizes the storage overhead and those which minimize the client and server computation. For each approach, we specify in Table 6 which one meets various requirements such as whether or not they are dynamic, if they can answer an unbounded number of queries and what is the extra storage they require.

Table 6: Attributes of some selected schemes

Protocol	PoR capable	Number of audits	Number of updates	Extra Storage
Sebé [9]	X	∞	0	$o(N)$
Ateniese et al. [2]	X	∞	0	$o(N)$
Ateniese et al. [15]	X	K	K	$o(N)$
Storj [16]	✓	K	K	$o(N)$
Juels et al. [3]	✓	K	0	$O(N)$
Lavauzelle	✓	∞	0	$O(N)$
Stefanov et al. [17]	✓	∞	∞	$O(N)$
Cash et al. [4]	✓	∞	∞	$O(N)$
Shi et al. [1]	✓	∞	∞	$O(N)$
Here	✓	∞	∞	$o(N)$

8.1 Low storage overhead

The schemes of Ateniese et al. [2] or Sebé et al. [9] are in the PDP model. Both of them have a storage overhead in $o(N)$. They use the RSA protocol in order to construct homomorphic authenticators, so that a successful audit guarantees data possession on some selected blocks. When all the blocks are selected, the audit is deterministic but the computation cost is high. So in practice, [2] minimizes the file block accesses, the computation on the server, and the client-server communication. For one audit on at most f blocks, the S-PDP protocol of [2] gives the costs seen in Table 7. A robust auditing integrates S-PDP with a forward error-correcting codes to mitigate arbitrary small file corruption. Nevertheless, if the server passes one audit, it guarantees only that a portion of the data is correct.

Table 7: S-PDP on f blocks : The file M is composed of N/b blocks of bit-size b . The computation is made mod Q , where Q is the product of two large prime numbers.

		Server	Communication	Client
Storage		$N + m$		$O(1)$
Comput.	Setup		$N + f$	$O(bf)$
	Audit	$O(f)$	$\leftarrow O(1)$	$O(f)$
			$\rightarrow O(1)$	

Later, Ateniese et al. [15] proposed a scheme secure under the random oracle model based on hash functions and symmetric keys. It has an efficient update protocol but uses tokens which impose a limited number of audits or updates.

Alternatively, verifiable computing can be used to go through the whole database with Merkel hash trees, as in [18, §6]. The latter proposition however comes with a large overhead in homomorphic computations and does not provide an Audit mechanism. Verifiable computing can provide an audit mechanism, as sketched in the following paper [7], but then it is not dynamic anymore.

Storj [16] (version 2) is a very different approach also based on Merkel hash trees. It is a dynamic PoR protocol with bounded Audits and updates. The storage is encrypted and cut into m blocks of size b . For each block and for a selection of K salts, a Merkel Hash tree with K leaves is constructed. The efficiency of [16] is presented Table 8.

8.2 Fast audits but large extra storage

PoR methods based on block erasure encoding are a class of methods which guarantee with a high probability that the client's entire data can be retrieved. The idea is to check the authenticity of a number of erasure encoding blocks during the data recovery step but also during the audit protocol. Those approaches will not

Table 8: Storj-V2: The file M is composed of N/b blocks of bit-size b . K is the number of salts.

		Server	Communication	Client
	Storage	$N + O(\frac{N}{b}K)$		$O(\frac{N}{b}K)$
Comput.	Setup		$\leftarrow N + O(\frac{N}{b}K)$	$O(N + \frac{N}{b}K)$
	Audit	$O(\frac{N}{b}K)$	$\leftarrow O(\frac{N}{b})$	$O(\frac{N}{b} \log K)$
			$\rightarrow O(\frac{N}{b} \log K)$	
	Update	$O(K)$	$\leftarrow O(1) + b$	$O(\log K)$
			$\rightarrow O(\log K)$	

detect a small amount of corrupted data. But the idea is that if there are very few corrupted blocks, they could be easily recovered via the error correcting code.

Lavauzelle et al., [19] proposed a static PoR. The Init protocol consists in encoding the file using a lifted q -ary Reed-Solomon code and encrypting it with a block-cipher. The Audit protocol checks if one word of q blocks belongs to a set of Reed-Solomon code words. This protocol has to succeed a sufficient number of times to ensure with a high probability that the file can be recovered. Its main drawback is that it requires an initialization quadratic in the database size. For a large data file of several terabytes this becomes intractable.

In addition to a block erasure code, PoRSYS of Juels et al. [3] use block encryptions and sentinels in order to store static data with a cloud server. Shacham and Waters [6] use authenticators to improve the audit protocol. A publicly verifiable scheme based on the Diffie-Hellman problem in bilinear groups is also proposed.

Stefanov et al. [17] were the first to consider a dynamic PoR scheme. Later improvements by Cash et al. or Shi et al. [4, 1] allow for dynamic updates and reduce the asymptotic complexity (see Table 9). 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.

Table 9: Shi et al. [1]: The file M is composed of $\frac{N}{b}$ blocks of bit-size b .

		Server	Communication	Client
	Storage	$O(N)$		$O(b)$
Comput.	Setup		$\leftarrow N + O(\frac{N}{b})$	$O(N \log N)$
	Audit	$O(b \log N)$	$O(b + \log N)$	$O(b + \log N)$
	Update	$O(b \log N)$	$O(b + \log N)$	$O(\log N)$

Table 10 compares the additional server storage and audit costs between [1] and three variants of our protocol: the first one saving on communication, the second one externalizing the storage of the client vector(s) $\mathbf{r}$ and $\mathbf{s}$, and the third one following the normal protocols outlined in Section 4. In the first case, an arbitrary parameter α can be used in the choice of the dimensions: $m = N^\alpha$ and $n = N^{1-\alpha}/\log_2(q)$.

Table 10: Comparison with Shi et al. [1]: The audit complexity depends on the number of blocks.

	Shi et al. [1]	Here Change Dimen.	Here Externalize
Server extra storage	$5N$	$o(N)$	$o(N)$
Server audit cost	$O(b \log N)$	$N + o(N)$	$N + o(N)$
Communication	$O(b + \log N)$	$O(N^\alpha)$	$O(\sqrt{N})$
Client audit cost	$O(b + \log N)$	$O(N^{1-\alpha})$	$O(\sqrt{N})$
Client storage	$O(b)$	$O(N^{1-\alpha})$	$O(1)$

This balances between the communication cost $O(N^\alpha)$ and the Client computation and storage $O(N^{1-\alpha})$.

Note that efficient solutions to PoR for dynamics data do not consider the confidentiality of the file M , but assume that the user can encrypt its data in a prior step if needed.

9 The Way Ahead

9.1 Applications

Our protocol builds on other academic work in the field of PoR and PDP. It also offers real world application. One main reason to continue development in the same field is to eventually have a solution to a problem that can be accepted in a more widespread matter. As our protocol has developed and changed, so have the practical applications. There are two main applications which are well suited for our protocol.

First, the need for integrity guarantees becomes even more relevant in decentralized storage networks (DSNs), where a user's data is distributed across a heterogeneous network of storage providers. Examples of DSNs currently in use include Sia [20], Storj [16], and Filecoin [21]. Typically, a data storage provider is paid (in cryptocurrency) for storing some other users' data. While these providers can be large tech companies, they can also be any individual with a computer willing to dedicate memory to a DSN. Here the storage providers have a clear monetary incentive to cheat by deleting user data, perhaps if that data has not recently been accessed, and attempting to thwart any integrity check mechanism. Our integrity check could be run by the maintainers of the DSN to ensure all providers are acting as expected. It could also be an option for a client of the DSN to use our integrity check to check their dispersed data specifically.

Second, anyone should be able to check a small store of data quickly. Specifically, this data should have a completely trusted backup, as our protocol does not offer a recovery method. However, a test of whether a store has been corrupted or not is extremely useful in systems where their trusted backup could take a long time to access. This is common for the DoD specifically.

As mentioned, the JEDI cloud contract will push cloud storage and computing power further out to the end user. This will certainly aid in lethality, but data such as actionable intelligence needs to be correct. An integrity check of surveillance photos, standing orders, launch coordinates, or enemy location should be taken to ensure that an enemy cannot spoof anything of this level of importance. If the audit returns a fail, then a reassessment can take place without taking action on corrupt data. The originals are still stored somewhere safe and can be regained in time, but the integrity check in the moment creates a quick and provable layer of confirmation.

9.2 Future Work

While the work we offer is self-contained and complete, there are always improvements to be made and adaptations to create in order to further research in the field, as we have done in the realm of integrity checks and PoRs. It is important to note that our protocol does not depend on formatted data. The data

can be plaintext, encrypted, stored in one file or many. Our protocol simply views all of the data on the bit level.

As mentioned in Section 6.6, a useful test to run on our protocol would be to host the client machine in a different geographic region (perhaps Europe or Asia) than the server. This is a valid parallel of the real world to test. While the communication is so insignificant to begin with, we do not suspect that communication will time drastically reduce performance. This test would allow for proof in one direction or the other.

The next step to take would be allowing this protocol to work over an entire hard drive (any storage device). As of now, it runs over a file (files can be concatenated to do multiple). By creating a Network Block Device (NBD), our protocol can be made to look over an entire file system with no change to the client.

As we have worked on a working implementation, an optimistic next step would be further development to allow this to be offered by Cloud providers. As we have stated, Integrity is not guaranteed by these providers. If our protocol was adopted by the providers to be used across their networks, it would ultimately provide more security to the client, all of us.

References

- [1] Elaine Shi, Emil Stefanov, and Charalampos Papamanthou. Practical dynamic proofs of retrievability. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security*, CCS '13, pages 325–336, New York, NY, USA, 2013. ACM.
- [2] Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary Peterson, and Dawn Song. Provable data possession at untrusted stores. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 598–609. Acm, 2007.
- [3] Ari Juels and Burton S Kaliski Jr. Pors: Proofs of retrievability for large files. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 584–597. Acm, 2007.
- [4] David Cash, Alptekin Küpçü, and Daniel Wichs. Dynamic proofs of retrievability via oblivious RAM. *J. Cryptol.*, 30(1):22–57, January 2017.
- [5] C. Chris Erway, Alptekin Küpçü, Charalampos Papamanthou, and Roberto Tamassia. Dynamic provable data possession. *ACM Trans. Inf. Syst. Secur.*, 17(4):15:1–15:29, April 2015.
- [6] Hovav Shacham and Brent Waters. Compact proofs of retrievability. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 90–107. Springer, 2008.
- [7] Dario Fiore and Rosario Gennaro. Publicly verifiable delegation of large polynomials and matrix computations, with applications. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 501–512, New York, NY, USA, 2012. ACM.
- [8] Rūsiņš Freivalds. Fast probabilistic algorithms. In J. Bečvář, editor, *Mathematical Foundations of Computer Science 1979*, volume 74 of *Lecture Notes in Computer Science*, pages 57–69, Olomouc, Czechoslovakia, September 1979. Springer-Verlag.
- [9] Francesc Sebé, Josep Domingo-Ferrer, Antoni Martínez-Ballesté, Yves Deswarte, and Jean-Jacques Quisquater. Efficient remote data possession checking in critical information infrastructures. *IEEE Transactions on Knowledge and Data Engineering*, 20:1034–1038, 2008.
- [10] Ralph C. Merkle. A digital signature based on a conventional encryption function. In Carl Pomerance, editor, *Advances in Cryptology — CRYPTO '87*, pages 369–378, Berlin, Heidelberg, 1988. Springer Berlin Heidelberg.
- [11] B. Laurie, A. Langley, E. Kasper, and Google. Certificate Transparency. RFC 6962, IETF, June 2013.
- [12] W. B. Hart. Fast Library for Number Theory: An Introduction. In *Proceedings of the Third International Congress on Mathematical Software*, ICMS'10, pages 88–91, Berlin, Heidelberg, 2010. Springer-Verlag. <http://flintlib.org>.
- [13] Tracy Kimbrel and Rakesh Kumar Sinha. A probabilistic algorithm for verifying matrix products using $O(n^2)$ time and $\log_2 n + O(1)$ random bits. *Information Processing Letters*, 45(2):107–110, February 1993.
- [14] Choon Beng Tan, Mohd. Hanafi Ahmad Hijazi, Yuto Lim, and Abdullah Gani. A survey on proof of retrievability for cloud data integrity and availability: Cloud storage state-of-the-art, issues, solutions and future trends. *J. Network and Comp. Applications*, 110:75–86, 2018.
- [15] Giuseppe Ateniese, Roberto Di Pietro, Luigi V Mancini, and Gene Tsudik. Scalable and efficient provable data possession. In *Proceedings of the 4th international conference on Security and privacy in communication networks*, page 9. ACM, 2008.
- [16] "paper@storj.io". Storj: A decentralized cloud storage network framework. Technical report, Storj labs Inc., 2018.

- [17] Emil Stefanov, Marten van Dijk, Ari Juels, and Alina Oprea. Iris: A scalable cloud file system with efficient integrity checks. In *Proceedings of the 28th Annual Computer Security Applications Conference*, pages 229–238. ACM, 2012.
- [18] Siavosh Benabbas, Rosario Gennaro, and Yevgeniy Vahlis. Verifiable delegation of computation over large datasets. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, pages 111–131, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.
- [19] Julien Lavauzelle and Françoise Levy dit Vehel. New proofs of retrievability using locally decodable codes. In *2016 IEEE International Symposium on Information Theory (ISIT)*, pages 1809–1813, July 2016.
- [20] Zach Herbert. "the ico paradox — and how to fix it". *Sia blog*, 2018.
- [21] Filecoin specification. Technical report, Filecoin, 2020.
- [22] Cong Wang, Qian Wang, Kui Ren, and Wenjing Lou. Privacy-preserving public auditing for data storage security in cloud computing. In *2010 proceedings ieee infocom*, pages 1–9. Ieee, 2010.
- [23] Richard A. DeMillo and Richard J. Lipton. A probabilistic remark on algebraic program testing. *Inf. Process. Letters*, 7(4):193–195, June 1978.
- [24] Richard Zippel. Probabilistic algorithms for sparse polynomials. In Ng [26], pages 216–226.
- [25] Jacob T. Schwartz. Probabilistic algorithms for verification of polynomial identities. In Ng [26], pages 200–215.
- [26] Edward W. Ng, editor. *EUROSAM '79, International Symposium on Symbolic and Algebraic Computation, Marseille, France, June 1979, Proceedings*, volume 72 of *Lecture Notes in Computer Science*. Springer, 1979.
- [27] Jean-Guillaume Dumas, Pascal Giorgi, and Clément Pernet. Dense linear algebra over prime fields. *ACM Transactions on Mathematical Software*, 35(3):1–42, November 2008.
- [28] Martin Albrecht, Gregory Bard, and William Hart. Algorithm 898: Efficient multiplication of dense matrices over $\text{GF}(2)$. *ACM Trans. Math. Softw.*, 37(1):9:1–9:14, January 2010.
- [29] Jean-Guillaume Dumas and Vincent Zucca. Prover efficient public verification of dense or sparse/structured matrix-vector multiplication. In Josef Pieprzyk and Suriadi Suriadi, editors, *ACISP 2017, 22nd Australasian Conference on Information Security and Privacy*, volume 10343 of *Lecture Notes in Computer Science*, pages 115–134. Springer, July 2017.
- [30] Jean-Guillaume Dumas, Erich Kaltofen, and Emmanuel Thomé. Interactive certificate for the verification of Wiedemann’s Krylov sequence: application to the certification of the determinant, the minimal and the characteristic polynomials of sparse matrices. Technical report, IMAG-hal-01171249 arXiv cs.SC/1507.01083, January 2016.
- [31] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, pages 223–238, 1999.
- [32] Laurent Fousse, Pascal Lafourcade, and Mohamed Alnuaimi. Benaloh’s dense probabilistic encryption revisited. In Nitaj and Pointcheval [33], pages 348–362.
- [33] Abderrahmane Nitaj and David Pointcheval, editors. *Progress in Cryptology - AFRICACRYPT 2011 - 4th International Conference on Cryptology in Africa, Dakar, Senegal, July 5-7, 2011. Proceedings*, volume 6737 of *Lecture Notes in Computer Science*. Springer, 2011.
- [34] Josh Benaloh. Dense probabilistic encryption. In *First Annual Workshop on Selected Areas in Cryptography*, pages 120–128, Kingston, ON, May 1994.

- [35] Payman Mohassel. Efficient and secure delegation of linear algebra. *IACR Cryptology ePrint Archive*, 2011:605, 2011.
- [36] Emmanuel Bresson, Dario Catalano, and David Pointcheval. A simple public-key cryptosystem with a double trapdoor decryption mechanism and its applications. In Chi-Sung Lai, editor, *Advances in Cryptology - ASIACRYPT 2003*, pages 37–54, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.
- [37] Ralph Charles Merkle. *Secrecy, authentication, and public key systems*. PhD thesis, Stanford University, 1979.
- [38] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In A. M. Odlyzko, editor, *Advances in Cryptology - CRYPTO'86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer-Verlag, 1987, 11–15 August 1986.
- [39] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Assche. Sponge-based pseudo-random number generators. In Stefan Mangard and François-Xavier Standaert, editors, *CHES 2010*, pages 33–47. Springer, 2010.
- [40] NIST. *FIPS Publication 202: SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*, August 2015.
- [41] Yihua Zhang and Marina Blanton. Efficient dynamic provable possession of remote data via balanced update trees. In *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*, ASIA CCS '13, pages 183–194, New York, NY, USA, 2013. ACM.
- [42] Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Osama Khan, Lea Kissner, Zachary Peterson, and Dawn Song. Remote data checking using provable data possession. *ACM Trans. Inf. Syst. Secur.*, 14(1):12:1–12:34, June 2011.
- [43] Michel Abdalla, Fabrice Benhamouda, and Alain Passelègue. An algebraic framework for pseudorandom functions and applications to related-key security. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015*, pages 388–409, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.
- [44] David Naccache and Jacques Stern. A new public key cryptosystem based on higher residues. In *CCS '98*, 1998.
- [45] Faheem Zafar, Abid Khan, Saif Ur Rehman Malik, Mansoor Ahmed, Adeel Anjum, Majid Iqbal Khan, Nadeem Javed, Masoom Alam, and Fuzel Jamil. A survey of cloud computing data integrity schemes: Design challenges, taxonomy and future trends. *Computers & Security*, 65:29–49, 2017.
- [46] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Sakura: A flexible coding for tree hashing. In Ioana Boureanu, Philippe Owesarski, and Serge Vaudenay, editors, *Applied Cryptography and Network Security*, pages 217–234, Cham, 2014. Springer International Publishing.
- [47] Morris J Dworkin. Recommendation for block cipher modes of operation: The xts-aes mode for confidentiality on storage devices. Technical report, 2010. <https://csrc.nist.gov/publications/detail/sp/800-38e/final>.
- [48] Markus Jakobsson, Frank Thomson Leighton, Silvio Micali, and Michael Szydlo. Fractal merkle tree representation and traversal. In Joye [49], pages 314–326.
- [49] Marc Joye, editor. *Topics in Cryptology - CT-RSA 2003, The Cryptographers' Track at the RSA Conference 2003, San Francisco, CA, USA, April 13-17, 2003, Proceedings*, volume 2612 of *Lecture Notes in Computer Science*. Springer, 2003.
- [50] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.
- [51] Nick Harvey. Chernoff bound, balls and bins, congestion minimization. Lecture 3 from CPSC 536N: Randomized Algorithms, 2015.

A Formal Security Proof

Here we give the formal security proof of our matrix-based protocol with required definitions. We give the algorithms for each operation, followed by the formal security proof. Then, Table 11 presents an overview of the formalized protocol with the externalization strategy. This is a merge of the algorithms in Section 3, for the *Externalization* strategy, with the Merkle implementation from Section 4.4.1, using two Merkle hash trees (one for the database D and one for the externalized control vectors $\mathbf{W}$; and thus twice the **MTInit**, **MTVerifiedRead** and **MTVerifiedWrite** protocols).

A.1 Formal Protocol Descriptions

Full definitions of the five algorithms, **Init**, **Read**, **Write**, **Audit** and **Extract**, as Algorithms 1 to 5, are given below, incorporating the improvements on control vector storage from Section 7.3. The **Extract** algorithm is needed for the formal proof, though it is not meant to be implemented or ever run. Rather, it shows that it demonstrates the possibility to complete the proof. They include subcalls to the classical Merkle hash tree operations defined in Section 4.4.1. When using the externalization strategy, E is the encryption function and D is the decryption function noted in Section 7.3.

A.1.1 Ensuring security assumptions via multiple checks

In order to reach a target bound $2^{-\lambda}$ on the probability of failure for the authenticity, it might be necessary to choose multiple independent $\mathbf{r}$ vectors during initialization and repeat the audit checks with each one. First, to ease independence considerations we forget the two prime ring and consider instead that tests are performed in $\mathbb{F}_q$, a finite field of size q . Second, we model multiple vectors by inflating the vectors $\mathbf{r}$ and $\mathbf{s}$ to be blocks of t non-zero vectors instead; that is, matrices $\mathbf{R}$ and $\mathbf{S}$ with t rows each. To see how large t needs to be, consider the probability of the Client accepting an incorrect response during an audit. An incorrect answer $\mathbf{z}$ to the audit fails to be detected only if

$$\mathbf{R} \cdot (\mathbf{z} - \mathbf{v}) = \mathbf{0}, \quad (3)$$

where $\mathbf{v} = \mathbf{D}\mathbf{c}$ is the correct response which would be returned by an honest Server.

If $\mathbf{R}$ is sampled uniformly at random among matrices in $\mathbb{F}_q^{t \times m}$ with non-zero rows, then since the Server never learns any information about $\mathbf{R}$, audit fails only if the Server can guess a vector in the right nullspace of $\mathbf{R}$. This happens with probability at most $1/q^t$.

Achieving a probability bounded by $2^{-\lambda}$, requires to set $t = \left\lceil \frac{\lambda}{\log_2(q)} \right\rceil$. In practice, reasonable values of $\lambda = 128$ and $q > 2^{64}$ mean that $t \leq 2$ in this case.

A.2 Security Proof

Before we begin the full security proof, we need the following technical lemma to prove that the **Extract** algorithm succeeds with high probability. The proof of this lemma is a straightforward application of Chernoff bounds.

Lemma 1. *Let $\lambda, n \geq 1$ and suppose $2e$ balls are thrown independently and uniformly into q bins at random. If $e = 2n + 12\lambda$ and $q \geq 8e$, then with probability at least $\exp(-\lambda)$, the number of non-empty bins is at least $e + n$.*

Proof. Let $B_1, B_2, \dots, B_{2e}$ be random variables for the indices of bins that each ball goes into. Each is a uniform independent over the q bins.

Let $X_{1,2}, X_{1,3}, \dots, X_{2e-1,2e}$ be $\binom{2e}{2}$ random variables for each pair of indices i, j with $i \neq j$, such that $X_{i,j}$ equals 1 iff $B_i = B_j$. Each $X_{i,j}$ is a therefore Bernoulli trial with $\mathbb{E}[X_{i,j}] = \frac{1}{q}$, and the sum $X = \sum_{i \neq j} X_{i,j}$ is the number of pairs of balls which go into the same bin.

We will use a Chernoff bound on the probability that X is large. Note that the random variables $X_{i,j}$ are *not* independent, but they are negatively correlated: when any $X_{i,j}$ equals 1, it only *decreases* the conditional expectation of any other $X_{i',j'}$. Therefore, by convexity, we can treat the $X_{i,j}$'s as independent in order to obtain an upper bound on the probability that X is large.

Algorithm 1 $\text{Init}(1^\lambda, m, n, q, b, M, \text{Strategy})$

Input: $1^\lambda, m, n, q, b \in \mathbb{N}; \mathbf{D} \in \mathbb{F}_q^{m \times n}$
Output: st_S, st_C

```

1:  $t \leftarrow \lceil \lambda / (\log_2 q) \rceil \in \mathbb{N}$ ;
2: Client:  $\mathbf{s} \xleftarrow{\$} \mathbb{F}_q^t$  with non-zero distinct elements {Secrets}
3: Client: Let  $\mathbf{R} \leftarrow [\mathbf{s}_i^j]_{i=1\dots t, j=1\dots m} \in \mathbb{F}_q^{t \times m}$ 
4: Client:  $\mathbf{S} \leftarrow \mathbf{R}\mathbf{D} \in \mathbb{F}_q^{t \times n}$  {Secretly stored or externalized}
5: Both:  $(\mathbf{D}, T_{\mathbf{D}} \mid r_{\mathbf{D}}) \leftarrow \text{MTInit}(1^\lambda, b, \mathbf{D})$ 
6: if ( $\text{Strategy} = \text{externalization}$ ) then
7:   Client:  $K \xleftarrow{\$} \mathcal{K}$ ;
8:   Client:  $\mathbf{W} \leftarrow E_K(\mathbf{S}) \in \mathcal{C}_q^{t \times n}$ ;
9:   Client: sends  $m, n, q, \mathbf{D}, \mathbf{W}$  to the Server;
10:  Both:  $(\mathbf{W}, T_{\mathbf{W}} \mid r_{\mathbf{W}}) \leftarrow \text{MTInit}(1^\lambda, b, \mathbf{W})$ 
11:  Server:  $st_S \leftarrow (m, n, q, \mathbf{D}, T_{\mathbf{D}}, \text{Strategy}, \mathbf{W}, T_{\mathbf{W}})$ ;
12:  Client:  $st_C \leftarrow (m, n, q, t, \mathbf{s}, r_{\mathbf{D}}, \text{Strategy}, K, r_{\mathbf{W}})$ ;
13: else
14:   Client: sends  $m, n, q, \mathbf{D}$  to the Server;
15:   Server:  $st_S \leftarrow (m, n, q, \mathbf{D}, T_{\mathbf{D}}, \text{Strategy})$ ;
16:   Client:  $st_C \leftarrow (m, n, q, t, \mathbf{s}, r_{\mathbf{D}}, \text{Strategy}, \mathbf{S})$ ;
17: end if

```

Algorithm 2 $\text{Read}(st_S, st_C, i, j)$

Input: $st_S, st_C, i \in [1..m], j \in [1..n]$
Output: $\mathbf{D}_{ij}$ or **reject**

```

1: Both:  $\mathbf{D}_{ij} \leftarrow \text{MTVerifiedRead}(\mathbf{D}, T_{\mathbf{D}} \mid (i, j), r_{\mathbf{D}})$ 
2: Client: return  $\mathbf{D}_{ij}$ 

```

Observe that $\mathbb{E}[X] = \binom{2e}{2}/q < e/4$. A standard consequence of the Chernoff bound on sums of independent indicator variables tells us that $\Pr[X \geq 2\mathbb{E}[X]] \leq \exp(-\mathbb{E}[X]/3)$; see for example [50, Theorem 4.1] or [51, Theorem 1].

Substituting the bound on $\mathbb{E}[x]$ then tells us that $\Pr[X \geq e/2] \leq \exp(-e/12) < \exp(-\lambda)$. That is, with high probability, fewer than $e/2$ pair of balls share the same bin. If n_k denotes the number of bins with k balls, the number of non-empty bins is

$$\sum_{k=2}^q n_k + 2e - \sum_{k=2}^q kn_k = 2e - \sum_{k=2}^q (k-1)n_k \geq 2e - \sum_{k=2}^q \binom{k}{2} n_k > \frac{3}{2}e,$$

which completes the proof. □

We now proceed to the main result of the paper.

Theorem 2. *Let $\lambda, m, n \in \mathbb{N}$, $\mathbb{F}_q$ a finite field satisfying $q \geq 16n + 96\lambda$ be parameters for our PoR scheme. Then the protocol composed of:*

- the **Init** operations in Algorithm 1;
- the **Read** operations in Algorithm 2;
- the **Write** operations in Algorithm 3;
- the **Audit** operations in Algorithm 4; and
- the **Extract** operation in Algorithm 5 with $e = 2n + 12\lambda$

Algorithm 3 $\text{Write}(st_S, st_C, i, j, \mathbf{D}'_{ij}, \text{Strategy})$

Input: $st_S, st_C, i \in [1..m], j \in [1..n], \mathbf{D}'_{ij} \in \mathbb{F}_q$
Output: st'_S, st'_C or **reject**

```

1: Both:  $(\mathbf{D}', T'_D \mid \mathbf{D}_{ij}, r'_D)$ 
                                      $\leftarrow \text{MTVerifiedWrite}(\mathbf{D}, T_D \mid (i, j), \mathbf{D}'_{ij}, r_D)$ 

2: if ( $\text{Strategy} = \text{externalization}$ ) then
3:   Both:  $\mathbf{W}_{1..t,j} \leftarrow \text{MTVerifiedRead}(\mathbf{W}, T_W \mid (1..t, j), r_W)$ 
4:   Client:  $\mathbf{S}_{1..t,j} \leftarrow D_K(\mathbf{W}_{1..t,j}) \in \mathbb{F}_q^t$ ;
5: end if
6: Client: Let  $\mathbf{R}_{1..t,i} \leftarrow [\mathbf{s}_k^i]_{k=1..t} \in \mathbb{F}_q^t$ 
7: Client:  $\mathbf{S}'_{1..t,j} \leftarrow \mathbf{S}_{1..t,j} + \mathbf{R}_{1..t,i}(\mathbf{D}'_{ij} - \mathbf{D}_{ij}) \in \mathbb{F}_q^t$ ;
8: if ( $\text{Strategy} = \text{externalization}$ ) then
9:   Client:  $\mathbf{W}'_{1..t,j} \leftarrow E_K(\mathbf{S}'_{1..t,j}) \in \mathbb{C}_q^t$ 
10:  Both:  $(\mathbf{W}', T'_W \mid \mathbf{W}_{1..t,j}, r'_W)$ 
                                      $\leftarrow \text{MTVerifiedWrite}(\mathbf{W}, T_W \mid (1..t, j), \mathbf{W}'_{1..t,j}, r_W)$ 

11:  Server: Update  $st'_S$  using  $\mathbf{D}', T'_D, \mathbf{W}'$ , and  $T'_W$ 
12:  Client: Update  $st'_C$  using  $r'_D$  and  $r'_W$ 
13: else
14:  Server: Update  $st'_S$  using  $\mathbf{D}'$  and  $T'_D$ 
15:  Client: Update  $st'_C$  using  $r'_D$  and  $\mathbf{S}'$ 
16: end if

```

satisfies correctness, adaptive authenticity and retrievability as defined in Definitions 1 to 3.

Proof. Correctness comes from the correctness of the Merkle hash tree algorithms, and from the fact that, when all parties are honest, $\mathbf{Rv} = \mathbf{R}\mathbf{Dc} = \mathbf{Sc}$.

For authenticity, first consider the secret control block vectors $\mathbf{R}$ and $\mathbf{S}$. On the one hand, in the local storage strategy, $\mathbf{R}$ and $\mathbf{S}$ never travel and all the communications by the Client in all the protocols are independent of these secrets. On the other hand, in the externalization strategy, $\mathbf{R}$ never travels and $\mathbf{S}$ is kept confidential by the IND-CPA symmetric encryption scheme with key K known only by the client. Therefore, from the point of view of the server, it is equivalent, *in both strategies*, to consider either that these secrets are computed during initialization as stated, or that they are only determined *after* the completion of any of the operations.

Now suppose that the server sends an incorrect audit response $\mathbf{z} \neq \mathbf{Dc}$ which the Client fails to reject, and let $f \in \mathbb{F}_q[X]$ be the polynomial with degree at most $m - 1$ whose coefficients are the entries of $(\mathbf{z} - \mathbf{Dc})$. Then from (3) and (7.1) in the prior discussion, each of the randomly-chosen values $s_1, \dots, s_t$ is a root of this polynomial f . Because f has at most $m - 1$ distinct roots, the chance that a single s_i is a root of f is at most $(m - 1)/q$, and therefore the probability that all $f(s_1) = \dots = f(s_t) = 0$, is at most $(m/q)^t$.

From the choice of $t = \lceil \lambda / \log_2(q/m) \rceil$, the chance that the Client fails to reject an incorrect audit response is at most $2^{-\lambda}$, which completes the proof of authenticity (Definition 2).

For retrievability, we need to prove that Algorithm 5 succeeds with high probability on the last step of the security game from Definition 3. Because of the authenticity argument above, all successful audit transcripts are valid with probability $1 - \text{negl}(\lambda)$; that is, each $\mathbf{v} = \mathbf{Dc}$ in the input to Algorithm 5. This **Extract** algorithm can find an invertible Vandermonde matrix $\mathbf{C} \in \mathbb{F}_q^{n \times n}$, and thereby recover $\mathbf{D}$ successfully, whenever at least n of the values ρ from challenge vectors $\mathbf{c}$ are distinct.

Therefore the security game becomes essentially this: The experiment runs the honest **Audit** protocol $2e = 4n + 24\lambda$ times, each time choosing a value ρ for the challenge uniformly at random from $\mathbb{F}_q$. The adversary must then select e of these audits to succeed, and the adversary wins the game by selecting e of the $2e$ random audit challenges which contain fewer than n distinct ρ values.

This is equivalent to the balls-and-bins game of Lemma 1, which shows that the **Extract** protocol succeeds with probability at least $1 - \exp(-\lambda) > 1 - 2^{-\lambda}$ for any selection of e out of $2e$ random audits. $\square$

Algorithm 4 $\text{Audit}(st_S, st_C, \text{Strategy})$

Input: st_S, st_C **Output:** accept or reject

```

1: Client:  $\rho \xleftarrow{\$} \mathbb{F}_q$ ;
2: Client: sends  $\rho$  to the Server;
3: Let  $\mathbf{c}^\top \leftarrow [\rho^1, \rho^2, \dots, \rho^n]$ 
4: Server:  $\mathbf{v} \leftarrow \mathbf{D}\mathbf{c} \in \mathbb{F}_q^m$ ;  $\{\mathbf{D} \text{ from } st_S\}$ 
5: Server: sends  $\mathbf{v}$  to Client;
6: if ( $\text{Strategy} = \text{externalization}$ ) then
7:   Both:  $\mathbf{W} \leftarrow \text{MTVerifiedRead}(\mathbf{W}, T_{\mathbf{W}} \mid (1..t, 1..n), r_{\mathbf{W}})$ ;
8:   Client:  $\mathbf{S} \leftarrow D_K(\mathbf{W}) \in \mathbb{F}_q^{t \times n}$ 
9: end if
10: Client: Let  $\mathbf{R} \leftarrow [\mathbf{s}_i^j]_{i=1..t, j=1..m} \in \mathbb{F}_q^{t \times m}$ 
11: if ( $\mathbf{R}\mathbf{v} = \mathbf{S}\mathbf{c}$ ) then
12:   Client: return accept
13: else
14:   Client: return reject
15: end if

```

Algorithm 5 $\text{Extract}(st_C, (\mathbf{c}_1, \mathbf{v}_1), \dots, (\mathbf{c}_e, \mathbf{v}_e))$

Input: st_C and $e \geq 2n + 12\lambda$ successful audit transcripts $(\mathbf{c}_i, \mathbf{v}_i)$ **Output:** $\mathbf{D}$ or fail

```

1:  $\ell_1, \dots, \ell_k \leftarrow$  indices of distinct challenge vectors  $\mathbf{c}_{\ell_i}$ 
2: if  $k < n$  then
3:   return fail
4: end if
5: Form matrix  $\mathbf{C} \leftarrow [\mathbf{c}_{\ell_1} \mid \dots \mid \mathbf{c}_{\ell_n}] \in \mathbb{F}_q^{n \times n}$ 
6: Form matrix  $\mathbf{V} \leftarrow [\mathbf{v}_{\ell_1} \mid \dots \mid \mathbf{v}_{\ell_n}] \in \mathbb{F}_q^{m \times n}$ 
7: Compute  $\mathbf{D} \leftarrow \mathbf{V}\mathbf{C}^{-1}$ 
8: return  $\mathbf{D}$ 

```

Table 11: Externalized PoR

	Server	Communications	Client
Init	D with N bits Stores D $T_D \leftarrow \text{MTCreatetree}(\mathbf{D})$ Stores W $T_W \leftarrow \text{MTCreatetree}(\mathbf{W})$	$\xleftarrow{\lambda, m, n, q, b} \mathbf{D}$ $\xleftarrow{\mathbf{W}}$	$\mathbf{D} \in \mathbb{F}_q^{m \times n}$ $r_D \leftarrow \text{MTRootFromLeaves}(\mathbf{D})$ $t \leftarrow \lceil \lambda / \log_2(q) \rceil$ $s \xleftarrow{\\$} S^t \subseteq \mathbb{F}_q^t$ Form $\mathbf{R} \leftarrow [s_j^i]_{i=1..n, j=1..t} \in \mathbb{F}_q^{n \times t}$ $\mathbf{S} \leftarrow \mathbf{R}\mathbf{D}, \mathbf{W} \leftarrow E(\mathbf{S})$ $r_W \leftarrow \text{MTRootFromLeaves}(\mathbf{W})$ discard D, S, W
Read		$\xleftarrow{i, j} \mathbf{D}_{ij, L_D}$	
$\mathbf{D}_{ij}$	$(\mathbf{D}_{i,j}, L_D) \leftarrow \text{MTElementAndPath}(i, j, \mathbf{D}, T_D)$		$r_D \stackrel{?}{=} \text{MTRootFromPath}(i, j, \mathbf{D}_{ij}, L_D)$
Write			
$\mathbf{D}'_{ij}$	$(\mathbf{W}_{i,1..t}, L_W) \leftarrow \text{MTElementAndPath}(i, 1..t, \mathbf{W}, T_W)$ $\mathbf{D}_{ij} \leftarrow \mathbf{D}'_{ij}$ $\mathbf{W}_{i,1..t} \leftarrow \mathbf{W}'_{i,1..t}$	$\xrightarrow{\mathbf{W}_{i,1..t}, L_W} \mathbf{W}$ $\xleftarrow{\mathbf{D}'_{ij}, \mathbf{W}'_{i,1..t}}$	$r_D \leftarrow \text{MTRootFromPath}(i, j, \mathbf{D}'_{ij}, L_D)$ $r_W \stackrel{?}{=} \text{MTRootFromPath}(i, 1..t, \mathbf{W}_{i,1..t}, L_W)$ $\mathbf{S}_{i,1..t} \leftarrow D(\mathbf{W}_{i,1..t})$ $\mathbf{S}'_{1..t, j} \leftarrow \mathbf{S}_{1..t, j} + \mathbf{R}_{1..t, i}(\mathbf{D}'_{ij} - \mathbf{D}_{ij}) \in \mathbb{F}_q^t$ $\mathbf{W}'_{i,1..t} \leftarrow E(\mathbf{S}_{i,1..t})$ $r_W \leftarrow \text{MTRootFromPath}(i, 1..t, \mathbf{W}'_{i,1..t}, L_W)$
Audit	form $\mathbf{c} \leftarrow [\rho, \rho^2, \dots, \rho^m]^\top$ $\mathbf{v}^\top \leftarrow \mathbf{c}^\top \mathbf{D}$	$\xleftarrow{\rho}$ $\xrightarrow{\mathbf{v}, \mathbf{W}}$	$\rho \xleftarrow{\\$} S \subseteq \mathbb{F}_q$ $r_W \stackrel{?}{=} \text{MTRootFromLeaves}(\mathbf{W})$ $\mathbf{S} \leftarrow D(\mathbf{W})$ $\mathbf{v}^\top \mathbf{R} \stackrel{?}{=} \mathbf{c}^\top \mathbf{S}$