S3ORAM: A Demonstration with a Real Dataset

Salvatore Gene Spena
Ying Wu College of Computing
New Jersey Institute of Technology
Newark, United State of America
sgs6@njit.edu

***Abstract*—Oblivious RAM (ORAM) algorithms supply data obfuscation through randomizing the access patterns of a process from the host system. Shamir Secret Sharing ORAM (S3ORAM), advances this idea with Secret Sharing which provides quantum security to the data in the structure. When testing the algorithm against four-line items of a real world TPC-H data table, we found that although the algorithm is not very efficient with its block usage and bulk query access times, it requires minimal resources for end clients to access their data from the servers.**

Keywords—S3ORAM, Oblivious RAM, ORAM, Client to Server Communication, Cyber Security, Cryptography.

I. Introduction

Oblivious RAM (ORAM) algorithms are a proposed type of algorithm that constantly decrypts, moves, and re-encrypts data in a systems RAM to obfuscate sensitive information from the host system by hiding access patterns within a database [1, 3, 4]. ORAM itself is not a new concept for Cyber Security and several different implementations of ORAM have been proposed over the years [1, 3, 4]. These implementations fall under two different categories: either the data is only encrypted, or the data is in Secret Shared form [1, 3, 4]. Encryption Only supplies computational security, which is to say security against a computationally bounded adversary [1]. On the other hand, Shared Secret improves this by creating unconditional security, which makes the system secure regardless of the adversary's computational ability [2, 3, 4]. These algorithm types provide great examples as to the capabilities of ORAM; however, most examples of these algorithms focus purely on the theoretical performance of the algorithm [3, 4]. In this paper, we look at the real-world performance of Shamir Secret Sharing ORAM (S3ORAM) on four anonymized columns of a synthesized TPC-H data table. Using this data, we show the viability of using S3ORAM in a production environment by using actual data tables compared to empty pointer variables. In addition, we bring to light the inefficiency of the data storage in S3ORAM by comparing the "real data" to empty fake blocks [3].

II. Background

A. Secure Secret Sharing

In cryptography, Secret Sharing refers to the distribution of secrets between multiple systems with shares, where each system does not have enough information to reconstruct the secret [2]. There are two different variants of this, secure secret sharing and insecure secret sharing [2]. In the former, all the shares are needed to reconstruct the secret, while the latter could reconstruct the secret with parts of the data [2]. In this paper, we are more interested in secure secret sharing as insecure secrete sharing does not provide the security desired in an ORAM algorithm.

B. S3ORAM

Shamir Secret Sharing ORAM, or S3ORAM, is a form of ORAM that utilizes the Shamir Secret Sharing algorithm to distribute shares amongst multiple ORAM servers [3, 4]. The Shamir Secret Sharing algorithm is a Secure Secret Sharing algorithm, which combines with ORAM to make the database quantum secure [3, 4]. S3ORAM was first proposed by Hoang et al., with the goal of being an efficient ORAM design with low communication overhead, computational overhead, and client block storage [3, 4]. A revised algorithm was later published, however, in this paper we will be looking at the original algorithm [3, 4].

III. Experiments

A. Experiment Design

We use two different experiment designs to evaluate the S3ORAM algorithm against our TPC-H dataset. We specifically wanted to use this as even a synthesized TPC-H dataset supplies actual integers representing data, as opposed to just data pointers to empty variables used in the original implementation. In addition, using this dataset allowed us see how the algorithm performs with multiple columns of data, per block, which the original implementation did not feature.

Each experiment will be ran ten times to generate an average value for our results. These experiments were designed to target a mix of the key benefits of the S3ORAM algorithm to see how they hold true, along with usability aspects that could be used to evaluate how usable S3ORAM is for the real world.

The first experiment looks at five configurations of increasing numbers of total data blocks. In this experiment, we are looking to see how increasing the bucket size and eviction rates corresponds to changes in the initialization time of the S3ORAM data structure, databases storage size, and initialization memory usage. Each configuration has a block size of forty, a height of nine, and uses three servers. Each configuration has a different Bucket Size and Eviction rate, which affects the quantity of Usable Blocks and Total Blocks in the structure as seen in Table I. As originally documented in the S3ORAM paper, the Total Usable Blocks column of Table I is generated by $A \cdot 2^{H-1}$, where A is the Eviction rate and H is the height [3]. In the paper, this is referred to as "real blocks" and reflects blocks where data is actually stored [3]. What is not explicitly mentioned is the fake blocks, or empty blocks which take up the same space as a real block, but does not actually contain any real data. The number of total blocks in the S3ORAM structure can be generated with $(2^{H+1}-1) \cdot Z$ where H is the height of the S3ORAM tree, and Z is the Bucket Size. We are uncertain why there are so many empty blocks in the algorithm, as this was not heavily touched on in the original paper [3].

TABLE I. EXPERIMENT I

Configuration	*Bucket Size*	*Eviction Rate*	*Total Usable Blocks*	*Total Blocks*
A	200	100	51,200	204,600
B	400	200	102,400	409,200
C	2,000	1,000	512,000	2,046,000
D	4,000	2,000	1,024,000	4,092,000

Configuration	***Bucket Size***	***Eviction Rate***	***Total Usable Blocks***	***Total Blocks***
E	8,000	4,000	2,048,000	8,184,000

In the second experiment, we look at the changes in random access time for various quantities of random-access queries. This second experiment uses configuration C as its base configuration to maximize the size of the S3ORAM structure that our hardware was able to run between the client and servers. In this experiment, there are six bulk query requests, as seen in Table II. These queries were ran back to back in order to build up the number of reads to cause an eviction operation.

TABLE II. EXPERIMENT II

Queries	***Starting # of Reads***	***Ending # of Reads***	***Ending # of Evictions***
100	0	100	0
500	100	600	0
1000	600	1,600	0
1000	1,600	2,600	1
5000	2,600	7,600	3

B. *Setup*

In the experiment, we have two machines to perform the tests. A modern AMD Zen3+ based Ryzen laptop and a AMD Zen based Epyc server. The Ryzen laptop will be used as the client machine, and the Epyc server will be used as the three servers. The two devices will directly connect with a gigabit RJ45 connection. In this configuration, data is generated on the client, split into secret shares, directly uploaded to the server via the ethernet connection, and handled by the server instances. From there any queries made to the server follows the same path, and any responses follow the reverse path to the client. Both systems are running Ubuntu 22.04.1 LTS as the OS, with the KDE Desktop Environment, and the latest system updates. Table III lists the client hardware, and Table IV lists the server hardware.

TABLE III. CLIENT

AMD Zen3+ Laptop	
Parts	***Description***
CPU	AMD Ryzen 5 6600U, 28w cTDP
Memory	2x 8 GB DDR5 4800 MHz, 14.9 GB Usable
Storage	InLand 1 TB PCI-E 4.0 SSD

TABLE IV. SERVER

AMD Zen Server	
Parts	***Description***
CPU	AMD Epyc 7401p, 170w cTDP
Memory	8x 8 GB DDr4 2400 MHz, 64 GB Usable
Storage	XPG GAMMIX 512GB S11 Pro 3D NAND PCIe NVMe Gen3x4

We use two separate machines as the client and server to help simulate a more realistic network configuration. It is important to note: the three servers used for S3ORAM will be ran on the same Epyc server. Ideally, this experiment should be performed against multiple virtual machine servers in different parts of the world, however, this was not within our experiments budget. Each instance of the server is provided with a single thread to simulate the reduced number of available threads in a virtual machine.

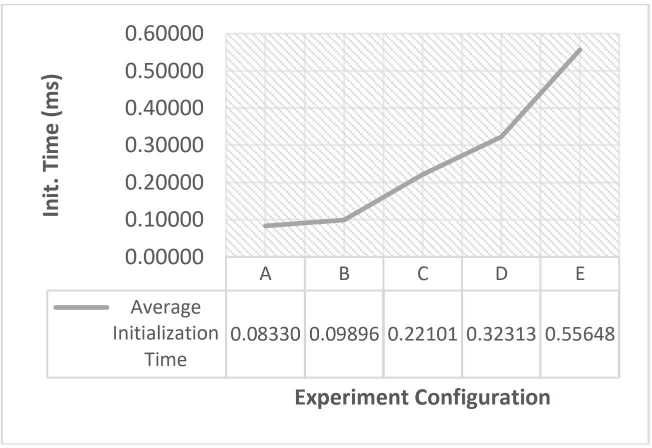
In addition to this, we use a modified version of Dr. Thang Hoang's original 2017 implementation that allows for four Line Items from our TPC-H table to be stored as the data. This modified version also adds the ability to query for values in column A of our data set, and additional feedback to see what real data we retrieved. This modified version should not have modified anything key to the algorithm itself based on the proposed psudocode, rather, it modifies the interface to the algorithm and the source of the data being loaded into the S3ORAM structure. Because this modification does not diverge from the original pseudocode proposed for the S3ORAM algorithm, the integrity of the algorithm should be intact. The modified source code, additional scripts, dataset, and data collected can be found at the following GitHub repository:

https://github.com/sgs6/S3ORAM-Modified

C. *Experiemnt 1 Results*

In our first experiment original we see a gradual exponential increase in the initialization time of the S3ORAM data structure, as seen in Figure 1. This mostly performed as expected, as doubling the amount of data should in theory double the time it takes to initialize. However, when looking from configuration C to D, or D to E, there it was closer to around a sixty to seventy percent increase in initialization time instead of one hundred percent. Our best explanation for this is that there might be some optimizations in the operating systems memory management system or some other low level operation out of our control.

Figure 1 Average Initialization Time (ms)

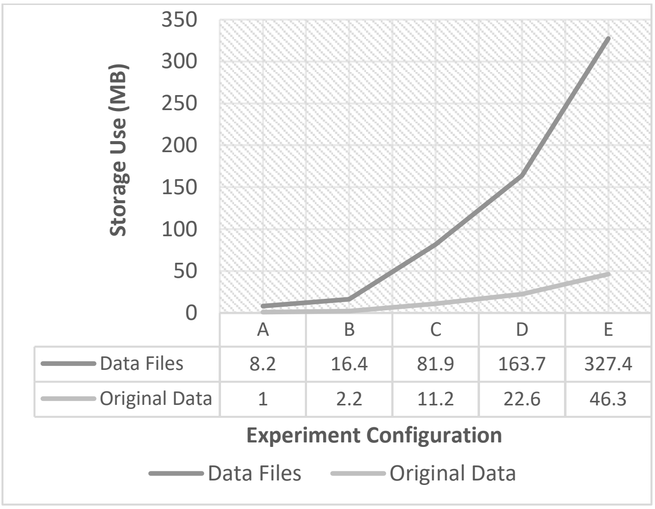


	A	B	C	D	E
Average Initialization Time	0.08330	0.09896	0.22101	0.32313	0.55648

This gradual exponential curve led to cases were doubling the number of blocks in data structure did not directly result in a similar increase to the initialization time. However, there is a limitation where our implementation does not allow for adding or removing data from the structure after the data has been initialized. This results in a quarter of the actual blocks being populated with real data, with the remaining three quarters being dummy data. This is also reflected in the original algorithm, where the block's storing data is referred to as "real blocks" [3]. It is very likely that if we could populate the entirety of the leaf nodes during the initialization of the data, the initialization time may have a stronger correlation to an increase in quantity of blocks. Based on our data, we can assume that this would not be a major bottleneck when using larger datasets with the current implementation.

Next, we looked at the size of the data files generated for the S3ORAM structure, compared to the original size of the data. More specifically, the data files are the actual node files generated when running the algorithm which stores the data, and the original data is a plaintext CSV file containing only the rows of data stored in the S3ORAM nodes. The mean data can be found in a graph comparing tin Figure 2. This graph closer reflects our expectations of a exponential growth compared to our earlier experiment.

Figure 2 Storage Usage vs Original Usage (MB)

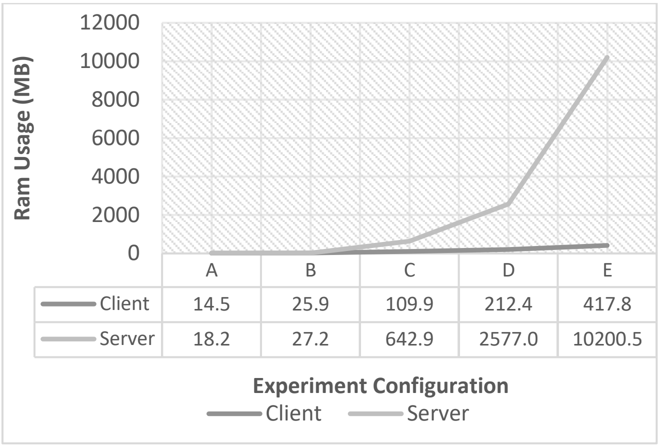


	A	B	C	D	E
Data Files	8.2	16.4	81.9	163.7	327.4
Original Data	1	2.2	11.2	22.6	46.3

When looking at the increase in data storage on the system compared to the original data, we can see that the data files consistently doubled in size as we doubled the total blocks in the structure. The main difference between the data files versus the original file size is how large they start. The data files start at a much higher file size likely due to the number of empty blocks that populate the data structure in the algorithm. This appears to be a major flaw in the design of the S3ORAM algorithm, as the required storage space for this algorithm is approximately seven times the required space of the original csv file.

The last part of our first experiment looks at the initialization memory usage. The mean data can be found in a graph in Figure 3. Once again, the data here appears to double in size each time for the client. However, for the server, things get more interesting. With larger datasets, notably between configuration C through E, the memory quadrupled. This likely is part of the algorithm's optimizations for reduced client resource usage versus server side usage.

Figure 3 Initialization Ram Usage (MB)



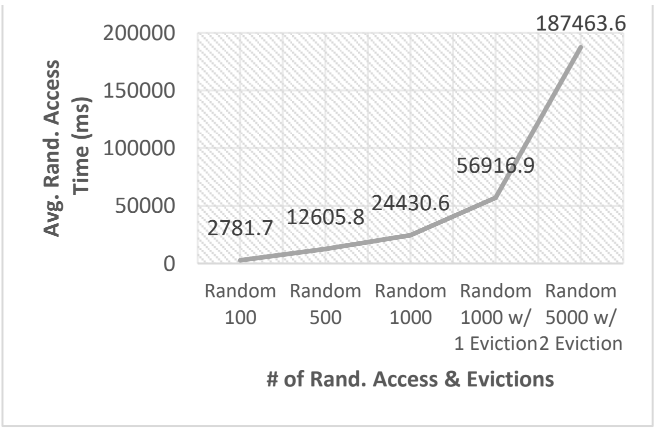
	A	B	C	D	E
Client	14.5	25.9	109.9	212.4	417.8
Server	18.2	27.2	642.9	2577.0	10200.5

Looking deeper, we can see that the client requires much less storage to store the position map compared to the full data tree on the servers. This definitely helps make S3ORAM more viable for clients, as even with over eight million blocks of data on the server, the client only needed to use less than half a gigabyte of memory. This would be a strong selling point of S3ORAM as it is unlikely a client would have a fraction of the memory as the S3ORAM server. One thing that we did note during our tests is that the client memory usage reached a peak of four times the initialization usage. This appears to not be caused by the algorithm itself, but rather uncleared variables that are only used during an eviction process. Multiple evictions did not increase the usage, which confirmed that our modified implementation did not contain a memory leak.

D. *Experiment 2 Results*

In experiment 2, we look at changes in the random access time as the queries become gradually larger. The mean data can be found in Figure 4. Based on our understanding of the algorithm, the data generated makes sense. When we searched five times as many queries, there was roughly a five times increase in cumulative query time. Or if we doubled the number of queries, the query time increased respectively. Additionally, once an eviction operation is performed, there is a severe increase in back to back query time due to the time it takes to perform an eviction.

Figure 4 Average Random Access Time (ms)



In Experiment 2, we wanted to see how long it takes for various sizes of randomized queries. It appears that without an eviction operation, if you double or quintuple the number of queries, the time of retrieval would increase correspondingly as expected. However, this time immediately spikes if the operation requires an eviction. The eviction operation appears to take a consistent amount of time per eviction, but will cause a massive jump in back to back query time. In a worst-case scenario, where two random queries are performed with one eviction in between, a long delay will occur. This is due to the requirement for an eviction to occur after a certain number of queries. This could be optimized if the randomized queries are in consistent batch sizes, as you could configure the eviction rate according to the batch size.

IV. Conclusion

ORAM algorithms are designed to obfuscate and hide the access patterns for data sets in a system [1, 3, 4]. There are more simplistic implementations that only encrypt the data, however, advanced versions utilize secrete sharing to provide computational security [1, 3, 4]. One of these advanced algorithms, S3ORAM, is relatively new and has mostly been evaluated against theoretical performance with arbitrary data [3, 4]. Because of this, we utilized the algorithm against four columns of a Line Item TPC-H table between two systems to see how it performs with this real world dataset. After our testing, we found that S3ORAM is not efficient with its data block usage with seventy-five percent of the data being useless, however, can be efficient on the clients end systems memory usage. In addition, we noticed that evicition operations can cause major batch query retrieval times if they are not optimized with the eviction rate.

There is also the question as to whether or not we think you could use S3ORAM in practice. Based on what we have seen, we don't think S3ORAM is viable outside of very niche situations. If your application is fine with using four times as much drive storage compared to the original datasets, and would not have frequent back to back queries, S3ORAM could be used. However, as many databases tend to have back to back queries, this would not be very viable as a data structure for a database.

Overall, we see the continued research into developing secure ORAM algorithms very beneficial for the future. In our view, if secure ORAM algorithms such as S3ORAM become more storage efficient and greatly reduce the time or remove the need for evictions, ORAM algorithms could become a major selling point of future cloud infrastructure. If these issues are resolved, S3ORAM or other ORAM algorithms could potentially be implemented in the virtual memory of virtual machines in a hypervisor based environment. With that, major cloud providers could prove with the algorithm that using their services, that the only way to know what is going on with the memory in the machine is to actually reverse engineer the source code of what's running on the virtual machine instead of looking at the access patterns from the host system.

Acknowledgment

Special thanks to Dr. Sharma for his generous help in overseeing the research and aiding in final reviews. Also special thanks to Wesley Fagan for aiding in the review of this paper.

References

[1] E. Stefanov, M. V. Dijk, E. S. Cornell, T.-H. H. Chan, C. Fletcher, L. Ren , X. Yu, and S. Devadas, "Path Oram: An extremely simple oblivious ram protocol: Journal of the ACM: Vol 65, no 4," Journal of the ACM, 01-Aug-2018. [Online]. Available: https://dl.acm.org/doi/10.1145/3177872. [Accessed: 30-Jan-2023].

[2] G. J. Simmons, "Secret-sharing," Encyclopædia Britannica, 26-Jul-1999. [Online]. Available: https://www.britannica.com/topic/cryptology/Secret-sharing. [Accessed: 30-Jan-2023].

[3] T. Hoang, A. A. Yavuz, and J. Guajardo, "A Multi-server Oram Framework with constant client bandwidth blowup," ACM Transactions on Privacy and Security, 01-Feb-2020. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/3369108. [Accessed: 30-Jan-2023].

[4] T. Hoang, C. D. Ozkaptan, A. A. Yavuz, J. Guajardo, and T. Nguyen, "S3oram: A computation-efficient and constant client bandwidth blowup Oram with Shamir Secret Sharing," Cryptology ePrint Archive, 01-Jan-1970. [Online]. Available: https://eprint.iacr.org/2017/819. [Accessed: 30-Jan-2023].