Protection of Access Pattern

Biao Gao
gaobiao@iie.ac.cn
State Key Laboratory of Information Security, Institute of Information Engineering, CAS
School of Cyber Security, University of Chinese Academy of Sciences
Beijing, China

Shijie Jia*
jiashijie@iie.ac.cn
State Key Laboratory of Information Security, Institute of Information Engineering, CAS
School of Cyber Security, University of Chinese Academy of Sciences
Beijing, China

Peng Yin
yinpeng@iie.ac.cn
Defence Industry Secrecy Examination and Certification Center
School of Cyber Security, University of Chinese Academy of Sciences
Beijing, China

Xueying Zhang
zhangxueying@cics-cert.org.cn
China Industrial Control Systems Cyber Emergency Response Team
Beijing, China

ABSTRACT

Encryption is insufficient to ensure the system security because the access patterns of user will still reflect the information about the data and serve as an indicator for adversary to infer the sensitive information. We conclude the related work on the access pattern protection to provide a thorough literature review of history independence and oblivious random access machine, which address the issues of static and dynamic access pattern observations respectively.

CCS CONCEPTS

• **Computer systems organization** → **Client-server architectures**; • **Networks** → **Security protocols**; • **Information systems** → **Data access methods**; **Information storage technologies**; • **Security and privacy** → **Security protocols**.

KEYWORDS

History independence, oblivious random access machine, secure deletion, access pattern protection

ACM Reference Format:
Biao Gao, Shijie Jia, Peng Yin, and Xueying Zhang. 2023. Protection of Access Pattern. In *2023 7th International Conference on Computer Science and Artificial Intelligence (CSAI) (CSAI 2023), December 08–10, 2023, Beijing, China.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3638584.3638585

1 INTRODUCTION

In modern networks, the amount of information is increasing exponentially, and people rely on various systems to store data. However, the data usually face the challenges of being audited timely (e.g., data protection regulations like HIPAA [10] and GDPR [40]) or being observed (e.g., cloud disk storage like Dropbox) by the system. HIPAA [10], for example, requires data to be disposed of in a timely manner once it has become obsolete. People strive to explore different methods like secure deletion [19], encryption algorithms to prevent the data from disclosure. Unfortunately, the methods are insufficient to provide full-fledged protection because the access patterns of users will still expose data to disclosure risk. For example, Islam et al. [22] analyzed the access patterns to successfully identify around 80% of keyword queries to an encrypted database.

The access pattern can be considered as a collection of attributes within user access sequences, encompassing factors such as access time points, access types, access locations, access precedence order, and other pertinent elements. These attributes contain significant information concerning user data. When the adversary possesses the privilege to observe the system, these attributes can serve as a direct indicator of program structures, data interdependencies, and in the worst-case scenario, enable complete restoration of deleted data or retrieval of plaintext from encrypted data [8, 22].

There are two categories of these observation behaviors: the *static* and *dynamic* observation. The *static* observation means that the adversary possesses the ability to observe the system at certain checkpoints and obtains the corresponding representations of the memory. The *dynamic* observation means that the adversary possesses the ability to observe the system during certain or entire procedures, throughout which the adversary obtains physical access behaviors.

To provide access pattern protection targeting at the *static* observation, traditional methods such as overwriting [5, 14, 19, 23, 45] are not sufficient because the user access patterns will leave artifacts in all the layout of the system, and the system states at various time points can be regarded as dependent values obtained by historical operations with data as the independent variable. This may reveal the existence of deleted data, the order of operations, etc. Therefore, the history independence (HI) was proposed to prevent information disclosure of historical operation sequences inferred from the current state of the data structures [20, 29, 31]. There are two lines

*This author is the corresponding author.

CSAI 2023, December 08–10, 2023, Beijing, China

ACM ISBN 979-8-4007-0868-8/23/12.
https://doi.org/10.1145/3638584.3638585

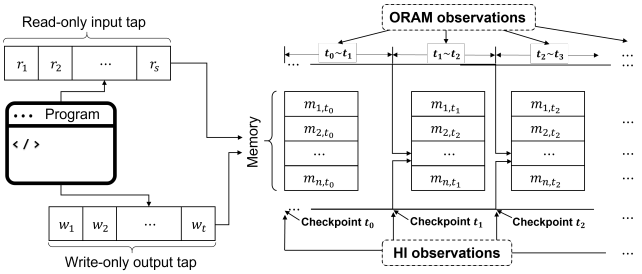

Figure 1: Model of Access Pattern Protection on a Random Access Machine. The program performs random accesses to the memory according to the read-only and write-only taps. The memory shows different states at corresponding time points. The HI focuses on the static observations of memory representation at time points while the ORAM focuses on the dynamic observations of memory operations in process.

of HI researches: designing history independent data structures (HIDS) [7, 16, 17, 20, 30, 31] and applying the history independence to protect the privacy of voters [31], the file system [4, 13], the database [3], etc., without disclosing the order in which the data are organized.

To provide access pattern protection targeting at the *dynamic* observation, oblivious random access machine (ORAM) was proposed to hide the *logical* access behaviors since the adversary can directly observe the *physical* access behaviors. In this scenario, although the transmitted data can be encrypted with encryption algorithms, the adversary may perform reverse-engineering attacks on these dynamic observations to derive the interdependencies of sensitive data and structures. To address this issue, ORAM provides a transparent memory access approach to an adversary. Similar to that of HI, the researches on ORAM also consist of two lines of work: one line is designing delicate ORAM structures and analyzing the theoretical overhead [2, 24, 41], and another line is employing ORAMs to various applications in practice such as in the case of multiple clients or servers. In the progress of ORAM research, people have applied ORAM into oblivious storage [34, 35, 43], oblivious file system [1, 46], secure multi-party computation [42, 47], secure processor [25, 27, 44], etc.

The key insights of HI and ORAM are both primarily to introduce extra (dummy) operations to confuse the potential adversary about which access behaviors happened or which data the user logically tends to access. Besides, we need to emphasize that while HI focuses on protection against static observations, the access pattern protection in this scenario is pervasive throughout the entire lifecycle of the system.

In this work, we present a thorough literature review of the access pattern protection. According to the types of adversary observations, we divide the access pattern protection into two scenarios: 1) the history independence against *static* observations; 2) the oblivious random access machine against *dynamic* observations. For both of these two scenarios, we summarize the pertinent theoretical and applied researches.

2 BACKGROUND

2.1 The Security Definition of HI

History independence focuses on protecting user's historical access patterns which lead to the current abstract state of system [20]. There are two types of definition in terms of the security of history independence, namely strong history independence (SHI) and weak history independence (WHI), which correspond to two kinds of adversary capabilities.

Definition 2.1. (**Strong History Independence** [31]).

Let S_1, S_2 be operation sequences with $P_1 = \{i_1^1, i_2^1, \cdots, i_l^1\}$ and $P_2 = \{i_1^2, i_2^2, \cdots, i_l^2\}$ as two lists of corresponding checkpoints respectively, such that $\forall b = 1, 2, \forall j \in \{1, \cdots, l\}$, we have that $1 \leq i_j^b \leq |S_b|$ and the data structures of i_j^b yield the same content. A data structure implementation is *strongly history independent* if after performing any such operation sequences, the memory representations at checkpoint i_j^1 and i_j^2 are always identical.

In other words, a system with SHI enables adversaries to have multiple snapshots of the system without compromising security whereas a system with WHI only allows adversaries to perform a single observation. Both SHI and WHI ensure that no information is disclosed beyond what is necessarily exposed by the content of the observed data structures.

2.2 Gale-Shapley Stable Marriage Algorithm

Gale and Shapley [12] proposed an algorithm G-S SMA which was well-known and widely used in many subsequent works [4, 7, 13, 16] to design structures and systems satisfying the security definition of HI. The G-S SMA was to solve the problem of guaranteeing a stable matching between two equal-sized sets of individuals. More specifically, there assumes to be two sets $M = \{m_i | 1 \leq i \leq n\}$ for men and $W = \{w_i | 1 \leq i \leq n\}$ for women. Each man/woman has his/her own preference order for all the elements in the other set. The SMP aims to find out a stable matching result $\{(m, w)_n\}$, such that $\forall i, j \in \{1, \cdots, n\}$, for any two matched pairs (m_i, w_i) and (m_j, w_j), w_i indeed prefers m_i than m_j whereas w_j indeed prefers m_j than m_i. If not, they called it an *unstable* match.

The algorithm works in a number of iterations as follows: Firstly, each man in M makes a proposal to their most preferred woman according to their preference orders. Secondly, each woman considers the proposals she has received and tentatively accepts the proposal from her most preferred man among those who have proposed to her so far. If a woman receives a proposal from a man while already provisionally engaged with another man, she compares the two proposals based on her preference order and retains the proposal from the more preferred man. Thirdly, the rejected men update their preference orders, removing the woman who rejected them from their lists, and propose to their next most preferred woman who has not yet rejected them. Fourthly, repeat the second and third step until each woman is tentatively engaged to a single man. This ensures that each man has proposed to every woman in his preference order. Finally, the SMA terminates when all men are engaged with a woman, resulting in a stable matching.

Gale and Shapley demonstrated the existence of a stable marriage arrangement for an equal number of men and women. Their algorithm ensures that the resulting pairs are always well-matched

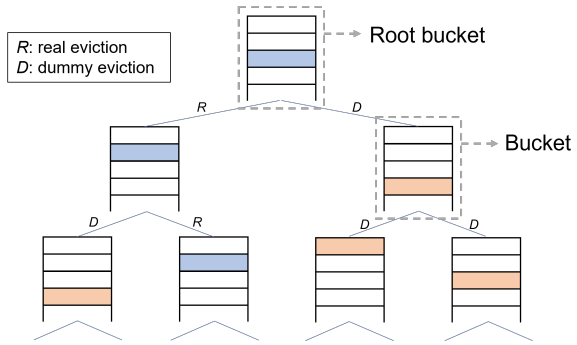

Figure 2: A Basic Binary-Tree Structure of ORAM.

and maintain stability. It is important to note that the algorithm follows the principle of suitor-optimality, ensuring that each man's partner in the final matching will not be ranked lower than any other potential partner in his preference list under alternative rules.

2.3 The Security Definition of ORAM

Oblivious random access machine focuses on protecting user's ongoing access pattern. In other words, the adversary cannot infer the true logical access sequences of users from the physical ones in progress. We present the definition in terms of the security of ORAM as follow.

Definition 2.2. (**Secure Oblivious Random Access Machine**). Let $\vec{y} := ((op_1, u_1, data_1), \cdots, (op_M, u_M, data_M))$ represent a data request sequence of length M, where for all $i \in \{1, \cdots, M\}$, the op_i denotes a $read(u_i)$ or $write(u_i)$ operation, the u_i denotes the identifier of the block to be read from or written to, and $data_i$ is the corresponding data. Let $A(\vec{y})$ represents the access sequence performed on the storage device given the data request sequence $\vec{y}$. The ORAM construction is considered to be *secure* if for any two equal-length data access request sequences $\vec{y}$ and $\vec{z}$, the advantage of distinguishing the access patterns $A(\vec{y})$ and $A(\vec{z})$ is computationally negligible to anyone except for the data owners.

2.4 Tree-based Structure of ORAM

The seminal work of ORAM proposed two classical constructions: square-root ORAM and hierarchical ORAM [15]. Subsequently, Shi et al. were inspired by the hierarchical ORAM to design a tree-based ORAM [36], which became the fundamental component of most existing ORAMs. We present a basic binary-tree structure of ORAM in Figure 2. Each node in the tree is called a bucket consisting of a fixed number of blocks. A basic idea behind hiding the logical accesses is to make the traditional read and write operations share the indistinguishable patterns, e.g., in Figure 2, despite of whether the user tends to read or write a block, the user always have to follow a predetermined set of rules to request for a serial of data blocks to hide which block is the target, and sequentially the user reshuffles and writes all the updated (like re-encrypted) blocks back to the tree. Conventionally, the data is firstly written to the root bucket and then evicted along the paths from the root node to the leaf nodes according to the ORAM scheme. The dummy and real evictions happen synchronously to eliminate the specificity of the data blocks to be evicted. However, we should note that most of the existing ORAM schemes suffer from larger overhead caused by these time-consuming operations like reshuffling, eviction, and some complicated cryptographic primitives.

2.5 Models and assumptions

In this subsection, we specify the adversary model and assumptions for access pattern protection in the following.

- We ignore other kinds of potential risks except for the access pattern disclosure. In other words, the adversary has a negligible advantage of using other methods to compromise the secret information (e.g., the decryption key/passwords) in both HI and ORAM schemes.
- In the case of HI, the adversary has a privilege to check the entire layout of the system storage at all layers to capture the snapshots at fixed checkpoints. This assumption applies to many real-world scenarios such as the data protection regulations like HIPAA [10].
- In the case of ORAM, the adversary has a privilege to continuously observe the system to capture all the access behaviors from users to system. This assumption applies to many real-world scenarios such as the cloud disk storage like Dropbox.

3 HISTORY INDEPENDENCE

In this section, we will present a literature review of history independence from two perspectives: the researches on the abstract data structures and applications of history independence.

3.1 History Independent Data Structure

Micciancio [29] firstly proposed a weak notion of history independence and abstracted the problem of designing data structures with property of obliviousness. Naor et al. [31] improved this notion into two significantly important types of definition focusing on the dictionaries: the strong history independence and the weak history independence. For the first, the strong HI denotes a data structure that allows multiple snapshots while only supporting insertions and queries from a set. In this case, the operations of insertion and queries both had amortized cost of $O(1)$ while the space utilization required $O(\log n)$. For the second, the weak HI denotes a data structure that allows only a one-time snapshot while supporting deletions besides the insertions and queries from a set. In this case, the amortized cost of insertion and deletion were both $O(1)$ while the space utilization war liner in the number of the elements inserted in the hash scheme in their work. Additionally, their work only performed well for the fixed-size records and left an open problem whether it is possible to implement a low-overhead scheme for a variable-size record.

In order to further reduce the cost and compensate for the lack of support for deletion in the SHI scheme of [31], Blelloch et al. [7] utilized the G-S SMA to construct a strongly history independent hash table, supporting queries with $O(1)$ overhead in the worst case and insertions and deletions with $O(1)$ overhead using $O(n)$ storage overhead. However, the expected cost for all operations used in their hash table implementation would increase exponentially with the growth of rate of space utilization, i.e., it yielded $O(1/(1-\alpha)^3)$ expected time where the hash table had p slots to store $n = \alpha p$

keys. Although they proposed another variant of scheme, its rate of space utilization was only 9%.

To improve the space utilization, Naor et al. [30] introduced a dynamic dictionary data structure with HI property by utilizing the cuckoo hashing method [32]. Although the cuckoo hashing does not possess the property of HI, they ensured each set of elements in the hash table had only one unique representation up to the process of initial randomness. And their data structure enabled insertions and deletions in worst-case logarithmic time complexity. When considering only insertions and membership lookup, the space utilization is approximately 50%. However, upon incorporating deletion operations, this utilization decreases further to 25%.

Besides the focus on the space utilization, Goodrich et al. [16] paid attention to the collision-time attacks which the [7] cannot defend against. The collision-time attacks mean that the adversaries exploit the timing characteristics of different operations to infer collision situations within a hash table and gain access to certain private information. Goodrich et al. relaxed the restricted requirement of SHI in [7] and leveraged the classic linear probing collision-handling scheme to design a hash table to withstand collision-time attacks. Specifically, their work calculated a probability for each element based on its current position, representing the likelihood of the element being evicted to the next position when a data collision occurred. Using this method, their solution transformed the temporal differences caused by hash collisions into independent distributions related to probabilities, thereby achieving history independence. Their scheme achieved twice the efficiency for insertions and deletions compared to [7] when the system was under heavy load. However, their work only ensured the property of WHI.

3.2 History Independent Application

Compared to theoretical researches, there are only a few studies dedicated to exploring the application of history independence, because current system designs often contradict the requirements of HI. For example, the data structures and disk distributions, are inherently geared towards data recovery to some extent. Additionally, various concerns, including trade-offs, disk search time, and the latency, need to be taken into account.

Bajaj and Sion embedded HIDS into the file system layer and constructed the first file system with SHI property, called HIFS [4]. They determined the data location based on document attributes, making the distribution of block data on the disk independent of any past operations. The document paths and read/write offsets serve as the computation basis for keys in the hash table, and the disk buckets map entries are treated as entries in the hash table. By leveraging the G-S SMA to establish the preference lists between keys and buckets in the hash table, the data distribution exhibits history independence. Compared to the non-HI EXT3 file system, [4] showed a decrease in sequential read efficiency ranging from 0.5X to 0.7X at a 60% load, and a decrease in random read efficiency ranging from 0.5X to 0.7X at a 90% load. However, the write efficiency significantly decreased when the loads were above 60%.

Gao et al. [13] proposed a more practical scheme, called eHIFS, to address the high cost of HIFS [4] in the case of large load factors. They relied on a significant observation that HIFS simply re-organizes the entire data in a history independence manner and many of those operations are unnecessary in the case of SHI. Because the adversaries have had multiple snapshots and already obtained the location information of the observed data at previous checkpoints. Therefore, eHIFS took advantage of knowledge on the adversaries' observations and fixed the observed data in corresponding locations at a determined time interval to eliminate those time-consuming and unnecessary operations. Their scheme claimed to have a 33X write throughput improvement when the file system load factor is 90%.

Besides file system, Bajaj et al. also applied HI into the database design. They introduced a relational database supporting untraceable deletion, called Ficklebase [3], by identifying residual information from deleted data and constructing a HIDS to hide historical operations. The key idea is to proactively maintain future versions of the database that excludes the to-be-deleted data. The Ficklebase would uniformly perform the deletion of data tuples at certain time intervals. However, their scheme has certain drawbacks. Firstly, it excludes tamper-resistant information such as audit logs considering that the logs are designed to be unreadable by adversaries. Secondly, maintaining a large number of future versions incurs significant overhead and reduced the performance. Thirdly, their approach requires integrating all application logic into the queries of the database, thus not supporting user-defined trigger mechanisms.

These limitations of HI application need to be carefully considered and balanced in practical applications. While they may be beneficial in terms of protecting data privacy and achieving history independence, it may not be suitable for scenarios that require robust auditing capabilities, flexible trigger mechanisms, or are sensitive to performance.

4 OBLIVIOUS RANDOM ACCESS MACHINE

The oblivious random access machine originated to prevent the memory from reverse engineering attacks by adversaries. The models of researches on ORAM have been promoted to the client-server model, where the client stores data on the remote untrusted server. However, the large overheads of the communication, computation and storage are the main obstacles for applications, because ORAM usually employs complicated cryptographic primitives, randomness and obfuscated access patterns within memory to ensure access pattern security.

4.1 Traditional Client-Server ORAM

Goldreich et al. proposed the pioneering work of ORAM with two classic constructions: the square-root ORAM and the hierarchical ORAM [15]. The square-root ORAM is a single-layer linear structure and introduces a buffer to temporarily store accessed data blocks. The idea behind it is to perform data read and update operations regardless of whether the target data is retrieved from the buffer or the storage space. By accessing data from the storage space even when the buffer is used, it helps to obfuscate the actual data access patterns and maintain the privacy of the user's access

behaviors. The hierarchical ORAM is a two-dimensional extension of the square-root ORAM, where the data structure is expanded into multiple layers, each equipped with a unique buffer. Furthermore, they established a mechanism for merging data across layers to ensure that access patterns remain undisclosed during data transfers between layers.

Based on [15], Shi et al. organized data to be stored in a binary tree, called Tree ORAM [36]. They took the overhead in the worst case into consideration, which previous works [15, 18, 33] mostly did not focused on. Shi et al. thought the commonly used operations like reshuffling led the overhead to $O(N)$. Corresponding description about Tree ORAM has been presented in Section 2.4. Their scheme achieved $O(\log^3 N)$ amortized and worst-case overhead using trivial bucket ORAM as the component while achieving $\tilde{O}((\log N)^{2.5})$ amortized overhead and $\tilde{O}((\log N)^3)$ worst-case overhead using square-root ORAM as the component, where $\tilde{O}$ denotes $poly \log \log$.

To simplify the Tree ORAM to be more practical, Stefanov et al. proposed Path ORAM [39]. It eliminated or optimized many complicated operations such as oblivious sorting, time-consuming eviction strategy, etc., reducing a low computation overhead. When accessing a block, the system would retrieve the entire path, where the target resided, to the local stash and then assigned new random paths for the written back blocks. The system would select a bucket on the corresponding path from the root node to the common node of the block's original read-out path and the new randomly chosen write-back path. The block was written into such a bucket in a greedy algorithm, i.e., as close to its leaf node as possible. This greedy algorithm saved eviction operations layer by layer. Besides, by leveraging the recursive maps locally, the Path ORAM reduced the client storage overhead from $N/\chi + O(\log N) \cdot \omega(1)$ to $O(\log^2 N/\log \chi) \cdot \omega(1)$. However, the bandwidth cost increased from traditional $O(\log N)$ to $O(\log^2 N/\log \chi)$, where the block size $B = \chi \cdot \log N$. As a representative ORAM scheme for small client storage, Path ORAM has gained widespread application and research interest due to its relatively simple construction and implementation.

To reduce the bandwidth cost, Stefanov et al. established a novel mechanism, namely Partition ORAM (or SSS ORAM) [38], where they divided a bigger ORAM into smaller ORAMs and blocks were evicted into a randomly assigned server partitions in the background to achieve a more practical performance. The motivation behind the Partition ORAM was that the primary source of overhead in previous ORAM works were the costly *remote oblivious sorting* protocol executed between the client and the server, which could take up to $O(N)$ time. Each partition would be a subsystem of the entire ORAM storage and could perform data transmission, data reshuffling concurrently. Compared to previous works, the client in Partition ORAM maintained a much smaller map from the data blocks to their partitions. Besides the reduced storage overhead, each access required retrieving the data blocks from the partition where the target data resided, rather than transmitting the entire dataset. This significantly reduced the communication overhead. However, we should note that the Partition ORAM is better suited for scenarios involving large-storage clients while worse for small-storage ones.

The Path ORAM and Partition ORAM both provide an important reference. They become the foundation for subsequent ORAM research endeavors and inspire the following works on multi-client and multi-server ORAMs, whose focuses are lying on more practical scenarios. It is worth mentioning that the state-of-the-art work on the theoretical analysis of ORAM overhead are proposed by Larsen et al. [24] in CRYPTO '18 and Asharov et al. [2] in J.ACM'23. They prove that the $\Omega(\log N)$ overhead in memory accesses is theoretically indispensable for any online ORAM to ensure computational security, where N is the number of data blocks.

4.2 Multi-client or Multi-Server ORAM

Recent researchers have focused more on the multi-party ORAM as the increase of need for more clients and servers in the current network. However, when applying ORAM into multi-client or multi-server scenarios, many new challenges will arise. For example, the mutual communication between these clients/servers may leak sensitive information about the access patterns like side channel attacks. Moreover, the multiple parities involved in the ORAM scheme will amplify the overhead of storage, communication and computation naturally.

Maffei et al. proposed PIR-MCORAM [28] to achieve a maliciously secure multi-client ORAM. They not only proved the server-side computational lower bound to be $\Omega(N)$, but also leveraged the technique of Private Information Retrieval (PIR) [9], a new accumulation technique, and an oblivious gossiping protocol to design an $O(\sqrt{N})$ communication overhead multi-client ORAM. Their scheme also incorporated the public-key cryptography and zero-knowledge proofs for access control mechanism. However, these techniques are significantly time-consuming and the $O(\sqrt{N})$ is impractical for real-world scenarios. Similar to PIR-MCORAM, Blass et al. [6] also presented a multi-client ORAM against malicious adversaries. Their idea was to divide client accesses into two distinct parts, allowing for more thorough examination to uncover any malicious behaviors. Additionally, they utilized a classical ORAM as a foundational component to store the meta data of Path ORAM, which was embedded in their scheme. However, the complicated shuffling in [6] still resulted in the large $O(N \log^2 N)$ computation overhead.

Cetin et al. proposed TaoStore [35], which firstly initiated the formal study of asynchronicity in ORAM system. They pointed out several issues that the multi-client ORAM may be faced with. For example, multiple paths of blocks may be requested for simultaneously or the data need to be updated while they are just written back, all of which may cause the problem of synchronization. The idea of [35] is to maintain a data structure, called request map, to keep track of all concurrent requests for the same block, i.e., the system will only react correctly to the first request for a block and other subsequent requests for the same block will trigger fake reads for a random path. Besides, TaoStore introduced a local invariant, *fresh-subtree*, to synchronize the data retrieved from the server to address the data conflict. Although TaoStore achieved $O(\log N)$ computation and communication costs, it only considered the semi-honest model and relied on a trusted third-party proxy to construct the system.

To further reduce the overhead of asynchronous multi-client ORAM, Cheng et al. presented *Tianji*, which extended the original S^3ORAM [21] to a new S^3ORAM$^+$. By removing the complicated operation such as the eviction, *Tianji* relied on a trusted third-party proxy to achieve the $O(\log N)$ computation overhead. However, we should note that the trusted third-party proxy is a stringent requirement which is impractical in most real-world scenarios.

Based on the traditional Partition ORAM, Stefanov et al. proposed MCOS [37], which leveraged the powerful computational capability among the non-colluding clouds to move the reshuffling operations from the client side to the server side. The idea of hiding the access pattern in MCOS is to separate the observations of different clouds, i.e., the non-requested servers, compared to the requested server, have different knowledge on the access behaviors. Specifically, the requested blocks will be shuffled in the selected cloud and sent to other clouds, then these clouds will response the client with these requested blocks. This kind of permutation ensures the security of access pattern in the model of non-colluding clouds. Although MCOS achieved $O(1)$ client-server communication overhead, the operations of onion encryption and shuffling still consumed large and the cloud-cloud communication resulted in the low response performance.

To further take advantage of the non-colluding clouds model to design ORAMs, Liu et al. proposed a NewMCOS [26], which further divided the partitions in MCOS [37] into multiple clouds. As a result, a original request could be separated into different smaller access sequences targeting at corresponding clouds. Therefore, in such a non-colluding model, there is no cloud having all the knowledge of the entire request from the client and this ensures the obliviousness of access pattern. Besides, NewMCOS utilized a small *evict cache* to perform the eviction in a burst way, whose idea was from the BurstORAM [11], to reduce the communication and computation overhead. Although NewMCOS finally achieved $O(1)$ client-server communication overhead, the *two-layer onion encryption* used in their scheme still consumed large cost.

5 CONCLUSION

The access pattern leakage provides adversaries with more possibilities to compromise the system confidentiality besides directly attacking on encryption algorithms. In this work, we present a thorough literature review of the access pattern protection from two perspectives of the observations: the researches on history independence for static observations and the researches on oblivious random access machine for dynamic observations. We hope that this work can be further exploited and become a reference for the subsequent works on access pattern protection to achieve more better designs.

ACKNOWLEDGMENTS

This work was supported by National Natural Science Foundation of China (No.62272457) and Defense Industrial Technology Development Program (Grant JCKY2021906B002).

REFERENCES

[1] Adil Ahmad, Kyungtae Kim, Muhammad Ihsanulhaq Sarfaraz, and Byoungyoung Lee. 2018. OBLIVIATE: A Data Oblivious Filesystem for Intel SGX. In *NDSS*.

[2] Gilad Asharov, Ilan Komargodski, Wei-Kai Lin, Kartik Nayak, Enoch Peserico, and Elaine Shi. 2022. Optorama: Optimal oblivious ram. *J. ACM* 70, 1 (2022), 1–70.

[3] Sumeet Bajaj and Radu Sion. 2013. Ficklebase: Looking into the future to erase the past. In *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*. IEEE, 86–97.

[4] Sumeet Bajaj and Radu Sion. 2013. HIFS: History independence for file systems. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 1285–1296.

[5] Steven Bauer and Nissanka B Priyantha. 2001. Secure data deletion for Linux file systems. In *Proceedings of the 10th conference on USENIX Security Symposium-Volume 10*. USENIX Association, 12.

[6] Erik-Oliver Blass, Travis Mayberry, and Guevara Noubir. 2017. Multi-client oblivious ram secure against malicious servers. In *International Conference on Applied Cryptography and Network Security*. Springer, Cham, 686–707.

[7] Guy E Blelloch and Daniel Golovin. 2007. Strongly history-independent hashing with applications. In *Foundations of Computer Science, 2007. FOCS'07. 48th Annual IEEE Symposium on*. IEEE, 272–282.

[8] Bo Chen, Shijie Jia, Luning Xia, and Peng Liu. 2016. Sanitizing data is not enough!: towards sanitizing structural artifacts in flash media. In *Proceedings of the 32nd Annual Conference on Computer Security Applications*. ACM, 496–507.

[9] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. 1998. Private information retrieval. *Journal of the ACM (JACM)* 45, 6 (1998), 965–981.

[10] United States Congress. 1996. Health Insurance Portability and Accountability Act. https://www.hhs.gov/hipaa/index.html

[11] Jonathan Dautrich, Emil Stefanov, and Elaine Shi. 2014. Burst ORAM: Minimizing ORAM Response Times for Bursty Access Patterns. In *23rd USENIX Security Symposium (USENIX Security 14)*. USENIX Association, Berkley, CA, 749–764.

[12] David Gale and Lloyd S Shapley. 1962. College admissions and the stability of marriage. *The American Mathematical Monthly* 69, 1 (1962), 9–15.

[13] Biao Gao, Bo Chen, Shijie Jia, and Luning Xia. 2019. eHIFS: An Efficient History Independent File System. In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*. 573–585.

[14] Simson L Garfinkel and Abhi Shelat. 2003. Remembrance of data passed: A study of disk sanitization practices. *IEEE Security & Privacy* 99, 1 (2003), 17–27.

[15] Oded Goldreich and Rafail Ostrovsky. 1996. Software protection and simulation on oblivious RAMs. *Journal of the ACM (JACM)* 43, 3 (1996), 431–473.

[16] Michael T Goodrich, Evgenios M Kornaropoulos, Michael Mitzenmacher, and Roberto Tamassia. 2016. More practical and secure history-independent hash tables. In *European symposium on Research in Computer Security*. Springer, 20–38.

[17] Michael T Goodrich, Evgenios M Kornaropoulos, Michael Mitzenmacher, and Roberto Tamassia. 2017. Auditable data structures. In *Security and Privacy (EuroS&P), 2017 IEEE European Symposium on*. IEEE, 285–300.

[18] Michael T. Goodrich and Michael Mitzenmacher. 2010. *MapReduce Parallel Cuckoo Hashing and Oblivious RAM Simulations*. Technical Report. arXiv:1007.1259 http://arxiv.org/abs/1007.1259

[19] Peter Gutmann. 1996. Secure deletion of data from magnetic and solid-state memory. In *Proceedings of the Sixth USENIX Security Symposium, San Jose, CA*, Vol. 14. 77–89.

[20] Jason D Hartline, Edwin S Hong, Alexander E Mohr, William R Pentney, and Emily C Rocke. 2005. Characterizing history independent data structures. *Algorithmica* 42, 1 (2005), 57–74.

[21] Thang Hoang, Ceyhun D Ozkaptan, Attila A Yavuz, Jorge Guajardo, and Tam Nguyen. 2017. S3oram: A computation-efficient and constant client bandwidth blowup oram with shamir secret sharing. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 491–505.

[22] Mohammad Saiful Islam, Mehmet Kuzu, and Murat Kantarcioglu. 2012. Access pattern disclosure on searchable encryption: ramification, attack and mitigation.. In *Ndss*, Vol. 20. 12.

[23] Nikolai Joukov and Erez Zadok. 2005. Adding Secure Deletion to Your Favorite File System *(SISW '05)*. IEEE Computer Society, USA, 63–70. https://doi.org/10.1109/SISW.2005.1

[24] Kasper Green Larsen and Jesper Buus Nielsen. 2018. Yes, there is an oblivious RAM lower bound!. In *Advances in Cryptology–CRYPTO 2018: 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19–23, 2018, Proceedings, Part II*. Springer, 523–542.

[25] Chang Liu, Xiao Shaun Wang, Kartik Nayak, Yan Huang, and Elaine Shi. 2015. Oblivm: A programming framework for secure computation. In *2015 IEEE Symposium on Security and Privacy*. IEEE, 359–376.

[26] Zheli Liu, Bo Li, Yanyu Huang, Jin Li, Yang Xiang, and Witold Pedrycz. 2019. NewMCOS: Towards a practical multi-cloud oblivious storage scheme. *IEEE Transactions on Knowledge and Data Engineering* 32, 4 (2019), 714–727.

[27] Martin Maas, Eric Love, Emil Stefanov, Mohit Tiwari, Elaine Shi, Krste Asanovic, John Kubiatowicz, and Dawn Song. 2013. Phantom: Practical oblivious computation in a secure processor. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. 311–324.

[28] Matteo Maffei, Giulio Malavolta, Manuel Reinert, and Dominique Schröder. 2017. Maliciously secure multi-client ORAM. In *International Conference on Applied Cryptography and Network Security*. Springer, Cham, 645–664.

[29] Daniele Micciancio. 1997. Oblivious data structures: applications to cryptography. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*. ACM, 456–464.

[30] Moni Naor, Gil Segev, and Udi Wieder. 2008. History-independent cuckoo hashing. In *International Colloquium on Automata, Languages, and Programming*. Springer, 631–642.

[31] Moni Naor and Vanessa Teague. 2001. Anti-persistence: History independent data structures. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*. ACM, Association for Computing Machinery, New York, NY, USA, 492–501.

[32] Rasmus Pagh and Flemming Friche Rodler. 2004. Cuckoo hashing. *Journal of Algorithms* 51, 2 (2004), 122–144. https://doi.org/10.1016/j.jalgor.2003.12.002

[33] Benny Pinkas and Tzachy Reinman. 2010. Oblivious RAM revisited. In *Annual cryptology conference*. Springer, Springer, Berlin, Heidelberg, 502–519.

[34] Daniel S Roche, Adam Aviv, and Seung Geol Choi. 2016. A practical oblivious map data structure with secure deletion and history independence. In *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 178–197.

[35] Cetin Sahin, Victor Zakhary, Amr El Abbadi, Huijia Lin, and Stefano Tessaro. 2016. Taostore: Overcoming asynchronicity in oblivious data storage. In *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 198–217.

[36] Elaine Shi, T-H Hubert Chan, Emil Stefanov, and Mingfei Li. 2011. Oblivious RAM with O ((logN) 3) worst-case cost. In *International Conference on The Theory and Application of Cryptology and Information Security*. Springer, Springer, Berlin, Heidelberg, 197–214.

[37] Emil Stefanov and Elaine Shi. 2013. Multi-cloud oblivious storage. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security (CCS '13)*. Association for Computing Machinery, New York, NY, USA, 247–258.

[38] Emil Stefanov, Elaine Shi, and Dawn Xiaodong Song. 2012. Towards Practical Oblivious RAM. In *19th Annual Network and Distributed System Security Symposium, NDSS 2012, San Diego, California, USA, February 5-8, 2012*. The Internet Society, Reston, VA, USA, 1–40.

[39] Emil Stefanov, Marten van Dijk, Elaine Shi, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. 2013. Path ORAM: An Extremely Simple Oblivious RAM Protocol. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security* (Berlin, Germany) *(CCS '13)*. Association for Computing Machinery, New York, NY, USA, 299–310.

[40] European Union. 2016. REGULATION (EU) 2016/679 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL. http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=uriserv:OJ.L_.2016.119.01.0001.01.ENG&toc=OJ:L:2016:119:TOC

[41] Xiao Wang, Hubert Chan, and Elaine Shi. 2015. Circuit oram: On tightness of the goldreich-ostrovsky lower bound. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. 850–861.

[42] Xiao Shaun Wang, Yan Huang, TH Hubert Chan, Abhi Shelat, and Elaine Shi. 2014. SCORAM: oblivious RAM for secure computation. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (CCS '14)*. 191–202.

[43] Xiao Shaun Wang, Kartik Nayak, Chang Liu, TH Hubert Chan, Elaine Shi, Emil Stefanov, and Yan Huang. 2014. Oblivious data structures. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. 215–226.

[44] Yongge Wang and Qutaibah M Malluhi. 2022. Privacy Preserving Computation in Cloud Using Reusable Garbled Oblivious RAMs. In *Information Security: 25th International Conference, ISC 2022, Bali, Indonesia, December 18–22, 2022, Proceedings*. Springer, 3–19.

[45] Michael Yung Chung Wei, Laura M Grupp, Frederick E Spada, and Steven Swanson. 2011. Reliably Erasing Data from Flash-Based Solid State Drives.. In *FAST*, Vol. 11. 8–8.

[46] Peter Williams, Radu Sion, and Alin Tomescu. 2012. Privatefs: A parallel oblivious file system. In *Proceedings of the 2012 ACM conference on Computer and communications security*. 977–988.

[47] Samee Zahur, Xiao Wang, Mariana Raykova, Adrià Gascón, Jack Doerner, David Evans, and Jonathan Katz. 2016. Revisiting square-root ORAM: efficient random access in multi-party computation. In *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 218–234.