Memory-Saving Oblivious RAM for Trajectory Data via Hierarchical Generation of Dummy Access over Untrusted Cloud Environment

Taisho Sasada[1][a] and Bernard Ousmane Sane[2,3][b]

[1]*Graduate School of Science and Technology, Nara Institute Science and Technology, Ikoma, Japan*

[2]*Graduate School of Media and Governance, Keio University Shonan Fujisawa Campus, Kanagawa, Japan*

[3]*Quantum Computing Center, Keio University, Kanagawa, Japan*

Keywords: Trusted Execution Environment, Encrypted Database, Oblivious Random Access Memory, Access Pattern Leakage, Homomorphic Encryption, Trajectory Data.

Abstract: The proliferation of smartphones and IoT devices has led to a rapid increase in the generation of trajectory data. Managing this continuously generated data poses a significant burden. To alleviate this burden, cloud databases have become widespread, leading to increased storage of data on servers managed by other individuals and organizations (third parties). However, if there are adversaries among these third parties, viewing the data contents could lead to personal information leaks and privacy violations. Therefore, there are expectations for the use of encrypted databases that allow searching and managing data while it remains encrypted (in ciphertext form), without revealing the contents. Since data owners (clients) encrypt their data before storing it, third parties cannot view the actual content. However, it is known that merely encrypting the data is not sufficient for security, as a vulnerability has been identified where the original data can be inferred from access patterns to the encrypted database even without seeing the actual data content. In this paper, we propose an anonymization method for access patterns on trajectory data in encrypted databases. For anonymization, we apply Oblivious Random Access Memory (ORAM), which generates dummy accesses alongside data aggregation and updates to make the original accesses unidentifiable. Trajectory data is often aggregated and updated on a trajectory basis rather than by individual points. Therefore, directly generating dummy accesses at the point level using ORAM leads to overhead in encrypted memory. In our proposed method, we separate the data storage memory into upper and lower levels to make access patterns unidentifiable at the trajectory level rather than the point level. The lower memory contains single points, while the upper memory contains multiple points (capable of representing part or all of a trajectory), and dummy accesses are generated using ORAM to make upper memory accesses mutually unidentifiable.

1 INTRODUCTION

Trajectory data records chronological changes in location information, such as people's movement routes, vehicle operation records, and logistics tracking. This data is utilized in various fields including urban planning, traffic optimization, marketing analysis, and understanding behavioral patterns. However, since it contains large amounts of position coordinates and timestamps continuously collected from sensors like GPS, the data volume tends to become enormous, making on-premises management challenging. Cloud database services have become widespread due to benefits such as reduced server management burden, lower implementation and operational costs, and centralized data management. As a result, there has been an increase in storing data on servers managed by other individuals and organizations (third parties).

However, if there are adversaries among these third parties, viewing the data contents could lead to personal information leaks and privacy violations. Therefore, data owners (clients) have increasingly begun encrypting their data before entrusting it to third-party servers such as cloud database services. This allows data to be managed on third-party servers without revealing its contents (keeping it in encrypted form), and this is called an encrypted database. Unlike encryption through database functionality itself, the data is encrypted on the client side, thus pro-

[a] https://orcid.org/0000-0003-2144-4949
[b] https://orcid.org/0000-0002-9249-8285

Sasada, T. and Sane, B. O.
Memory-Saving Oblivious RAM for Trajectory Data via Hierarchical Generation of Dummy Access over Untrusted Cloud Environment.
DOI: 10.5220/0013370100003899
Paper published under CC license (CC BY-NC-ND 4.0)
In *Proceedings of the 11th International Conference on Information Systems Security and Privacy (ICISSP 2025) - Volume 2*, pages 635-642
ISBN: 978-989-758-735-1; ISSN: 2184-4356
Proceedings Copyright © 2025 by SCITEPRESS – Science and Technology Publications, Lda.

tecting privacy even if the database administrator has malicious intent. But multiple studies have revealed that encrypted database alone does not provide sufficient protection. A particularly significant vulnerability is information leakage through access pattern disclosure. Access patterns are chronological records of data access logs, and statistical analysis of these patterns can enable inference of confidential information about the original encrypted data. In fact, (Islam et al., 2012) demonstrated that 80% of search queries could be inferred by analyzing access patterns to encrypted email repositories, highlighting the severity of this vulnerability.

To address this vulnerability, access pattern concealment methods using Oblivious Random Access Memory (ORAM) (Goldreich and Ostrovsky, 1996) with homomorphic encryption have been proposed. ORAM is a technology that makes access patterns unidentifiable by generating dummy accesses in addition to regular accesses. The good compatibility between ORAM and homomorphic encryption primarily stems from the ability to eliminate noise in statistical measurements caused by dummy accesses. In conventional ORAM, while dummy accesses are performed to conceal access patterns, these dummy accesses would create noise in statistical measurements. By using homomorphic encryption, the content of dummy accesses can be set to encrypted zeros (or other neutral values). This allows for increasing the number of accesses to hide access patterns while simultaneously preventing any impact on statistical measurements (Moataz et al., 2015; Liu et al., 2018; Falk et al., 2023).

In this paper, we propose **M**emory-saving **ORA**M for trajectory **D**ata **O**ver Encrypted Database, MORADO. While we apply ORAM for anonymization, trajectory data is often aggregated and updated on a trajectory basis rather than by individual points. Therefore, directly generating dummy accesses at the point level using ORAM leads to overhead in the encrypted database. In our proposed method, we separate the ORAM memory that stores data into upper and lower levels to make access patterns unidentifiable at the trajectory level rather than the point level. The ORAM lower memory contains single points, while the ORAM upper memory contains multiple points (capable of representing part or all of a trajectory), and dummy accesses are generated using ORAM to make upper memory accesses mutually unidentifiable. This enables anonymization of access patterns at the trajectory level. The structure of this paper is as follows: In Section 2, we explain the basic technologies such as ORAM, TDX in this research. In Section 3, we explore the detailed design of proposed method and flow of dummy access generation. In Section 4, we present the results of the experimental evaluation. In Section 2, we outline related works on databases utilizing homomophirc encryption, TEE, and their challenges. Finally, in Section 6, we conclude the paper by summarizing the contributions of this research.

2 PRELIMINARY

2.1 Partially Homomorphic Cryptosystem

A Partially Homomorphic Cryptosystem is a special encryption scheme that allows operations (particularly addition) between ciphertexts. When the result of these operations is decrypted, it matches the result of performing the same operations on the original plaintexts. One representative additive homomorphic encryption scheme is the Paillier Homomorphic Encryption (PHE).

The key generation in PHE proceeds as follows. First, select two large prime numbers p and q, and calculate their product n and the least common multiple λ of $p-1$ and $q-1$. Next, choose an integer g from $\mathbb{Z}^*_{n^2}$ and calculate the multiplicative inverse μ of $L(g \bmod n^2)$ modulo n, where:

$$L(u) = \frac{u-1}{n} \tag{1}$$

The resulting public key is (n,g) and the private key is (λ,μ). In the encryption process, a random value r is selected from $\mathbb{Z}^*_n$, and for a plaintext m, the ciphertext is computed as:

$$c = g^m \cdot r^n \bmod n^2 \tag{2}$$

For decryption, given a ciphertext c, the plaintext is recovered by computing:

$$m = L(c^\lambda \bmod n^2) \cdot \mu \bmod n \tag{3}$$

An important point is that while both encryption and decryption use modular exponentiation, decryption has a higher computational cost. This is because the base of exponentiation during decryption is the ciphertext. The size of ciphertext is significantly larger than the public key n. An important property of PHE is that when the product of two ciphertexts is decrypted, it yields the sum of the corresponding plaintexts. That is:

$$\mathrm{Dec}(\mathrm{Enc}(m_1) \cdot \mathrm{Enc}(m_2) \bmod n^2) = m_1 + m_2 \bmod n \tag{4}$$

This property enables additive operations to be performed on encrypted data.

2.2 Trust Domain Extensions

Intel Trust Domain Extensions (TDX) is a TEE based on secure virtualization (Intel, 2023). TDX has the capability to deploy hardware isolated virtual machines called trust domains (TD). TDs are isolated from virtual machine manager (VMM)/hypervisor and any other software which are not related to TD as shown. This strong isolation provides the required security guarantees for a TEE system. Moreover, Intel TDX uses multi-key total memory encryption (MK-TME) and hashing to maintain the confidentiality and integrity of the code and data in TD. Intel TDX module is designed to run in Secure Arbitration Mode (SEAM). SEAM introduces an expansion to the Virtual Machine Extension (VMX) architecture, establishing a fresh VMX root mode known as SEAM root. This SEAM root mode serves as the platform for accommodating a CPU-attested module designed for generating VM guests termed TD. SEAM mode can be used as two logical modes: TDX non-root mode and TDX root mode. TDX root mode is used for Host side operations and non-root mode is used for TD guest operations.

2.3 Oblivious Random Access Memory

ORAM (Oblivious RAM) (Goldreich and Ostrovsky, 1996) is a primitive that obscures a user's (processor's) access patterns to storage (DRAM). ORAM transforms a user's sequence of program address accesses into access patterns that appear random. While physical access locations remain visible to an attacker, the ORAM interface guarantees that physical access patterns are independent of logical access patterns, preventing leakage of data-dependent access patterns. Additionally, it uses probabilistic encryption to protect data content and hide whether updates have occurred.

Path ORAM (Stefanov et al., 2018) is currently the most efficient and well-researched ORAM implementation. It consists of two main hardware components: binary tree storage and an ORAM controller. Each node in the binary tree can store up to Z useful data blocks, with dummy blocks filling empty slots. The ORAM controller is trusted hardware that includes a position map and stash. In Path ORAM, accessing a data block involves consulting the position map, reading and decrypting blocks along the path, remapping to a new random position, and writing blocks back from the stash. This ensures each ORAM access is random and untraceable. However, Path ORAM incurs significant energy and performance penalties compared to regular DRAM.

Write-only ORAM (WoORAM) (Li and Datta, 2017) is a lightweight version of ORAM that only obscures write access patterns. It offers better performance than full ORAM against attackers who cannot monitor read access patterns. Li and Datta's scheme was proposed for private information retrieval in data centers but is not efficient in the context of secure processors.

DetWoORAM (Deterministic Write-only ORAM) (Roche et al., 2017) is a deterministic Write-only ORAM scheme that doesn't require a stash. It generates fixed, deterministic physical write access patterns regardless of logical write access patterns. While this eliminates the need for a stash, it requires additional accesses to move data from temporary storage to persistent main memory, resulting in performance penalties. Hardware implementation complexity also remains a challenge.

3 PROPOSED METHOD

3.1 Threat Model

For the architectural design of the proposed method, we define a threat model centered on protecting data and access patterns. This model serves as the foundation for protecting sensitive data and its usage patterns in cloud environments. In the threat model, we consider the Cloud Service Provider (CSP) as the primary threat actor. We assume the CSP has extensive access rights across all software layers (OS, hypervisor, and other system components). These access rights potentially enable the CSP to observe and analyze the original data and its processing. Among the CSP's attack capabilities, we particularly focus on memory dump attacks and cold boot attacks (Yitbarek et al., 2017; Halderman et al., 2009). In memory dump attacks, the CSP can acquire and analyze system memory contents at any time, risking exposure of unencrypted data and access patterns. In cold boot attacks, original data might be extracted from residual RAM data by acquiring memory contents immediately after system power-down. Furthermore, CSP can continuously monitor and view data loaded into memory. This capability enables the CSP to track data flows and processing patterns in detail and analyze temporal correlations. This suggests the possibility of inferring not only data contents but also access patterns and DO's behavior patterns.

In addition to CSP-related threats, malware and malicious software targeting the cloud must also be considered. These can lead to the theft of sensitive information such as usernames and passwords, as well as entire databases hosted in clusters. This means that beyond data interception and tampering, there are threats of access to pattern observation and recording. Specific threats include malware such as Siloscape, which targets Windows containers in cloud environments, and Kinsing Malware, which targets Docker/Kubernetes clusters. These malware specimens can remain dormant in systems for extended periods, secretly monitoring data flows and access patterns. As a summary, the main protection targets in this model are the confidentiality of DO's data and the privacy of its access patterns. In addition to protecting data content, information about who accesses the data, when and how is treated as equally important protection targets.

3.2 Overall Architecture

The concept of the proposed method is shown in the Figure 1. The proposed method leverages the fact that, in the context of trajectory data insertion, the majority of queries are trajectory-unit rather than point-unit queries, thereby reducing the total amount of dummy accesses generated by ORAM. The architecture of the proposed method is broadly divided between inside and outside the cloud. Within the cloud, memory is divided into several upper and lower memories. Each upper memory consists of multiple smaller lower memories. DO encrypts their trajectory data using Paillier homomorphic encryption (PHE) and sends it to the cloud side through a TLS/SSL connection. Homomorphically encrypted data insertions from DOs are first aggregated into appropriate lower memories within these upper memories. Each upper memory is assigned its own ORAM client, with corresponding ORAM servers existing on the database server. On the proxy server, to conceal the distribution of requests, the number of requests processed by each upper memory is uniformized. Specifically, a value $\lambda*(u,\alpha)$ is calculated, representing the maximum number of requests in all upper memories. Each upper memory appears to process $\lambda*(u,\alpha)$ requests, even when the actual number of requests is lower. On the database server side, each ORAM server stores homomorphically encrypted data $(\sum Enc(d1),\ldots,\sum Enc(d\lambda*(u,\alpha)))$ from the corresponding upper memory. This prevents the CSP from knowing how many requests each upper memory is processing.

3.3 Flow of Dummy Access Generation

As input, the algorithm accepts a set $E = \varepsilon_1,\ldots,\varepsilon_w$ consisting of w homomorphically encrypted values.

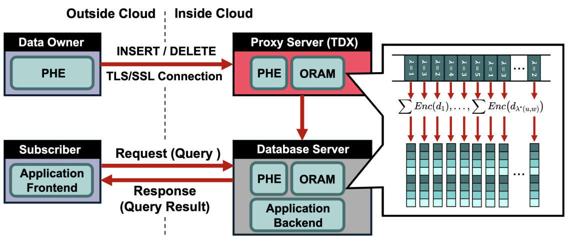

Figure 1: MORADO's System Model Architecture.

Each encrypted value ε_i is defined as $\varepsilon_i = Enc(r_i)$, where Enc is an encryption function applied to the corresponding original request r_i. Additionally, the algorithm requires a target lower memory set $M^{\text{lower}} = m_1^{\text{lower}},\ldots,m_\alpha^{\text{lower}}$ and seed values $S = \varepsilon_1^o,\varepsilon_2^o$ for dummy access generation as input parameters. The expected output is an indistinguishable assignment of these requests to the target lower memorys.

Phase 1. Access Aggregation: The first phase executes system-wide aggregation of encrypted requests. Specifically, the cumulative value $\Sigma\varepsilon$ for each lower memory within all upper memory m^{upper} is initialized to zero. Subsequently, for each lower memory m_j^{lower} $(1 \leq j \leq \alpha)$, its upper memory m_i^{upper} is identified, and the encrypted value ε_j is added to the cumulative value $\Sigma\varepsilon$ of lower memory m_j^{lower}.

Phase 2. Dummy Access Calculation: The second phase determines the number of dummy accesss required for privacy protection. For each upper memory m_i^{upper}, the number of non-zero lower memorys λ_i is calculated. Here, λ_i represents the crucial metric of how many lower memorys within m_i^{upper} contain actual requests. Subsequently, the maximum value $\lambda^* = \max_{1 \leq i \leq u} \lambda_i$ across all upper memorys is derived. This λ^* functions as a fundamental parameter defining the system-wide privacy protection level and plays a decisive role in the subsequent dummy access generation process.

Phase 3. Request Processing: The third phase implements a two-stage refined processing for each upper memory m_i^{upper}. The first stage focuses on real request processing: for each lower memory m_j^{lower} holding aggregated requests $\Sigma\varepsilon_j$, an `aggregate_request`$(\Sigma\varepsilon_j,\ m_j^{\text{lower}})$ is issued to ORAM client i. This processing enables secure transfer of real requests. The second stage handles dummy access generation and issuance, processing each index j from $\lambda_i + 1$ to λ^*. During this stage, a lower memory m^{lower} is randomly selected, and an XOR operation $\varepsilon^o = \varepsilon_1^o \oplus \varepsilon_2^o$ is applied. The resulting value is transmitted to ORAM client i via `aggregate_request`$(\varepsilon^o,\ m^{\text{lower}})$. Finally, either ε_1^o or ε_2^o in seed value S is probabilistically replaced with ε^o.

Algorithm 1: MORADO.

```
Input: Set of encrypted values
       $E = \{\varepsilon_1, ..., \varepsilon_\alpha\}$ where $\varepsilon_i = Enc(r_i)$
       Target lower memory set
       $m^{\text{lower}} = \{p_1, ..., p_\alpha\}$
       Seed values $S = \{\varepsilon_1^o, \varepsilon_2^o\}$
Output: Indistinguishable $m^{\text{lower}}$

1  Phase 1: access aggregation
2  foreach upper memory $m_i^{upper}$ do
3      foreach lower memory $m_j^{lower}$ in $m_i^{upper}$ do
4          $\Sigma\varepsilon \leftarrow 0$
5      end
6  end
7  for $j \leftarrow 1$ to $\alpha$ do
8      Identify $m_i^{\text{upper}}$ containing $m_j^{\text{lower}}$
9      $\Sigma\varepsilon_{m_j^{\text{lower}}} \leftarrow \Sigma\varepsilon_{m_j^{\text{lower}}} + \varepsilon_j$
10 end
11 Phase 2: dummy access Calculation
12 foreach upper memory $m_i^{upper}$ do
13     Calculate $\lambda_i \leftarrow$ number of non-zero lower
         memorys in $m_i^{\text{upper}}$
14 end
15 $\lambda^* \leftarrow \max_{1 \le i \le u} \lambda_i$
16 Phase 3: Request Processing
17 foreach upper memory $m_i^{upper}$ do
       // Stage 1: Process real requests
18     foreach lower memory $m_j^{lower}$ with $\Sigma\varepsilon_j \neq 0$ in $m_i^{upper}$ do
19         aggregate_request($\Sigma\varepsilon_j, m_j^{\text{lower}}$) to
             ORAM client $i$
20     end
       // Stage 2: Generate and process dummy accesss
21     for $j \leftarrow \lambda_i + 1$ to $\lambda^*$ do
22         Randomly select lower memory
             $m^{\text{lower}}$ in $m_i^{\text{upper}}$
23         $\varepsilon^o \leftarrow \varepsilon_1^o \oplus \varepsilon_2^o$
24         aggregate_request($\varepsilon^o, m^{\text{lower}}$) to
             ORAM client $i$
25         Replace either $\varepsilon_1^o$ or $\varepsilon_2^o$ with $\varepsilon^o$ in $S$
             Randomly.
26     end
27 end
```

3.4 Security Analysis

We prove the security of the MORADO in the following mathematical indistinguishability. In Morado, an access request R_{access} in the form $(id, \mathsf{Enc}(s), c)$ is submitted from DOs. Let D_{view} represent the state (view) of the database prior to R_{access}, where D_{view} consists of the aggregate values in all lower memories. The proxy server generates the updated view D'_{view} after R_{access}. Then, CSP (adversary) $\mathcal{A}$ receive both D_{view} and D'_{view} with id and $\mathsf{Enc}(s)$. Then, the proxy server gets the corresponding lower memory $m^{\text{lower}} = \Pi(c)$, and chooses another $c' \neq c$ such that $m^{\text{lower'}} = \Pi(c') \neq m^{\text{lower}}$ randomly. Both m^{lower} and $m^{\text{lower'}}$ are submitted to $\mathcal{A}$. The adversary guesses which of m^{lower} or $m^{\text{lower'}}$ corresponds to access R_{access}. The mathematical indistinguishability requires that for any probabilistic polynomial-time adversary $\mathcal{A}$ for all possible m, N, and R_{access}:

$$\Pr[\mathcal{A}(V_D, V'_D, id, \mathsf{Enc}(s), m^{\text{lower}}, m^{\text{lower'}}) = m^{\text{lower}}] \tag{5}$$

$$\le \frac{1}{2} + \varepsilon(\delta) \tag{6}$$

In this Equation, $\varepsilon(\cdot)$ is a mask mechanism and δ is the security parameter. The adversary downloads view D_{view} from the proxy server. In MORADO, D_{view} contains a sequence of ORAM accesss, and contain the DO's ID. Lower memories is in the form $(id_{\text{upper}}, id_{\text{lower}})$, where id_{upper} and id_{lower} represent the upper memory ID and lower memory ID within the upper memory. After downloading D_{view}, the adversary sends the ID to the proxy server, and the proxy server returns the correct lower memory $m^{\text{lower}} = (id_{\text{upper}}, id_{\text{lower}})$ and a false lower memory $m^{\text{lower'}} = (id'_{\text{upper}}, id'_{\text{lower}})$ for the adversary to estimate between m^{lower} and $m^{\text{lower'}}$. In the worst case, the adversary is enable to match the correct access ID j within the batch with ID, but the adversary cannot distinguish between id_{upper} and id'_{upper}. The adversary also is not able to distinguish between id_{lower} and id'_{lower}, this is based on the mathematical security provided by the ORAM. Then, ORAM determines whether the access sequences having id_{lower} and id'_{lower} are the same or different. It means, the adversary cannot distinguish between m^{lower} and $m^{\text{lower'}}$, and MORADO guarantees indistinguishability.

4 EVALUATION

This experiment use a 2016 ride record dataset obtained and processed from the NYC Taxi and Limousine Commission and synthetic data. This dataset contains approximately 2.08 million records, with each record consisting of pickup timestamps and GPS location information for taxi rides. By dividing the

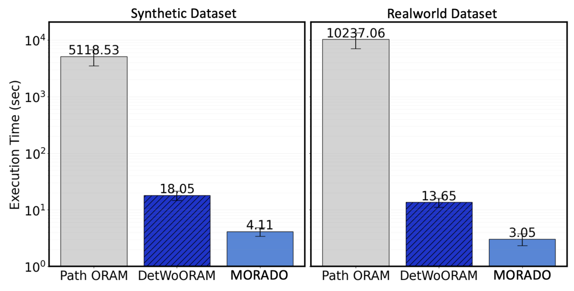

Figure 2: Execution Time.

latitude and longitude points into sufficiently small, equal-sized waypoints grids and excluding inaccessible areas, we ultimately obtained 16,367 location grids.

4.1 Execution Time

As the Figure 2 showing, the proposed method significantly faster than DetWoORAM, which is faster than Path ORAM. In the synthetic dataset, while Path ORAM takes approximately 5118.53 seconds to execute, DetWoORAM requires only 18.05 seconds, and the proposed method achieves an even shorter time of 4.11 seconds. Similar trends are observed with real-world datasets, where Path ORAM takes 10237.06 seconds, compared to DetWoORAM's 13.65 seconds and the proposed method's 3.05 seconds, showing substantial improvements. Overall, the proposed method achieves a 51.39% reduction compared to using DetWoORAM alone. This is because the DetWoORAM-only approach requires more decryption operations for each access and experiences redundant updates. What's particularly noteworthy is that in both datasets, the improvement from Path ORAM to DetWoORAM is dramatic (orders of magnitude), and the optimization from DetWoORAM to the proposed method achieves about a 4x performance improvement. This demonstrates the high practicality of the proposed method.

The left boxplot in Figure 3 evaluates the scalability of the MORADO framework with respect to the total number of M_{lower}. The graph measures the processing time for 1000 accesses while varying M_{lower} from 2^{10} to 2^{22}. The right boxplot in Figure 3 evaluates the system's scalability when varying the total of accesses α. It examines how processing time changes as the number of accesses increases from 0 to 10000, and the graph shows that the increase in processing time gradually becomes more moderate as the number of accesses grows. A notable point is that even with the current implementation, 10000 accesses can be processed within about 17.5 seconds, suggesting that

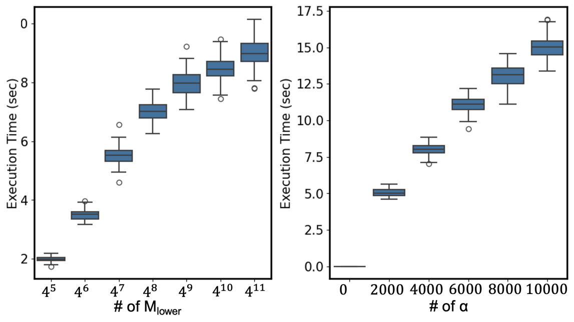

Figure 3: Overhead of access Processing.

further performance improvements can be expected through parallelization of database query processing.

4.2 Storage Cost

The analysis demonstrates both the bandwidth charges incurred between TEE and database, and the primary storage costs for each component, based on 1,000 access operations. From the graph, we can see that for TEE storage costs, Path ORAM is about 35 units and DetWoORAM is about 32 units, while MORADO shows a slight increase to about 45 units. On the other hand, for database storage costs, while Path ORAM is very high at about 4,000 units, DetWoORAM is about 1,500 units, and MORADO achieves a significant reduction to about 1,200 units. The baseline proposed method without redundant updates exceeds both DetWoORAM and Path ORAM in terms of bandwidth (included storage costs). The proposed method shows increased storage costs on TEE primarily due to dummy accesses used to hide access distribution between SPs. However, since the proposed method is based on homomorphic encryption, it saves network bandwidth during transfers. A particularly noteworthy point is that the proposed method (MORADO) achieves approximately 70% reduction in database storage costs compared to Path ORAM. This represents a significant advantage in building practical systems.

5 RELATED WORK

There are many approaches to realizing privacy-preserving data access. In this section, we explain different approaches to this problem with and without TEEs.

In threat models where database administrator (DBA) are considered adversaries, homomorphic encryption is known as a promising approach. Homomorphic encryption is a cryptographic method that

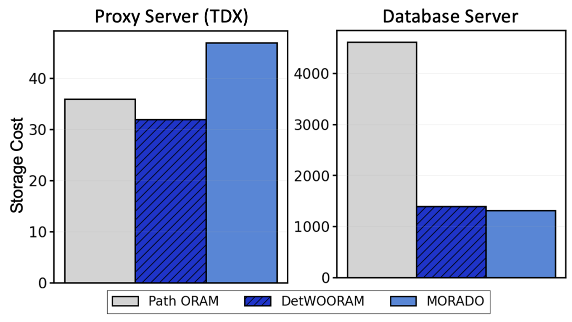

Figure 4: Storage Cost.

enables computations on encrypted data, allowing for aggregation and statistical processing without decryption. In particular, encrypted databases utilizing homomorphic encryption can respond to queries while keeping the data encrypted (Bian et al., 2023; Poddar et al., 2016; Popa et al., 2011). However, as pointed out by Isram et al. (Islam et al., 2012), encrypted databases can still leak information to DBAs through access patterns during memory operations. In encrypted database searches, the client sends encrypted search keywords to the server, and the server performs the search using these encrypted values. While the documents themselves are encrypted, the mapping of which encrypted keywords are contained in which documents must be stored on the server side to enable searching. As a result, the server can learn patterns about which documents are returned when searching with particular encrypted keywords.

In threat models where CSP are considered adversaries, TEE approaches have garnered attention. TEE is a technology that provides a trusted execution environment, with hardware-based implementations like Intel SGX being widely used. Using TEE enables isolation of programs and data running on the cloud, even from CSP. There is also substantial research on implementing databases within TEE-generated encrypted memory (enclaves) rather than using homomorphic encryption (Yang et al., 2024; Suzuki et al., 2024; Yoshimura et al., 2023; Vinayagamurthy et al., 2019). However, like homomorphic encryption-based encrypted databases, information can still leak to the CSP through access patterns during memory deployment. TEE (such as Intel TDX or AMD SEV-SNP) executes protected virtual machines or processes, but memory management (paging) must depend on the host OS. This is due to TEE physical memory capacity limitations, the need for efficient memory management, and integration with OS-level resource management. Consequently, information about which memory pages are accessed, when page faults occur, and page replacement patterns is exposed, allowing attackers to infer program execution flow, observe memory access patterns, and collect information at the cache line level. As a result, control flows like conditional branches and data-dependent memory access patterns are exposed, enabling attackers to infer program behavior and data characteristics through side-channel attacks, potentially leaking privacy information. Neither HE nor TEE methods can prevent access pattern leakage. This is why ORAM is being researched as a technique to conceal access patterns. There are various oblivious protocols, including those that make data packets unidentifiable. Notable examples include Oblivious Transfer(Rabin, 2005), which allows a receiver to obtain specific information from multiple pieces of information held by the sender without the sender knowing which information was selected. It can be used for decoupling statistic from data volume(Sasada et al., 2023; Sasada et al., 2022). Oblivious Message Routing(Kirman and Martinez, 2010), which prevents information leakage through communication pattern analysis by concealing message sources and destinations along the communication path.

6 CONCLUSION

This research paper presents a novel approach to protecting trajectory data in cloud computing environments through a memory-saving ORAM implementation. The proposed method successfully addresses the challenges of securing sensitive location data while maintaining system efficiency. By implementing a hierarchical memory structure that manages both trajectory-level and point-level access patterns, the solution achieves significant performance improvements over existing approaches. The experimental results demonstrate reduction in storage cost compared to PathORAM, while maintaining robust security measures against potential threats from CSP.

However, our security model has vulunerability in collusion attack between DOs and CSP. First, generate security parameter δ pseudonymous IDs in our system. Then, during a short time period id_{upper}, emulate access from these DOs regarding the target lower memory m^{lower}. If the victim DO id sends a access during the same period, the colluding CSP succeed in determining whether m^{lower} is the target lower memory of id's access by validating whether the total of ORAM accesses in this update round is δ or $\delta+1$. Our future work is prevention from this attack by increasing the cost of making pseudonymous IDs or by mitigating attack via increasing the number of authorized DOs.

ACKNOWLEDGEMENTS

This research was partly supported by JSPS KAKENHI Grant Numbers JP22J23910 and the Daiichi-Sankyo "Habataku" Support Program for the Next Generation of Researchers, NAIST Senju Monju Project.

REFERENCES

Bian, S., Zhang, Z., Pan, H., Mao, R., Zhao, Z., Jin, Y., and Guan, Z. (2023). He3db: An Efficient and Elastic Encrypted Database via Arithmetic-And-Logic Fully Homomorphic Encryption. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 2930–2944.

Falk, B. H., Nema, R., and Ostrovsky, R. (2023). Linear-Time 2-Party Secure Merge from Additively Homomorphic Encryption. *Journal of Computer and System Sciences*, 137:37–49.

Goldreich, O. and Ostrovsky, R. (1996). Software Protection and Simulation on Oblivious RAMs. *Journal of the ACM*, 43(3):431–473.

Halderman, J. A., Schoen, S. D., Heninger, N., Clarkson, W., Paul, W., Calandrino, J. A., Feldman, A. J., Appelbaum, J., and Felten, E. W. (2009). Lest We Remember: Cold-Boot Attacks on Encryption Keys. *Communications of the ACM*, 52(5):91–98.

Intel (2023). Intel Trust Domain Extensions (Intel TDX) Module v1.5 Base Architecture Specification. https://www.intel.com/content/www/us/en/developer/articles/technical/inteltrust-domain-extensions.html.

Islam, M. S., Kuzu, M., and Kantarcioglu, M. (2012). Access Pattern Disclosure on Searchable Encryption: Ramification, Attack and Mitigation. In *Network and Distributed System Security Symposium*, volume 20, page 12. Citeseer.

Kirman, N. and Martinez, J. F. (2010). A Power-Efficient All-Optical On-Chip Interconnect Using Wavelength-Based Oblivious Routing. *ACM Sigplan Notices*, 45(3):15–28.

Li, L. and Datta, A. (2017). Write-Only Oblivious RAM-Based Privacy-Preserved Access of Outsourced Data. *International Journal of Information Security*, 16:23–42.

Liu, Z., Huang, Y., Li, J., Cheng, X., and Shen, C. (2018). DivORAM: Towards a Practical Oblivious RAM with Variable Block Size. *Information Sciences*, 447:1–11.

Moataz, T., Mayberry, T., and Blass, E.-O. (2015). Constant Communication ORAM with Small Blocksize. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 862–873.

Poddar, R., Boelter, T., and Popa, R. A. (2016). Arx: A Strongly Encrypted Database System. *IACR Cryptol. ePrint Arch.*, 2016:591.

Popa, R. A., Redfield, C. M., Zeldovich, N., and Balakrishnan, H. (2011). CryptDB: Protecting Confidentiality with Encrypted Query Processing. In *Proceedings of the twenty-third ACM symposium on operating systems principles*, pages 85–100.

Rabin, M. O. (2005). How to Exchange Secrets with Oblivious Transfer. *IACR Cryptology ePrint Archive*, 2005(187).

Roche, D. S., Aviv, A., Choi, S. G., and Mayberry, T. (2017). Deterministic, Stash-Free Write-Only ORAM. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 507–521.

Sasada, T., Taenaka, Y., and Kadobayashi, Y. (2022). Decoupling Statistical Trends from Data Volume on LDP-Based Spatio-Temporal Data Collection. In *2022 IEEE Future Networks World Forum*, pages 262–269. IEEE.

Sasada, T., Taenaka, Y., and Kadobayashi, Y. (2023). Oblivious Statistic Collection with Local Differential Privacy in Mutual Distrust. *IEEE Access*, 11:21374–21386.

Stefanov, E., Dijk, M. v., Shi, E., Chan, T.-H. H., Fletcher, C., Ren, L., Yu, X., and Devadas, S. (2018). Path ORAM: An Extremely Simple Oblivious RAM Protocol. *Journal of the ACM*, 65(4):1–26.

Suzuki, T., Sasada, T., Taenaka, Y., and Kadobayashi, Y. (2024). Mosaicdb: An efficient trusted/untrusted memory management for location data in database. *The Sixteenth International Conference on Advances in Databases, Knowledge, and Data Applications*, pages 1–6.

Vinayagamurthy, D., Gribov, A., and Gorbunov, S. (2019). Stealthdb: a scalable encrypted database with full sql query support. *Proceedings on Privacy Enhancing Technologies*.

Yang, X., Yue, C., Zhang, W., Liu, Y., Ooi, B. C., and Chen, J. (2024). Secudb: An in-enclave privacy-preserving and tamper-resistant relational database. *Proceedings of the VLDB Endowment*, 17(12):3906–3919.

Yitbarek, S. F., Aga, M. T., Das, R., and Austin, T. (2017). Cold Boot Attacks Are still Hot: Security Analysis of Memory Scramblers in Modern Processors. In *2017 IEEE International Symposium on High Performance Computer Architecture*, pages 313–324. IEEE.

Yoshimura, M., Sasada, T., Taenaka, Y., and Kadobayashi, Y. (2023). Memory efficient data-protection for database utilizing secure/unsecured area of intel sgx. *The Sixteenth International Conference on Advances in Databases, Knowledge, and Data Applications*, pages 45–50.