# HORAM: Hybrid Oblivious Random Access Memory Scheme for Secure Path Hiding in Distributed Environment

Snehalata Funde[1]*, Gandharba Swain[2]
Department of Computer Science and Engineering
Koneru Lakshmaiah Education Foundation, Vaddeswaram-522502, Guntur, Andhra Pradesh, India

***Abstract*—Now-a-days in most of the sectors digitization has taken place to store data and process it easily with enhanced techniques. Online transactions produce very huge data daily in various sectors like health care, military, government office. To store huge data many firms, take the help of third-party organizations and store data on machines provided by them which creates new security issues. While performing operations on the data or accessing data metadata leakage may happen due to untrustworthy systems. This paper proposed hybrid oblivious random-access memory (HORAM) offers users to access their data from untrusted storage devices without sharing any information about their access patterns or techniques. Here random data block shuffling approach is used which helps in hiding storage policies about the user data blocks placement and preserving privacy of data. HORAM techniques perform pull-push operations on data in a parallel manner which in turn minimizes network overhead and reduces the execution time of operation. An extensive experimental analysis of the proposed system produces better results than weak and strong Federated oblivious random access memory (FEDORAM) respectively. The method is faster than weak FEDORAM and strong FEDORAM as it takes 0.96 seconds for communication with 5 servers whereas weak and strong FEDORAM takes 1.5 and 2 seconds respectively for reading and writing data.**

***Keywords*—HORAM; metadata; data blocks; privacy; block shuffling***

## I. INTRODUCTION

Big data analytics is becoming very easy with the evolving techniques in big data management and cloud storage. Nonetheless, placing information on untrusted servers raises security concerns. Nowadays privacy is required in every sector as everything is becoming digital [1]. Every system is getting transformed from offline form to online as it can be accessed from anywhere. Many of the people are giving priority to online system instead of offline system. In the current situation of corona pandemic most of the systems like government offices, educational systems, healthcare systems as well as private sectors put everything online to make it easy to people. Online platform opens easy entrance to security threats to enter in the database systems and obtain the information easily [1][4]. Especially if the information is extremely sensitive it becomes very harmful and owner may need to pay high cost for it if it gets stolen by third party.

If for example a health care system is considered with a database of patient records, specific record denotes each patient information, and columns reflect various characteristics. By executing queries on a particular attribute, a health care system may obtain details of the patient. For example, a point query will return results for people aged 30, while a range query would provide data for those aged between 19 and 30. Outsourcing such information to a user is a common practice that allows for efficient querying [2]. While executing queries encryption is employed because a user could always be trusted with critical information. To query the leased database effectively, system creates a key and encrypts it together with a data encryption structure, balancing system security and reliability. There are many existing techniques like encryption to protect data but it's not much efficient as data is exponentially growing. With existing cryptographic technique data can be secured but metadata about data can't be secured because it is created daily with the various web activities hosted by web server in many organizations [2][4]. Only encryption cannot secure dataset completely.

Applying encryption algorithm to user message may give information privacy, however it isn't adequate to address metadata concerns. Specifically, if information regarding access pattern get disclosed then it is going to damage whole record. By guessing access pattern attacker can get access directly to the private data. There is scope for cloud as well as other attacker for catching the leaked access pattern and misuse it [3]. Everyday internet thefts are finding new tricks to access data for the financial advantage. Thus, there is need Oblivious Random Access Machine (ORAM) is a strategy that allows customer to retrieve encrypted data from the cloud servers in secret way without disclosing path of data retrieval. Path of physical location is different from actual data access by user in ORAM. Basically to accomplish the motive of ORAM many researchers have contributed. Researchers have updated basic model of ORAM to improve performance of ORAM. As ORAM is limited in terms of complexity researchers tried to reduce it so that it will be more functional [5] to have dynamic strategy to deal with these security risks to sensitive information [4]. The Path ORAM techniques are used for security in recent years, initially anticipated by Stefanov et al. [5]. Path ORAM works to store the data blocks into the binary tree structure, including multiple leaf nodes such as buckets. Every bucket of a tree having a specific constant number of blocks which is denoted as z. When the tree has initialized, the

*Corresponding Author.

leaf bucket is defined as 0 to N-1, while each block has a random tag or position from range 0 to N [5]. Moreover, it contains a single small stash region that holds numerous blocks temporarily. If a block contains tag p, it will reside in the cache or some along the route from the base of the plant to the $p^{th}$ leaf node, according to the tree's constant.

A fully functional Oblivious RAM [6], sometimes abbreviated as ORAM, is fundamental that obscures the device's access privileges to a repository such as DRAM. In contrast, an attacker cannot learn very little about the data transmitted by watching the main memory trends. The ORAM interface converts the user's program accessing sequencing into a series of ORAM visits to seemingly random address information. Because the opponent is aware of the actual locations getting accessed, the ORAM implementation of global that the physical and logical sequence is autonomous of the proper access sequence, ensuring that the user's access sequences are not disclosed. Moreover, information stored inside database is secured using stochastic encryption to hide the captured data. Some security problem arises in hardware, software, and application levels. Several recent studies have taken use of the growing availability of trustworthy equipment for database systems.

Bajaj et al. [7] introduced TrustedDB which implements tamper-proof data aggregation using IBM 4758 PCI [8]. CryptSQLite encases the SQLite processor in an Intel SGX compartment to provide secrecy with a bit of efficiency hit [9]. ObliDB, a more recent study, improves point query speed to 722x quicker than current encrypted communication oblivious databases [10]. Access pattern threats in untrustworthy storage are identified by StealthDB and EnclaveDB which offer cryptographic solutions based on protected hardware [11] [12]. They are distinct from their ProDB regarding security border, access pattern depreciation, and high connectivity adjustments with hardware enclave, ORAM, and disk space. Hardware enclaves are used to solve database problems or build data structures with particular usage [13] [14].

ZeroTrace uses a new components library in its suggested ORAM microcontroller to offer extra protection against application attacks are launched on the SGX enclave, i.e., the oblivious positioning map access [15]. Even with processor enclaves, Oblix and ObliDB recognize the presence of access pattern leaking of the insight of database table employed in index searches and provide more effective performance than the naive worst-case buffer [16] [17]. Pro-ORAM increases system performance by utilizing the number of co Shuffle with SGX enclaves. Even though many researchers put efforts in providing security to metadata it is very difficult to fill the gap between security and practical usability of system [18].

Even though there are many ORAM techniques as mentioned above to secure metadata in online transaction they have some limitations as mentioned below.

- Traditional ORAM techniques suffer from more complexity in model construction.
- Many of the existing techniques are able to provide obliviousness to metadata but failed to improve performance with increasing data.
- As more number of users increases response time decreases.
- Existing system are unable to maintain balance between security and performance.

By considering previous works main inspiration of this paper is to seek out solution which can provide combined solution with maintaining privacy and improving performance of existing ORAM technique.

Our Contribution:

We design our system to achieve main three goals: 1) To reduce complexity and response time 2) To secure metadata with active adversary attacks 3) To improve performance of overall system with increase in number of users.

The proposed system focuses on providing security to metadata in online transactions. As previous ORAM technique strong FEDORAM [28] faces problem of high response time for communication in between client and server, our proposed system tries to reduce the execution time for pull-push operations by making the system work in parallel manner. Parallelizing tasks will optimize the ORAM system in turn reducing response time and will improve performance of overall system. As we observed weak FEDORAM suffers from sensitive data leakage problem in active adversary attacks, our proposed system focuses on protecting data from various attacks like collusion attack, session hijacking, bypass authentication, sink hole and warn hole attacks with designing an XOR-based lightweight cryptographic technique for data encryption as well as decryption during the communication.

Moreover, the further sections of the paper are divided as follows: Section II describes related work done by previous researchers. In Section III describes the algorithm for proposed implementation. The Section IV describes the experimental setup for evaluating the proposed work and results achieved with our methodology and comparative analysis with various state-of-art methods. Section V concludes the proposed work and provides future work guidelines.

## II. Related Work

Yanyu Huang et al. [19] proposed real-time oblivious data exchange into the Fog Computing. This approach can eliminate the complex execution process of the client-side and requires low communication cost, including the minimum response time, and it reduces to computation up to 2x than state-of-art methods. The Edge computing environment has been deployed, and all transactions are performed on the edge node. This system depicts an extensive experiment analysis, and it achieves low network bandwidth utilization, fixed data storage on the client machine, and minimum network overhead. The new approach of path oblivious random-access memory is called as R-Path ORAM with large root basket dimensions including the small constant size of remaining buckets in the tree [20]. A thorough examination of the root bucket capacity is carried out in order to arrive at a restricted solution for such necessary root buckets size with a minimal error possibility. Using a common platform, the effectiveness of the R-Path ORAM is assessed to that of the conventional Path ORAM. The results of the tests indicate that R-Path ORAM offers

much less server bandwidth and time taken than the original Path ORAM. This is also a hidden eviction method for reducing the size of the bottom bucket and preventing system failure.

Cao et al. [21] proposed an approach string ORAM access using spatial and temporal optimization techniques. This approach can improve the string ORAM access by using temporal and spatial optimization methodologies. Initially it recognizes dummy data blocks with significantly waste storage space and defines the optimized ORAM scheme that reduces high time computation and effective scheduling. The outcome of this approach reduces the 30% execution time complexity, thus a 40% reduction of memory utilization during the execution. A similar approach of fast and secure ring data retrieval techniques has been proposed by Yeuzhi che et. al. [22]. According to Fletcher et al. [23] secure processors have a quality and speed inefficiency of more than 50%. Fletcher et al. [23] suggested a dynamic system with a limited amount of emission allowed.

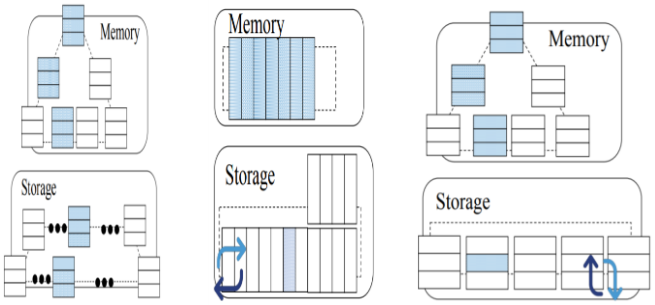
The first Path ORAM implementations on hardware were presented by Maas et al. [24]. Parallel Computing techniques have been used with implementing the super demon during the process of read and write execution. In demon, two methods were employed to improve Path ORAM's effectiveness. Treetop caching is the first method. Treetop caching saves the first coefficient of determination of the tree in the cache because only the bottom layers are changed while reading and writing to the ORAM, decreasing latency and complexity. The Phantom is the second method. The second Phantom method is min-heap evictions, which stores the cache as a min-heap and evicts the blocks that have been utilized the least in the past initially.

In addition, Fork Path ORAM was introduced by Zhang et al. [25]. Fork Path ORAM combines two successive ORAM applications. Researchers highlighted two consecutive queries could have containers in their routes which are overlapped. As a result, they recommended when a noticed request is made and the entire pathway of the requested data block is received from the servers and put into the stash, the rewriting back of the whole route be postponed until the subsequent request is made. The buckets that intersect in the two ways are not written back in the given details, and only the containers in the first request's path are published back. Furthermore, only the elements in the second route that do not overlap with the direction of the first demand were read into the cache to execute the new request. The procedure is then repeated with the second and third requests, and so on. Researchers also recommended postponing any outstanding ORAM requests. Even though all of their studies were done using the safe processor option, Sanchez [26] found that the advantage of combining the requests is negligible. Sanchez demonstrated that combining queries of size two may save one bucket. Fletcher et al. [27] proposes an optimization that uses a large group counter and many tiny individual numbers per Position Map block to condense these markers to a manageable amount.

Pujol et al. [29] presented FEDORAM. Weak and strong FEDORAM tried to tradeoff between security and performance in the instance messaging. Weak FEDORAM focused on performance of system while strong FEDORAM focused on security of the system. Weak FEDORAM suffers sensitive data leakage problem while strong FEDORAM suffers from increasing response time with increase in number of users.

Apart from the access cost imposed by ORAM procedures, contemporary ORAM architectures ignore the current computer system's extensive memory and processing hierarchy. According to [28], if the data size is higher than actual memory capacity, it directly enhances the leaf nodes of storage in the background illustrated in Fig. 1(a). Although most layers will be in the high-speed memory area, the tree-top caching has a simple design. On the other hand, each path access is converted into a series of rapid memory locations and sluggish I/O accesses. Due to the general poor locality, alternative caching methods find it difficult to adapt the tree-type structure. Such a design is improvident in terms of I/O frequency cost due to the design difference between storage and I/O access, as well as the disparity of storage and I/O use.

In FEDORAM and Multi-User Oblivious Storage via Secure Enclaves (MOSE), both techniques emphasize on reducing the input-output overhead because they extract single block data during the transaction from backend storage which is demonstrated in Fig. 1(b) and Fig. 1(c). Furthermore, the flat memory structure enables effective top-layer buffering. On the other hand, the shuffling procedure must be done often, and the whole storage must wait for such shuffled to finish before proceeding to another ORAM process. It adds extra waiting time to the process resulting in delayed output [29] [30].




(a). Path ORAM (b). MOSE (c). FEDORAM

Fig. 1. Various ORAM Techniques for Efficient Input and Output Data Access.

## III. Proposed System Design

### A. System Architecture

In the proposed system client server architecture is considered. Fig. 2 shows architecture of proposed HORAM. Initially if client $Cl_1$ wants to send message to client $Cl_2$ from server destination list ( $Ser_{dest}$ list), then the client $Cl_1$ generates the message according to below equation 1.

$$M = \{M, Rd\_l\} \quad (1)$$

Here Rd_l is the random dummy leaf of destination node list. The $Cl_1$ establishes connection with entry server $S_E$ and $S_E$ establishes connection to root server $S_R$ to forward M to $S_R$. The $S_R$ established parallel connection with Rd_l candidate servers such as distributed environment. The all-candidate servers perform the decrypt operation with given M,

and if it is accurate with server id then it is destination server selected by $Cl_1$. The same time data has been stored in internal tree structure by particular $S_D$. $S_R$ stores positionMap[id] and OTMap[id] in which the positionMap[id] describes each leaf node information while OTMap[id] gives the information of message identifier of encrypted text.

In the system architecture four terms are more important.

*1)* Client
*2)* Entry Server
*3)* Root Server
*4)* Destination Server

In this architecture direct connection between client and destination server is avoided. Instead two entities are added in between client and destination server for secure communication.

In data storage algorithm initially client generates a message to Entry server $S_E$. Then $S_E$ establishes connection with root server $S_R$ which keeps virtual id of all real and dummy messages received from entry server to destination server. Then encrypted message will be sent to all servers in the federation to get reply from actual destination server in parallel manner. The server who has authority to decrypt the message will get back to root server by decrypting message with its keys. Sending message in parallel manner saves communication time instead of sending it in parallel manner. After that root server makes entry about the current transaction id, user id and server id in its position map for further reference.

In data access algorithm, step 1 to step 3 states about connection establishment from user to root server through entry server. After establishment of connection to root server current server id for transaction is fetched and data will be extracted from specific server. After that for securing metadata and hiding path of current access to destination server current destination id will be replaced with new destination server id. Then entry for current sever id will get deleted and root server will be updated with new server id. In this way metadata privacy preserving access can be performed using the HORAM data access algorithm with employing parallelism in architecture to reduce overall response time of system.

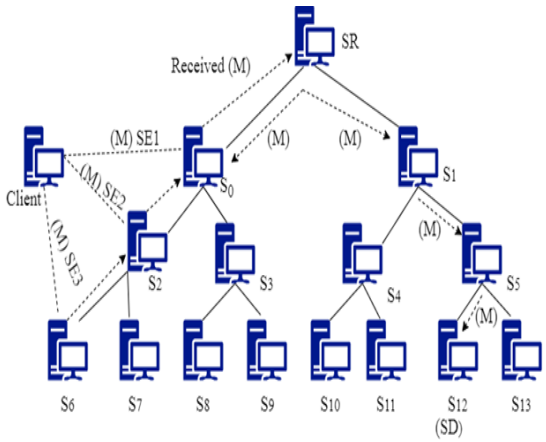



Fig. 2. Architecture of Proposed HORAM.

In data access algorithm we describe the pull activity perform by client $Cl_1$. Once data has been extracted it decrypts outside of ORAM, and perform the eviction function for update the repository of access patterns.

### *B. Algorithm Design*

Some basic data structures have been used for implementation for proposed system. Basic ORAM tree is a binary tree of encoded text. Every node of tree contains certain number of data blocks. In the below section we describe data structure used for proposed hybrid ORAM during the execution.

TABLE I. SYMBOLS USED IN PROPOSED HORAM

| Symbol | Operation |
|---|---|
| $M$ | Generated message block |
| $S(id)$ | server id |
| $U_{id}$ | Current session user identity |
| $T_{id}$ | Database transaction identity |
| $random(n, m)$ | Random function for selection from n to m |
| $S_{current}(id)$ | Current utilized server |
| $S_{new}(id)$ | Selected new server |
| positionMap | Positional map |
| Stash | Temporary buffer memory |
| R & W | Read and write operation representation |
| $S_D$ | Destination server |
| msg | Message |
| $S_E$ | Entry server |

Data storage algorithm (Push):

1: Initially client generates a message using below equation to send to entry server $S_E$, Message is the random text data.

$$M \leftarrow \text{Genereate\_Message\_Block}(msg, S_D) \quad (2)$$

2: Send message M to entry server $S_E$.

3: Once entry server receives message from client then $S_E$ creates connection to $S_R$.

Now, $S_R$ establishes concurrent online transactions with all m servers.

$$S(id) = \sum_{D=1}^{m} (\text{ReadNextServer}(D, m)) \quad (3)$$

4: Once server connection has done, according to decryption process only one server who can decrypt the data will return data. The data has forwarded to push function in the form of Push (M, S(id), $U_{id}$).

5: Then root server generates transaction id for further transaction and add entry into the positional map with encrypted data like below.

**Function-** Add_ positionMap($T_{id}$, $U_{id}$, S(id))

TABLE II. POSITION MAP

| $T_{id}$ | $U_{id}$ | $S_{(id)}$ |
|---|---|---|
| T454565 | U343 | SS203 |
| T454568 | U345 | SS204 |
| T454575 | U347 | SS201 |
| T454589 | U349 | SS203 |

6: Commit transaction

Data access algorithm (Pull):

1: Client sends message to Entry server as below:

$M \leftarrow$ requestMessage (Msg, $U_{id}$) (4)

2: Entry server gives request to root server to get allocated server with get_Server_info ($U_{id}$) and detect the allocated server for specific user.

$$S_{current}(id) = \sum_{n=1}^{m}(getServer((Msg, id)) \quad (5)$$

3: Establish connection with S(id) from entry server and extract data from

$$Dset \leftarrow getData\ (S_{current}(id), Msg) \quad (6)$$

4: Now, select server from set of servers by entry server

$$S_{new}(id) = \sum_{n=1}^{m}(random(n, m)) \quad (7)$$

5: Once server selection has done, forward data to alter function given as Alter(M, $S_{new}(id)$, $U_{id}$).

6: Entry server generates transaction id for further transaction and update the positional map on root party server like below step 7 and step 8.

7: Delete ($S_{current}(id)$, M)

8: Update positionMap($T_{id}$, $U_{id}$,$S_{(id)}$ )

Here, Table I shows symbols used in the algorithm of proposed HORAM technique and Table II shows entries of position map for particular transaction along with user id and server id.

## IV. DISCUSSION

### A. Environmental Setup

The proposed implementation is an open-source java environment with 10 data servers in parallel computation for HORAM. In the configuration setup, all are homogeneous with a single client. In all servers there should be a single entry server and single root server, and one destination server in the remaining servers.

### B. HORAM Performance

*1) Response time:* Fig. 6 depicts the average response time for the system when 100 messages sent over the network. It shows better result than existing strong FEDORAM as it took less time than strong FEDORAM with increase in number of users. It took little less time than weak FEDORAM. For 300 users it takes average 5 seconds for strong FEDORAM, 2.6 seconds for weak FEDORAM and 3.2 seconds for HORAM. It is observed with the experiment that our technique takes less response time with increase in number of servers. It takes 1.4 seconds for weak FEDORAM, 2 seconds for strong FEDORAM while 0.9 seconds for HORAM.

*2) Complexity:* Table III illustrates our innovations and compares our system to some of the most cutting-edge ORAM structures, as seen above. Where N denotes the total number of messages stored in the whole oblivious system. Because our HORAM's client-to-server connection is based on the RAM, they have similar client-to-server bandwidth and device storage complexity. The federation's communication channels and server computation are both linear.

TABLE III. PROPOSED ORAM AND EXISTING ORAM COMPLEXITY COMPARATIVE ANALYSIS

| Scheme | Bandwidth cost | Client storage | Server storage |
|---|---|---|---|
| **Weak FEDORAM** | $O(N \log N)$ | $O(N)$ | $O(N)$ |
| **Strong FEDORAM** | $O(N \log N)$ | $O(N)$ | $O(N)$ |
| **HORAM (Proposed)** | $O(N \log N)$ | $O(N \log N)$ | $O(N \log N)$ |

### C. Security Analysis

The proposed approach provides how it achieves higher security and eliminate the metadata leakage problem during communication.

- Data Generation: The client generates random message, and encrypt with proposed XOR operation techniques with the help of receiver's token id. The encryption works like one-way hash function, due to no existence of both encryption and decryption key in message generation and transmission. The encrypted data could transfer to $S_E$ and $S_R$ respectively. Moreover if $S_E$ or $S_R$ compromised with attacker even though attacker can't extract the decrypted text, due to dependency of receiver's token.
- Data Forwarding: The $S_E$ and $S_R$ can forward data to next hops or servers. Initially $S_E$ receives the M and he knows the client as well as $S_R$. The $S_E$ forward similar data to $S_R$ and generate positionMap and OTMap respectively. The positionMap[id] describes each leaf node information while OTMap[id] gives the information of message identifier of encrypted text, this information stored on root server. The $S_R$ securely keeps both records in ciphertext format that eliminates the possibility of internal or external attacks. The defined ciphertext works like a one-way hash function, which requires a negligible cost to operate; it also does not require significant dependency for encryption and decryption. Moreover, worst case, we consider root server compromised with any attacker even then they are not able to extract actual plain text due to this lightweight cryptographic policy.

- Data Extraction: When any client wants to extract the data, it gives a request to entry server $S_E$ and $S_E$ forward to $S_R$ . The message extracted from positionMap with its server information and similar requests were forwarded to SD from $S_R$ and downloaded the plain text. Once the user extracts data properly, the proposed algorithm works to provide additional security to stored information. It first erases the current record from positionMap and selects any random server from the available server set. When the user extract data from $S_R$ holds that decrypted plain text in cache memory. The selected new server and current plain encrypt again by cryptography function and generate a new entry into the positionMap. Once a new transaction is successfully committed, it erases the previous entry of duplicate data.

In proposed architecture, it can be observed that the last transaction has changed on root server into the positional map. This activity can change every time when similar frequent access request has generated by client. The stash memory auto release when time complexity generates such 2N for N data blocks. This functionality provides eliminate the dummy blocks and reduce the time as well as space complexity respectively. This algorithm automatically erases the previous entry of a particular transaction with location details from the position map when the user has performed a data pool operation. It generates and stores the new entry into the position map. The significant advantage of this functionality traitor never identifies the background knowledge extracted data source as well as the location of data source.

## V. Results

Fig. 3 describes the time required in seconds for data encryption as well as decryption. Based on this experiment, the decryption could take high time than the encryption process.

The two-way encryption techniques are also carried out to achieve security to data during transmission and dynamic decryption at the selection of the destination server. The below table we demonstrate the complexity of proposed and existing systems.

According to above Fig. 4, the data uploading and downloading time required for the client-server in the proposed HORAM. The time required based on the proposed configuration could be flexible when the operating environment has changed. Fig. 5 shows network utilization in communication with number of servers.

The performance evaluation of the proposed evaluation is based on the communication cost required for data push and pull events. When a data push event has been generated, all n receiver data nodes are utilized for communication. Furthermore, the network capacity is handled to 10kb data in a single M. The message size could be changed when the client updates the information or generate a new message. The below Fig. 5 describes a network utilization in MB during data transmission.

In another experiment, we evaluated the communication cost required for data push and pop event from $S_R$ to $S_D$ . According to FEDORAM, it describes one-to-one communication between all servers, which may produce high communication costs [29]. The proposed module generates a parallel connection between $S_R$ to all available sets of servers S.

Fig. 6 and Fig. 7 depict response with number of servers and number of users and how proposed approach reduces the computation cost than state of the art methods.

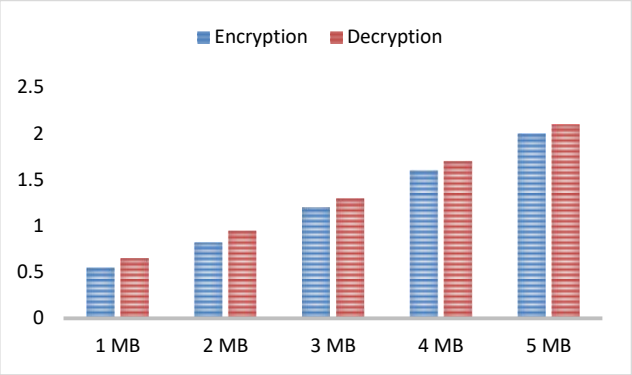



Fig. 3. Time Required in Seconds for Data Encryption and Decryption.

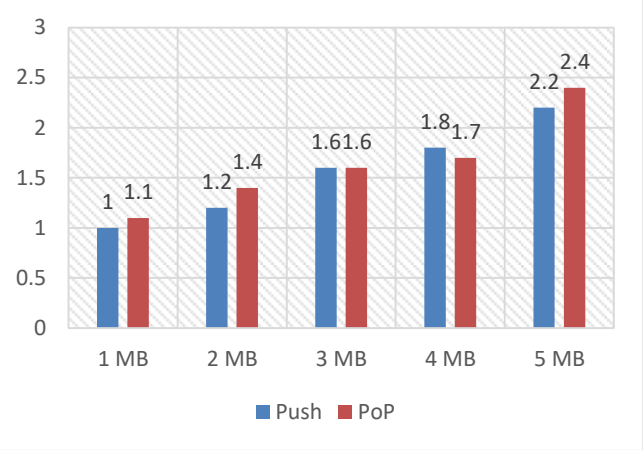



Fig. 4. Time Required in Seconds for Data Push and Pop Operation with all (10) Servers.

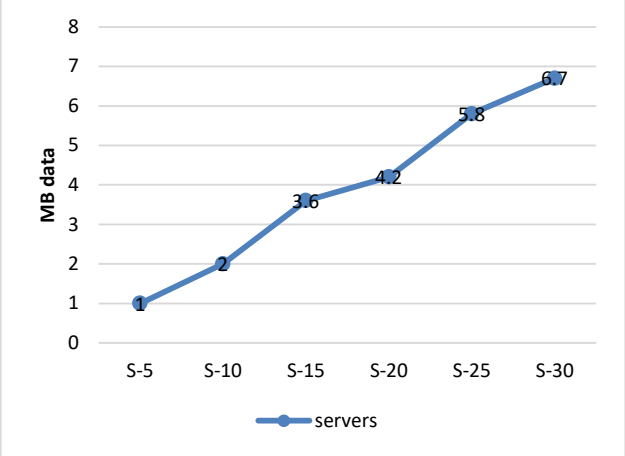



Fig. 5. Network Utilization (MB) with Number of Servers.

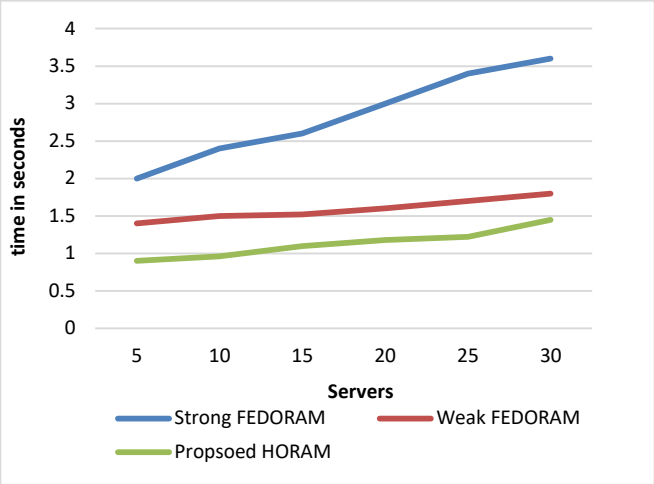



Fig. 6. Average Response Time with Number of Servers.

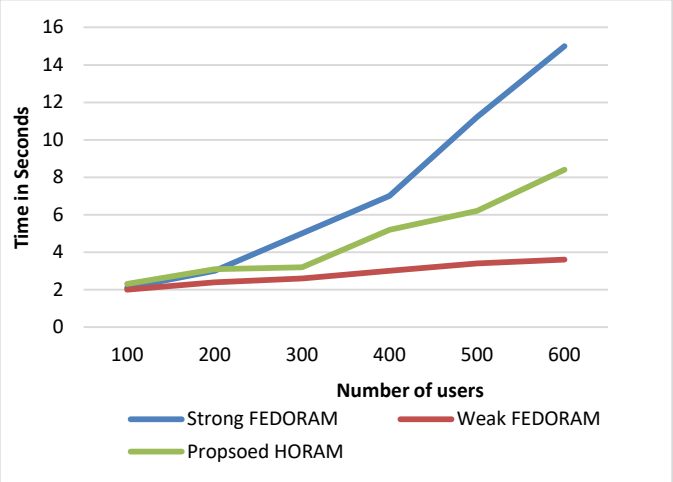



Fig. 7. Average Response Time with Number of Users.

Table IV compares different existing techniques with HORAM for response time taken in data communication.

The proposed approach has evaluated with number of users and number of servers for communication cost, based on both results our system is efficient than [29] in both experiments.

TABLE IV. AVERAGE RESPONSE TIME REQUIRED FOR HORAM AND EXISTING TECHNIQUES

| **Sr. No.** | **Techniques** | **Response time with number of servers in seconds** | | | **Response time with number of users in seconds** | | |
|---|---|---|---|---|---|---|---|
| *No. of users →* | | *N=5* | *N=10* | *N=30* | *N=100* | *N=200* | *N=300* |
| 1 | Weak FEDORAM | 1.4 | 1.5 | 1.8 | 2 | 2.4 | 2.6 |
| 2 | Strong FEDORAM | 2 | 2.4 | 3.6 | 2.1 | 3 | 5 |
| 3 | HORAM | 0.9 | 0.96 | 1.45 | 2.3 | 3.1 | 3.2 |

## VI. CONCLUSION

The proposed HORAM, an innovative ORAM approach achieves high level data privacy and low time computation in distributed environment with untrusted memory. The proposed parallel distribution HORAM provides low computation for database transaction such as push and pull respectively. Experimental analysis shows that the HORAM gives better results in terms of computation time. The method is faster than weak FEDORAM and strong FEDORAM as it takes 0.96 seconds for communication with 5 servers whereas weak and strong FEDORAM takes 1.5 and 2 seconds respectively for reading and writing operation. It improves security in comparison with weak FEDORAM by avoiding direct contact of user with destination sever and provides more privacy to metadata with data shuffling and XOR based lightweight cryptographic technique. To enhance this system with large data processing environment for achieving security and privacy of data will be addressed in future work. In future work emphasis will be on reducing complexity of encryption and decryption of extensive data.

## REFERENCES

[1] M. Suresh Babu, K. Bhavana Raj, and D. Asha Devi, “Data Security and Sensitive Data Protection using Privacy by Design Technique”, 2nd EAI International Conference on Big Data Innovation for Sustainable Cognitive Computing, 2021, ISBN : 978-3-030-47559-8.

[2] N. B. Gayathri, G. Thumbur, P. Rajesh Kumar, M. Z. U. Rahman, P. V. Reddy, and A. Lay-Ekuakille, "Efficient and Secure Pairing-Free Certificateless Aggregate Signature Scheme for Healthcare Wireless Medical Sensor Networks", IEEE Internet of Things Journal, vol. 6, no. 5, pp. 9064-9075, Oct. 2019, doi: 10.1109/JIOT.2019.2927089.

[3] A. Tarannum, Z. U. Rahman, L. K. Rao, T. Srinivasulu, and A. Lay-Ekuakille, "An Efficient Multi-Modal Biometric Sensing and Authentication Framework for Distributed Applications", IEEE Sensors Journal, vol. 20, no. 24, pp. 15014-15025, Dec.15, 2020, doi: 10.1109/JSEN.2020.3012536.

[4] R. Nellutla and Moulana Mohammed, “Survey: A Comparative Study of Different Security Issues in Big Data”, Emerging Research in Data Engineering Systems and Computer Communications, Vol. 1054, 2020, ISBN: 978-981-15-0134-0.

[5] E. Stefanov, M. V. Dijk, E. Shi, C. W. Fletcher, L. Ren, X. Yu, and S. Devadas, “Path ORAM: an extremely simple oblivious RAM protocol”, Journal of the ACM, vol. 65, no. 4, pp. 1-26, 2018.

[6] G. Asharov, I. Komargodski, W. Lin, K. Nayak, E. Peserico, and E. Shi, “Optorama: Optimal oblivious ram”, Advances in Cryptology–EUROCRYPT 2020, Volume 12106, 2020.

[7] S. Bajaj and R. Sion, "TrustedDB: A Trusted Hardware-Based Database with Privacy and Data Confidentiality", in IEEE Transactions on Knowledge and Data Engineering, vol. 26, no. 3, pp. 752-765, March 2014, doi: 10.1109/TKDE.2013.38.

[8] M. T. Basu and J. K. R. Sastry, “Enhancing Data Security under Multi-Tenancy within Open Stack”, International Journal of Advanced Trends in Computer Science and Engineering, vol. 9, no.1, January – February 2020.

[9] Y. Wang, L. Liu, C. Su, J. Ma, L. Wang, Y. Yang, Y. Shen, G. Li, T. Zhang, and X. Dong, “Cryptsqlite: Protecting data confidentiality of sqlite with intel sgx, in: Networking and Network Applications (NaNA)”, 2017 International Conference on Networking and Network Applications (NaNA), pp. 303–308, 2017.

[10] S. Eskandarian and M. Zaharia, “An oblivious general-purpose SQL database for the cloud”, CoRR abs/1710.00458, 2017.

[11] A. Gribov, D. Vinayagamurthy, and S. Gorbunov, “Stealthdb: a scalable encrypted database with full sql query support”, arXiv preprint arXiv:1711. 02279, 2017.

[12] C. Priebe, K. Vaswani, and M. Costa, "EnclaveDB: A Secure Database Using SGX", 2018 IEEE Symposium on Security and Privacy (SP), pp. 264-278, 2018, doi: 10.1109/SP.2018.00025.

[13] A. Ahmad, K. Kim, M. I. Sarfaraz, and B. Lee, Obliviate: A data oblivious file system for intel sgx”, 25th Annual Network and Distributed System Security Symposium, NDSS, 2018.

[14] H. Brekalo, R. Strackx, and F. Piessens, “Mitigating password database breaches with intel sgx”, SysTEX@ Middleware, p. 1, 2016.

[15] S. Sasy, S. Gorbunov, and C.W. Fletcher, “Zerotrace: Oblivious memory primitivesfrom intel sgx”., IACR Cryptol. ePrint Arch. 2017 (2017) 549.

[16] P. Mishra, R. Poddar, J. Chen, A. Chiesa, and R.A. Popa, “Oblix: An efficient oblivious search index, in: 2018 IEEE Symposium on Security and Privacy (SP)”, IEEE, pp. 279–296, 2018.

[17] S. Eskandarian and M. Zaharia, “Oblidb: Oblivious query processing using hardware enclaves”, arXiv preprint arXiv:1710.00458, 2017.

[18] S. Tople, Y. Jia, and P. Saxena, “Pro-oram: Practical read-only oblivious {RAM}”, 22nd International Symposium on Research in Attacks, Intrusions and Defenses ({RAID} 2019), pp. 197–211, 2019.

[19] Y. Huang, B. Li, Z. Liu, J. Li, S. M. Yiu, T. Baker, and B. B. Gupta, "ThinORAM: Towards Practical Oblivious Data Access in Fog Computing Environment", in IEEE Transactions on Services Computing, vol. 13, no. 4, pp. 602-612, 1 July-Aug. 2020, doi: 10.1109/TSC.2019.2962110.

[20] K. S. Al-Saleh and A. Belghith, "Radix Path: A Reduced Bucket Size ORAM for Secure Cloud Storage," in IEEE Access, vol. 7, pp. 84907-84917, 2019, doi: 10.1109/ACCESS.2019.2925789.

[21] D. Cao, M. Zhang, H. Lu, X. Ye, D. Fan, Y. Che, R. Wang, "Streamline Ring ORAM Accesses through Spatial and Temporal Optimization," 2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA), 2021, pp. 14-25, doi: 10.1109/HPCA51647.2021.00012.

[22] Y. Che, Y. Hong and R. Wang, "Imbalance-Aware Scheduler for Fast and Secure Ring ORAM Data Retrieval," 2019 IEEE 37th International Conference on Computer Design (ICCD), 2019, pp. 604-612, doi: 10.1109/ICCD46524.2019.00087.

[23] C. W. Fletchery, L. Ren, X. Yu, M. Van Dijk, O. Khan and S. Devadas, "Suppressing the Oblivious RAM timing channel while making information leakage and program efficiency trade-offs," 2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA), 2014, pp. 213-224, doi: 10.1109/HPCA.2014.6835932.

[24] M. Maas, E. Love, E. Stefanov, M. Tiwari, E. Shi, K. Asanovic, J. Kubiatowicz, and D. Song, “PHANTOM: Practical Oblivious Computation in a Secure Processor”, In Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security, Berlin, Germany, 4–8 November 2013, ACM: New York, NY, USA, pp. 311–324, 2013.

[25] X. Zhang, G. Sun, C. Zhang, W. Zhang, Y. Liang, T. Wang, Y. Chen, J. Di, "Fork Path: Improving efficiency of ORAM by removing redundant memory accesses," 2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), pp. 102-114, 2015, doi: 10.1145/2830772.2830787.

[26] M. Sánchez-Artigas, "Enhancing Tree-Based ORAM Using Batched Request Reordering," in IEEE Transactions on Information Forensics and Security, vol. 13, no. 3, pp. 590-604, March 2018, doi: 10.1109/TIFS.2017.2762824.

[27] C. Fletcher, L. Ren, A. Kwon, M. V. Dijk, and S. Devadas, “Freecursive ORAM: [Nearly] Free Recursion and Integrity Verification for Position-based Oblivious RAM”, Proceedings of the 20th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), Istanbul, Turkey, pp. 103-116, March 2015.

[28] S. Sasy, S. Gorbunov, and C. W. Fletcher, “ZeroTrace: Oblivious memory primitives from Intel SGX”, Symposium on Network and Distributed System Security (NDSS), 2018.

[29] A. Pujol, L. Murphy and C. Thorpe, "FedORAM: A Federated Oblivious RAM Scheme," in IEEE Access, vol. 8, pp. 187687-187699, 2020, doi: 10.1109/ACCESS.2020.3027516.

[30] T. Hoang, R. Behnia, Y. Jang, A. A. Yavuz, “MOSE: Practical Multi-User Oblivious Storage via Secure Enclaves”, Proceedings of the Tenth ACM Conference on Data and Application Security and Privacy, pp. 17-28, march 2020.