A Range Query Method for Data Access Pattern Protection Based on Uniform Access Frequency Distribution

Jing Yan[1], Zhao Chang[1], Ke Cheng[1], and Shuguang Wang[1,2]
[1]School of Computer Science and Technology, Xidian University, Xi'an, Shaanxi, 710071, China
[2]Shandong Institute of Standardization, Jinan, Shandong, 250013, China

Data encryption is necessary to keep user information secure and private on the cloud. However, adversaries can still learn valuable information about the encrypted data by observing data access patterns. To solve this issue, Oblivious RAMs (ORAMs) are proposed to hide access patterns. However, ORAMs are expensive and not suitable for deployment in a large database. In this work, we propose a range query algorithm while providing data access pattern protection based on uniform access frequency. In the preprocessing, multiple key-value pairs in the database are grouped and stored in each storage module, and we make copies for frequently accessed key-value pairs and also add some dummy key-value pairs on each storage module. In the online query processing, according to the range query length of the received query access request, we visit the specific storage module for the query and obtain the query result. Based on the techniques above, our method makes the uniform distribution of access frequency of data blocks in the database and achieves a security guarantee as strong as the state-of-the-art method. Compared with data queries that do not provide data access pattern protection, the ratio of network communication overhead is constant rather than logarithmic in ORAMs.

***Index Terms*—Data access pattern, data security, range query.**

I. Introduction

With the development of cloud computing and computer security, people's awareness of privacy protection is getting stronger and stronger. Among the increasing security and privacy techniques in large-scale data management, database encryption algorithms have made great progress in both data security and query efficiency. Encrypted databases such as Cipherbase [1], [2], CryptDB [3], TrustedDB [4], SDB [5], and Monomi [6], as well as various query execution techniques over encrypted databases [7]–[10] have been developed.

However, the most common encryption method for databases is to directly encrypt the data in the database, but executing a query request on an encrypted database will still disclose the process of this query. This leaked information includes the frequency of these encrypted data blocks being accessed, through which the correlation between data blocks, the importance between data blocks, and then the specific content of data blocks can be inferred.Therefore, simply encrypting the database and then executing query requests cannot avoid all attacks, and this method still has security risks [11]–[14]. By observing the user's data access patterns in a database, a lot of private information can be inferred. For example, it is possible to analyze the importance of different areas in the database by counting the frequency of accessing data items from the clients, and with certain background knowledge, the server can learn a great deal with client queries and/or data [15]. In another example, for a medical encrypted database containing patients' medical records, the extremely infrequent access to the medical record data area is likely to involve some rare diseases. If a user happens to read this part of the data, likely, the user is highly correlated with these rare diseases. Therefore, in big data and cloud computing applications, the privacy of users cannot be completely protected by encrypting the content of the data itself. The protection of users' access patterns is also a key research goal at present.

To that end, it is possible to protect the user access patterns from the cloud by using oblivious RAMs (ORAMs). ORAM is originally proposed by Goldreich [16] and Ostrovsky [17]. It allows a client to access encrypted data on a server without revealing its access patterns to the server. However, most ORAMs are still very expensive, and not suitable for deployment in a large database [11]. In comparison with data queries that do not provide data access pattern protection, the ratio of network communication overhead in ORAM is logarithmic.

Recently, Grubbs *et al.* [18] proposes Pancake privacy scheme, where the ratio of network communication overhead is constant rather than logarithmic, in comparison with data queries that do not provide data access pattern protection. The pancake algorithm mainly realizes the indistinguishability of data access patterns of query requests through the combination of selective replication, false access, and query batching, and supports point queries of key-value pairs in the database. Pancake assumes that the frequency of user access to data in the future can be predicted and all the queries are independent with each other. Under these assumptions, Pancake copies frequently accessed data blocks and increase the access frequency of fake data blocks, so that the overall access frequency of all data blocks satisfies a uniform distribution, and the data access pattern is protected.

However, Pancake only supports point query algorithms for data access pattern protection, but *does not provide data access pattern protection for range queries*. The query independence assumption that Pancake relies on makes it difficult to safely extend to support range queries.

In this work, we propose a range query algorithm while providing data access pattern protection based on uniform

Manuscript received December 2, 2022; revised March 15, 2023. Corresponding author: Zhao Chang (email: changzhao@xidian.edu.cn).

access frequency. The preprocessing includes the following steps. First, according to the maximum length of the range query, multiple key-value pairs in the database are grouped and stored, and several original storage modules are obtained. Second, depending on the frequency of access to the data blocks, we will store more copies of the data blocks that are accessed frequently. In order to ensure the privacy of the total number of data blocks, some dummy data blocks are also added, so that the total number of data blocks is exactly twice the original data block. Last, we calculate the frequency of false access to replicas for each key-value pair in each final storage module. In the online query processing, according to the range query length of the received query access request, we visit the specific final storage module for the query and obtain the query result. Based on the techniques above, our method makes the uniform distribution of access frequency of data blocks in the database, and achieves the security guarantee as strong as Pancake.

In brief, the major contribution of this work is to support database range queries while providing data access pattern protection based on uniform access frequency distribution, so that query algorithms can achieve the balance between data security and query efficiency. Compared with data queries that do not provide data access pattern protection, the ratio of network communication overhead is constant rather than logarithmic. Specifically, the best range query algorithms that protect data access patterns today have a feasible memory overhead, but the time overhead is too large to be applied in real life. However, the algorithm proposed in this paper uses about 1.6 times more memory overhead than the above method, but in exchange for at least 456 times less time overhead, so that the range query algorithm that protects the access mode can be applied in real life.

The rest of this article is organized as below. Section II describes the work, Sections III and IV introduce the modeling of the system and implementation options, Section V discusses the results of the assessment, and section VI concludes the paper.

II. Related Work

A. Database encryption algorithm

The algorithm is an active defense mechanism, which can prevent data leakage caused by plaintext storage, external hacker attacks that break through boundary protection, and data theft from internal high-privilege users, and fundamentally solve the problem of database sensitive data leakage [19]. Database encryption technology is the top protection means of database security measures, and it is also the technology with the highest technical requirements, and it is also very important for the stability of products [20]. At present, the database encryption technologies still used in different scenarios include pre-proxy encryption, application system encryption, file system encryption, post-proxy encryption, tablespace encryption, and disk encryption.

B. Oblivious random access machine

Current methods of protecting data access patterns include Oblivious Random Access Machines (ORAM) and other methods of protecting data access. Among them, the Oblivious Random Access Machine (ORAM) can provide data access pattern protection for data storage, and support data reading and writing and point query. ORAM is an important means of protecting access patterns, which protects information such as access operations and access locations by obfuscating each access and making it indistinguishable from random access. The use of ORAM can reduce the possibility of the attacker's speculating about private information through access patterns, reduce the attack surface of the system, and provide more secure and complete services. However, ORAM can also introduce additional client storage overhead or network communication overhead.

Oblivious RAM (ORAM) is proposed by Goldreich and Ostrovsky. It allows the client to access encrypted data in a remote server while hiding its access patterns. Since then, many ORAM constructions have been proposed; among them the Path-ORAM construction gains its popularity due to its simplicity and efficiency [21]. For a detailed analysis of various ORAM constructions, please refer to the recent work.

There exist more advanced ORAM constructions that either leverage parallelism or support multiple clients more efficiently. PrivateFS [22] is an oblivious file system based on a new parallel Oblivious RAM mechanism. The objective is to enable access to remote storage and keep both the file content and client access patterns secret. Its major contribution is to *support multiple ORAM clients*. Shroud [23] is a general storage system and functions as a virtual disk, which can hide access patterns from the cloud servers running it. It achieves this objective by adapting oblivious algorithms to enable parallelization. It focuses on *using many inexpensive coprocessors acting in parallel* to reduce operation request latency. ObliviStore [24] is a high performance, *distributed* ORAM based cloud data store. It uses an ORAM construction that there is similar to TP-ORAM [25]. ObliviStore is able to achieve high operation-level throughput by *making I/O operations asynchronous*. CURIOUS [26] fixes a security flaw in ObliviStore arising in concurrent environments. However, Sahin $et\ al.$ [27] show that CURIOUS is also insecure under asynchronous scheduling of network communication. They implement a new oblivious storage system TaoStore. TaoStore is built on a tree-based ORAM scheme that processes client requests *concurrently and asynchronously in a non-blocking fashion*.

There are also recent studies that investigate how to support the ORAM primitive more efficiently inside the architectural design of new memory technologies, $e.g.$, the recent work on designing secure DIMM with ORAM support [28].

However, there is a theoretical lower bound on the performance of ORAMs. Compared to data queries that do not provide data access schema protection, the ratio of network communication overhead is logarithmic, resulting in a significant time overhead for data query processing.

C. Oblivious query processing

Oblivious query processing techniques for specific query operations have also been explored. Li $et\ al.$ [29] study how to compute theta-joins obliviously. Arasu $et\ al.$ [13] present

oblivious query processing algorithms for a rich class of database queries involving selections, joining, grouping, and aggregation. However, their study is purely theoretical and does not lead to practical implementations. Xie $et\ al.$ [30] propose ORAM based solutions to perform privacy-preserving shortest path computation, for which earlier work explores private information retrieval (PIR) based solutions [31], [32]. In general, ORAM based solutions can provide much better performance and scalability. ZeroTrace [33] is a new library of oblivious memory primitives, combining ORAM techniques with SGX. However, it only performs basic get/put/insert operations over Set/Dictionary/List interfaces. Obladi [34] is the first system to provide ACID transactions while also hiding access patterns. The contribution is orthogonal to our study.

However, the techniques above are still limited by the theoretical lower bound on the performance of ORAMs or oblivious algorithms. Compared to data queries that do not provide data access schema protection, the ratio of network communication overhead is logarithmic, resulting in a significant time overhead for data query processing.

Recently, Grubbs $et\ al.$ proposes a Pancake privacy scheme, and the ratio of network communication overhead is constant rather than logarithmic, in comparison with data queries that do not provide data access schema protection. This algorithm mainly realizes the indistinguishability of data access patterns of query requests through the combination of selective replication, false access, and query batching, and supports point queries of key-value pairs in the database. Although this approach supports point queries against key-value data while protecting data access patterns, it relies on query independence assumptions that make it difficult to safely extend to support range queries.

D. Secure multi-party computation

Some recent work explores building an ORAM for secure multi-party computation (MPC) [35]. MPC is a powerful cryptographic primitive that allows multiple parties to perform rich data analytics over their private data while preserving each party's data privacy [36].

For oblivious query processing in MPC setting, Bater $et\ al.$ [37] propose the Private Data Network (PDN), a federated database for querying over the collective data of mutually distrustful parties, where each member database does not reveal its tuples to its peers nor to the query writer. They introduce SMCQL, a framework for executing PDN queries. SMCQL translates SQL statements into MPC primitives to compute query results over the union of its source databases without revealing sensitive information about individual tuples to peer data providers or the honest broker. Volgushev $et\ al.$ [38] point out that many relational analytics queries can maintain MPC's end-to-end security guarantee without using cryptographic MPC techniques for all operations. They implement a query compiler Conclave, which accelerates such queries by transforming them into a combination of data-parallel, local cleartext processing, and small MPC steps.

In summary, MPC ensures that all parties can obtain the computation result, but no party can learn the data from another party. That is to say, the problem setting in MPC is clearly different from our cloud database setting. Therefore, these MPC-based solutions [35]–[38] are designed for a different context and we do not evaluate them in our study.

E. Differential privacy

Differential privacy (DP) is an effective model to protect against unknown attacks with guaranteed probabilistic accuracy. Existing DP-based solutions build key-value data collection [39], build an index for range query [40] or support general SQL queries [41], [42]. Local differential privacy (LDP) has a stronger privacy model than DP [43]. Protocols satisfying LDP enable parties to *collect aggregate information* while protecting each client's privacy [44]. LDP can be applied on private spatial data aggregation [45] and answering multi-dimensional analytical queries [46], [47]. In brief, DP-based solutions [39]–[47] provide *differential privacy for query results*, while our setting is to answer queries *exactly*.

F. Side channel attacks

The core idea of this method is to obtain ciphertext information by encrypting various leaked information generated by the operation of software or hardware [48]. Intrusive, semi-intrusive, and non-intrusive attacks against security devices fall under the category of side-channel attacks. In a broad sense, side-channel attacks are often brain-opening, and there are many attack methods, such as side-channel attacks on keyboard tapping content, such as sound analysis attacks, electromagnetic analysis attacks, attacks through WIFI channel status, and attacks through kernel usage state and process information.

III. System Model

In this paper, we introduce a new system model that can be used to solve a persistent observer's attack on the database.

The basic core structural details of this range query scheme will be described in detail. It is divided into five main points.

A. Preprocess the data store

Its main job is to group and store the data in the database according to specific rules. That is a plaintext data store $KV = (k_i, v_i)$ with n key-value pairs, where $i \in \{1, 2...n\}$, grouped and stored by the maximum length of the range query. All data is stored in the database in the form of key-value pairs, and to improve the efficiency of range queries, the data needs to be stored in groups. Although this step also consumes time and memory, as the number of range queries increases, this cost will continue to be evenly distributed, and the amortized cost is much less than the cost required by the original ORAM range query. In summary, preprocessing data storage is essential. In detail, first, take a group of 2^0 key-value pairs, that is, a key-value pair as a group, a total $g = n$ group, and store it here $sub\_id : 0$; Next, take 2^1 key-value pairs in a group, that is, all key-value pairs in a group of 2, a total of $g = \lceil n/2 \rceil$ group, stored here in $sub\_id : 1$; And so on, finally, in a group of 2^l key-value pairs, all key-value pairs in a group of 2^l, a total of $g = \lceil n/(2^l) \rceil$ group, stored in $sub\_id : l$. where l needs to satisfy $2^l \leq n$. According to the maximum span of the range query, set l so that the maximum length a of the range query meets $2^{l-1} < a \leq 2^l$.

B. Selective replication

Calculate the access frequency of each key-value pair stored in the database, add multiple copies to the key-value pair with high access frequency proportionally, and distribute the access frequency of these data blocks. The rule of addition is the more frequent access, the more replicas are added. When an access request arrives, a copy is randomly selected for access, so that the final access distribution reaches a uniform distribution. Create $R(K)$ copies of the keyword key, and the number of copies $R(K) = \lceil \pi(K) \cdot n \rceil$. n is the number of keyword keys in the data store, and $\pi(K)$ is the frequency of keyword key access. n' is the total number of replicas for all key-value pairs, which ensures that there is always a total number of replicas $n' \leq 2n$.

C. Adding dummy keywords

The approach outlined above will result in a different total number of replicas n' for different distributions, which leaks information about the distribution since an attacker may learn some distribution information from n'. To avoid this leakage, here enough virtual copies are used to initialize key-value pairs so that the total number of copies is always $2n$. Specifically, a dummy keyword D with $2n - n'$ copies is added, so the total number of copies is always exactly $2n$ regardless of π. When an access request comes, a mix of fake and true queries with the same proportion will be used to ensure that the frequency of access is an uniform distribution.

D. Calculate the frequency of fake access on replicas

Calculate a complementary distribution of fake access on replicas such that the sum of the probabilities of fake and true access for any given replica is equal to $1/2n$, where $2n$ is the total number of replicas. Adding fake access in this way always ensures that the probability of accessing any keyword is equal. Choose a constant δ to satisfy $0 < \delta \leq 1$, and then use the formula $\delta(\pi/R(k)) + (1 - \delta)(\pi_f(k, j)) = 1/2n$ to calculate π_f such that the probability of accessing any replica (k,j) satisfies: the probability of a real access replica and a false access replica is a set of convex combinations.

E. Query batching

Use a random process to mix fake visits with real visits. To increase the likelihood of processing the actual access of the client immediately, a small batch of access is sent to KV' here for each client request. Specifically, when a client submits an access request to the keyword $k \in KV$, a batch algorithm is run. It randomly selects a copy of k,j, adds (k,j) to the query queue, and prepares a batch of access requests of size B. By default, we set $B = 3$. For each access, it accesses the real one based on the probability δ and the false one with a probability of $1-\delta$. Specifically, First, when the client submits an access request to a range query, for a range query of length a, find a size l that satisfies $2^{l-1} < a \leq 2^l$. Query the lth data storage area in the database, which is $sub\_id : l$. The process of each range query is decomposed into two subqueries corresponding to the range endpoint, the batch process is called, and a batch of requests of size $B = 3$ is sent for each subquery access request. Access real key-value groups with probability ¦Ä in batching and fake key-value groups with a probability of $1 - \delta$. Next, query the access request, preserving the query results of the subquery access request. Finally, the keywords corresponding to the subquery results are compared with the keywords corresponding to the query access request to obtain the query results of the query access request.

IV. Range Query Scheme

The present embodiment provides a specific description of the theoretical scheme in Section III. through simulation experiments.

In the simulation, the setup database includes plaintext data with 16 key-value pairs (i.e., $N = 16$), corresponding to $KV = (k_m, v_m)$ as(0,0), (1,1), (2,2), (3,3), (4,4), (5,5), (6,6), (7,7), (8,8), (9,9), (10,10) , (11,11), (12,12), (13,13), (14,14), (15,15). Among them, the access frequency corresponding to each key-value pair is assigned as 0.125, 0.0625, 0.03125, 0.03125, 0.1, 0.05, 0.05, 0.2, 0.0625, 0.03125, 0.03125, 0.025, 0.05, 0.05, 0.05, 0.05.

A. Preprocess the data store

First, these 16 key-value pairs are stored in groups, and a total of 5 copies need to be stored, which are recorded as $sub\_id : 0$; $sub\_id : 1$£ $sub\_id : 2$£ $sub\_id : 3$£ $sub\_id : 4$¡£ Among them, in $sub\_id : 0$, 2^0 key-value pairs are grouped and stored in the form of a group, that is, 1 key-value pair is stored in a group, that is, all data needs to be stored in one copy, and the data with the keywords 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 is stored. In $sub\_id : 0$, $sub\_id : 1$, $sub\_id : 2$, $sub\_id : 3$, $sub\_id : 4$, it is stored as a group of 2^1, 2^2, 2^3, 2^4, key-value pairs,respectively.

B. Selective replication

According to the formula $R(K) = \lceil \pi(K) \cdot n \rceil$, the number of copies of the created keyword $R(K)$ is calculated. This ensures that the sum of the number of all key-value pairs n' always guarantees that the total number of replicas $n' \leq 2n$. namely,Under $sub\_id : 0$, the number of replicas corresponding to the 16 keywords is 2, 1, 1, 1, 2, 1, 1, 4, 1, 1, 1, 1, 1, 1, 1, 1; Under $sub\_id : 1$, the number of replicas corresponding to the eight keywords is 2, 1, 2, 2, 1, 1, 1, 1, 1; Under $sub\_id : 2$, the number of replicas corresponding to the four keywords is 1, 2, 1, 1, respectively; Under $sub\_id : 3$, the number of replicas corresponding to the two keywords is 2, 1, respectively; Under $sub\_id : 4$, one keyword corresponds to one replica.

C. Adding dummy keywords

A dummy keyword D with $2n - n'$ copies is added to each data stored under sub_id, so the total number of replicas is always exactly $2n$ regardless of $\pi(k)$.

Use a random function to generate a dummy key for each data stored under sub_id. Under $sub\_id : 0$, $sub\_id : 1$, $sub\_id : 2$, $sub\_id : 3$, $sub\_id : 4$, the virtual keyword key is 3112378623742493, and the number of copies is 11, 5, 3, 1, and 1, respectively.

TABLE I
INFORMATION STORED IN $sub\_pancake$ ($sub\_id : 0$)

Key	Access frequency $\pi(k)$	Number of replicas $R(k)$	Frequency of fake access to replicas $\pi_f(k,j)$
0	0.125	2	0
1	0.0625	1	0
2	0.03125	1	0.03125
3	0.03125	1	0.03125
4	0.1	2	0.0125
5	0.05	1	0.0125
6	0.05	1	0.0125
7	0.2	4	0.0125
8	0.0625	1	0
9	0.03125	1	0.03125
10	0.03125	1	0.03125
11	0.025	1	0.0375
12	0.05	1	0.0125
13	0.05	1	0.0125
14	0.05	1	0.0125
15	0.05	1	0.0125
3112378623742493	-	11	-

TABLE II
INFORMATION STORED IN $sub\_pancake$ ($sub\_id : 1$)

Key	Access frequency $\pi(k)$	Number of replicas $R(k)$	Frequency of fake access to replicas $\pi_f(k,j)$
0	0.1875	2	0.03125
2	0.0625	1	0.0625
4	0.15	2	0.05
6	0.25	2	0
8	0.09375	1	0.03125
10	0.03125	1	0.06875
12	0.1	1	0.025
14	0.1	1	0.025
3112378623742493	-	5	-

TABLE III
INFORMATION STORED IN $sub\_pancake$ ($sub\_id : 2$)

Key	Access frequency $\pi(k)$	Number of replicas $R(k)$	Frequency of fake access to replicas $\pi_f(k,j)$
0	0.25	1	0
4	0.4	2	0.05
8	0.15	1	0.1
12	0.2	1	0.05
3112378623742493	-	3	-

D. Calculate the frequency of fake access on replicas

According to the formula$\delta(\pi/R(k)) + (1-\delta)(\pi_f(k,j)) = 1/2n$, calculate the frequency of fake access on the replica $\pi_f(k,j)$. Here let $\delta = 0.5$, a complementary distribution of fake access is calculated on replicas such that the sum of the probabilities of fake access and real access for any given replica is equal and equal to $1/2n$.

Through steps 1-4 above, the information stored in the final enclosure is summarized as shown in Table 1-Table 5.

E. Query batching

A range query for queries 11-14 is made. In the actual code run, the range span of 11-14 is calculated as 4, which satisfies $2^1 < 4 \leq 2^2$, so you need to find the data below $sub\_id : 2$, In $sub\_id : 2$, the keyword keys include 0, 4, 8, 12 as well as dummy keywords. Therefore, two real requests will be generated, one with a key of 8 and one with a key of 12. Taking an actual run as an example, according to the probability, four fake access requests are generated, namely key is 3112378623742493 twice, the key is 4 once, and the key is 12 once. At this time, the return of the fake access request is not processed, and the return of 8-15 for the real access request is compared with the 11-14 of the real request, and the values corresponding to the keywords required by the real request are returned 11, 12, 13, 14. The data returned successfully.

V. EVALUATION

In this paper, we evaluate this method experimentally. Next, the method of evaluation will be briefly described, and then, the experimental results will be described in detail.

A. Experiments with different size datasets occupying memory sizes

Experiments were performed on datasets of different sizes, and the memory occupied size was recorded, and the experimental results are shown in Fig. 1. When the number of key-value pairs is 10, the memory space occupied by the data store for preprocessing of the dataset is 336 bytes. When the number of key-value pairs is 100, the memory space occupied by the data store for preprocessing of the dataset is 3232 bytes. When the number of key-value pairs is 1,000, the memory space occupied by the data store for preprocessing of the dataset is 32,016 bytes. rORAM stands for ORAM-based range query algorithm, in which when the number of key-value pairs is 10, the memory space occupied by the data store for preprocessing of the dataset is 5192 bytes. When the number of key-value pairs is 100, the memory space occupied by the data store for preprocessing of the dataset is 9308 bytes. When the number of key-value pairs is 1,000, the memory space occupied by the data store for preprocessing of the dataset is 19,584 bytes. It can be found that the algorithm proposed in this article consumes slightly more memory than the rORAM algorithm, but the maximum advantage is about 1.6 times.

B. Experiment with the time required for 1000 range queries of different range sizes on different size datasets

Experiments were performed on datasets of different sizes, recording the time required to execute 1000 queries of different ranges under each dataset, and the experimental results are shown in Fig. 2. When the number of key-value pairs is 10 and the maximum range of a range query is 10 key-value pairs, the time required to execute 1000 range queries is 474.517 ms. When the number of key-value pairs is 100 and the maximum range of the query is 10 key-value pairs, the time required to execute 1000 range queries is 1025.68ms; when the maximum range of the query is 100 key-value pairs, the time required to

TABLE IV
INFORMATION STORED IN $sub\_pancake$ (sub_id : 3)

Key	Access frequency $\pi(k)$	Number of replicas $R(k)$	Frequency of fake access to replicas $\pi_f(k,j)$
0	0.65	2	0.175
8	0.35	1	0.15
3112378623742493	-	1	-

TABLE V
INFORMATION STORED IN $sub\_pancake$ (sub_id : 4)

Key	Access frequency $\pi(k)$	Number of replicas $R(k)$	Frequency of fake access to replicas $\pi_f(k,j)$
0	1	1	0
3112378623742493	-	1	-

execute 1000 range queries is 2718.24 ms. When the number of key-value pairs is 1000 and the maximum range of the query is 10 key-value pairs, the time required to execute 1000 range queries is 2996.9ms; when the maximum range of the query is 100 key-value pairs, the time required to execute 1000 range queries is 11788.06 ms, and when the maximum range of a query is 1,000 key-value pairs, the time required to execute 1,000 range queries is 70,579ms. In this experiment, you can see that the size of the range in the range query has a large impact on the query results. The larger the range, the longer it takes to make a query.

rORAM stands for ORAM-based range query algorithm, in which when the number of key-value pairs is 10, the time required for range query of this dataset is 18832.533 ms; When the number of key-value pairs is 100, the time required for range query of the dataset is 293052.911 ms. When the number of key-value pairs is 1000, the time required for range query of the dataset is 32239257.11 ms. The results of the comparative experiment are shown in Fig. 3. In this experiment, it can be seen that with the increase of data blocks, the proposed method in this paper is more than 456 times faster than the rORAM algorithm in terms of time.

Just 1,000 range queries on a database containing 1,000 blocks, the execution time of the rORAM algorithm is nearly 9 hours, although the security is guaranteed, but this time cost is too high. This is the real reason why the rORAM algorithm has been proposed, but it has not been applied in real life.

Through the comparison of these two experiments, it can be seen that the algorithm proposed in this paper uses the idea of exchanging space for time, and uses slightly more memory overhead than the existing methods in exchange for time overhead.Through experimental comparison, it can be found that this method is very feasible and very successful.

C. Experiment with the time it takes to make multiple times range queries on datasets of different sizes

Experiments were performed on datasets of different sizes, and the time required for multiple queries under each dataset

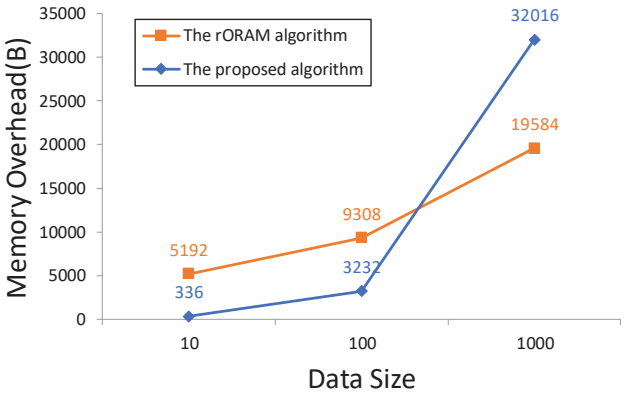


Fig. 1. Comparison of the memory overhead of the proposed algorithm and the rORAM algorithm when executing range queries on datasets of different sizes.(rORAM is an oram-based range query)

was recorded, and the experimental results are shown in Fig. 4. When the number of key-value pairs is 10, the time required for 10 range queries to record the dataset is 25.794ms, 60.374ms for 100 range queries, and 474.517ms for 1000 range queries. When the number of key-value pairs is 100, the time required to record 10 range queries on the dataset is 237.793ms, 715.95ms for 100 range queries, and 2718.24ms for 1000 range queries. When the number of key-value pairs is 1000, the time required for 10 range queries on the dataset is 4712.8ms, 9411.74ms for 100 range queries, and 70579ms for 1000 range queries.

The analysis data shows that when the number of key-value pairs is 10, 10 range queries are performed, with an average time of 2.579ms on each range query, 100 range queries are performed, the average time is 0.6037ms, and 1000 range queries are performed, with an average time of 0.474ms. When the number of key-value pairs is 100, 10 range queries are performed, with an average time of 23.779ms on each range query, 100 range queries with an average time of 7.159ms, and 1000 range queries with an average time of 2.718ms. When the number of key-value pairs is 1000, 10 range queries are performed, with an average time of 471.28ms on each range query, 100 range queries are performed, an average time is 94.117ms, and 1000 range queries are performed, with an average time of 70.579ms. In the three sets of experiments, it can be seen that as the number of range queries increases under the same conditions, the average time to each query has the same decreasing trend. Therefore, it can be analyzed that when faced with the same batch of data, the more frequently the range is queried, the higher the fitness of the method.

Since the range span size of the range query in this experiment takes the case of a random range, the size of the range in each range query will also have a great impact on the variable query time in the final experimental result, (as can be seen from Fig. 4), so the above result will be the average value obtained by taking 10 range queries for each case.

The results of the statistics, analyzed and calculated, show that in the use of this method for range query, when the number of range queries is increasing, the average time and memory overhead on each range query are getting smaller and smaller.

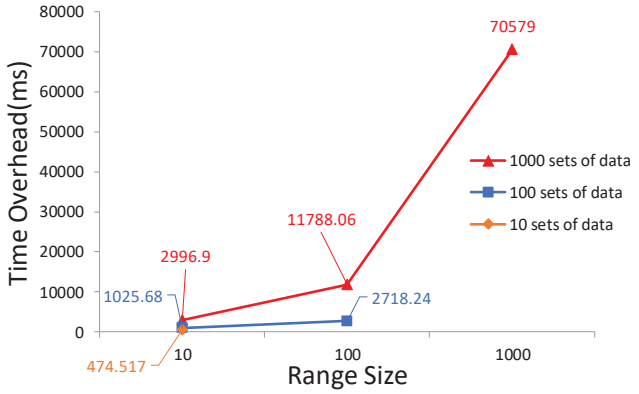


Fig. 2. Experiment with the time required for 1000 range queries of different range sizes on different size datasets.

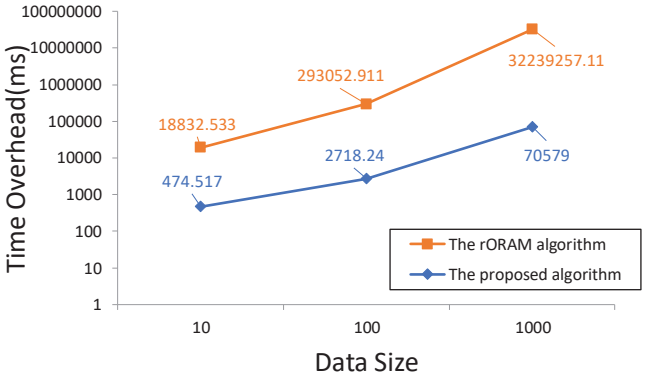


Fig. 3. Comparison of the time overhead of the proposed algorithm and the rORAM algorithm when executing range queries on datasets of different sizes.(rORAM is an oram-based range query)

VI. Conclusions

In this paper, we summarize a range query method for data access pattern protection based on uniform access frequency distribution, and the beneficial effects of this method compared with existing technologies are: First, the range query method of data access mode protection based on uniform access frequency of this method uses the method of uniform access distribution frequency to achieve uniform distribution of data block access frequency in the database, and has strong security and privacy protection functions. Second, it can quickly implement range query and can find in the stored data according to the start and end point of the range query, to improve query efficiency. Third, compared with range queries that do not provide data access mode protection, its network communication overhead ratio is constant level, and it has high query performance. In summary, this method can support database-wide queries, so that the query algorithm can achieve a balance between ”data security” and ”query efficiency”.

Acknowledgment

This work was supported in part by the National Key R&D Program of China under grant No. 2021YFB3101100, the Natural Science Basic Research Program of Shaanxi under Grant No. 2019JC-17, the Key Research and Development Program of Shaanxi under Grant No. 2022KXJ-093, the Jinan ¡®New Universities 20 Items¡¯ Introduced Innovation Team Project under Grant No. 2021GXRC064, and the Fundamental Research Funds for the Central Universities under grant No.XJSJ23040.

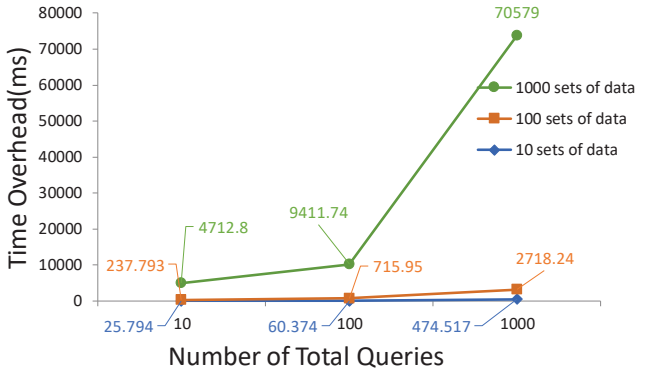
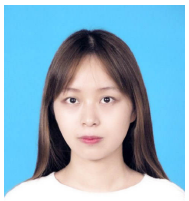


Fig. 4. Experiment with the time it takes to make multiple times range queries on datasets of different sizes.

References

[1] A. Arasu, K. Eguro, M. Joglekar, R. Kaushik, D. Kossmann, and R. Ramamurthy, “Transaction processing on confidential data using Cipherbase,” in *ICDE*, 2015, pp. 435–446.
[2] A. Arasu, S. Blanas, K. Eguro, M. Joglekar, R. Kaushik, D. Kossmann, R. Ramamurthy, P. Upadhyaya, and R. Venkatesan, “Secure database-as-a-service with Cipherbase,” in *SIGMOD*, 2013, pp. 1033–1036.
[3] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan, “CryptDB: Protecting confidentiality with encrypted query processing,” in *SOSP*, 2011, pp. 85–100.
[4] S. Bajaj and R. Sion, “TrustedDB: A trusted hardware-based database with privacy and data confidentiality,” *TKDE*, vol. 26, no. 3, pp. 752–765, 2014.
[5] Z. He, W. K. Wong, B. Kao, D. W. Cheung, R. Li, S. Yiu, and E. Lo, “SDB: A secure query processing system with data interoperability,” *PVLDB*, vol. 8, no. 12, pp. 1876–1879, 2015.
[6] S. Tu, M. F. Kaashoek, S. Madden, and N. Zeldovich, “Processing analytical queries over encrypted data,” *PVLDB*, vol. 6, no. 5, pp. 289–300, 2013.
[7] A. Arasu, K. Eguro, R. Kaushik, and R. Ramamurthy, “Querying encrypted data,” in *SIGMOD*, 2014, pp. 1259–1261.
[8] H. Hacigümüs, B. R. Iyer, C. Li, and S. Mehrotra, “Executing SQL over encrypted data in the database-service-provider model,” in *SIGMOD*, 2002, pp. 216–227.
[9] B. Yao, F. Li, and X. Xiao, “Secure nearest neighbor revisited,” in *ICDE*, 2013, pp. 733–744.
[10] W. K. Wong, B. Kao, D. W. Cheung, R. Li, and S. Yiu, “Secure query processing with data interoperability in a cloud database environment,” in *SIGMOD*, 2014, pp. 1395–1406.
[11] Z. Chang, D. Xie, and F. Li, “Oblivious RAM: A dissection and experimental evaluation,” *PVLDB*, vol. 9, no. 12, pp. 1113–1124, 2016.
[12] W. Zheng, A. Dave, J. G. Beekman, R. A. Popa, J. E. Gonzalez, and I. Stoica, “Opaque: An oblivious and encrypted distributed analytics platform,” in *NSDI*, 2017, pp. 283–298.
[13] A. Arasu and R. Kaushik, “Oblivious query processing,” in *ICDT*, 2014, pp. 26–37.
[14] T. Hoang, C. D. Ozkaptan, G. Hackebeil, and A. A. Yavuz, “Efficient oblivious data structures for database services on the cloud,” *IEEE Trans. Cloud Computing*, 2018.
[15] Z. Chang, D. Xie, F. Li, J. M. Phillips, and R. Balasubramonian, “Efficient oblivious query processing for range and knn queries,” *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 12, pp. 5741–5754, 2022.
[16] O. Goldreich, “Towards a theory of software protection and simulation by oblivious RAMs,” in *STOC*, 1987, pp. 182–194.
[17] R. Ostrovsky, “Efficient computation on oblivious RAMs,” in *STOC*, 1990, pp. 514–523.

[18] P. Grubbs, A. Khandelwal, M. Lacharité, L. Brown, L. Li, R. Agarwal, and T. Ristenpart, "Pancake: Frequency smoothing for encrypted data stores," in *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*, S. Capkun and F. Roesner, Eds., 2020, pp. 2451–2468.
[19] A. Nadeem and M. Y. Javed, "A performance comparison of data encryption algorithms," in *2005 international Conference on information and communication technologies*. IEEE, 2005, pp. 84–89.
[20] L. Bouganim *et al.*, "Database encryption," 2009.
[21] E. Stefanov, M. van Dijk, E. Shi, C. W. Fletcher, L. Ren, X. Yu, and S. Devadas, "Path ORAM: An extremely simple oblivious RAM protocol," in *CCS*, 2013, pp. 299–310.
[22] P. Williams, R. Sion, and A. Tomescu, "PrivateFS: A parallel oblivious file system," in *CCS*, 2012, pp. 977–988.
[23] J. R. Lorch, B. Parno, J. W. Mickens, M. Raykova, and J. Schiffman, "Shroud: Ensuring private access to large-scale data in the data center," in *FAST*, 2013, pp. 199–214.
[24] E. Stefanov and E. Shi, "ObliviStore: High performance oblivious cloud storage," in *S&P*, 2013, pp. 253–267.
[25] E. Stefanov, E. Shi, and D. X. Song, "Towards practical oblivious RAM," in *NDSS*, 2012.
[26] V. Bindschaedler, M. Naveed, X. Pan, X. Wang, and Y. Huang, "Practicing oblivious access on cloud storage: the gap, the fallacy, and the new way forward," in *CCS*, 2015, pp. 837–849.
[27] C. Sahin, V. Zakhary, A. E. Abbadi, H. Lin, and S. Tessaro, "TaoStore: Overcoming asynchronicity in oblivious data storage," in *S&P*, 2016, pp. 198–217.
[28] A. Shafiee, R. Balasubramonian, M. Tiwari, and F. Li, "Secure DIMM: moving ORAM primitives closer to memory," in *HPCA*, 2018, pp. 428–440.
[29] Y. Li and M. Chen, "Privacy preserving joins," in *ICDE*, 2008, pp. 1352–1354.
[30] D. Xie, G. Li, B. Yao, X. Wei, X. Xiao, Y. Gao, and M. Guo, "Practical private shortest path computation based on oblivious storage," in *ICDE*, 2016, pp. 361–372.
[31] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan, "Private information retrieval," *J. ACM*, vol. 45, no. 6, pp. 965–981, 1998.
[32] P. Williams and R. Sion, "Usable PIR," in *NDSS*, 2008.
[33] S. Sasy, S. Gorbunov, and C. W. Fletcher, "ZeroTrace: Oblivious memory primitives from intel SGX," in *NDSS*, 2018.
[34] N. Crooks, M. Burke, E. Cecchetti, S. Harel, R. Agarwal, and L. Alvisi, "Obladi: Oblivious serializable transactions in the cloud," in *OSDI*, 2018, pp. 727–743.
[35] X. S. Wang, Y. Huang, T.-H. H. Chan, A. Shelat, and E. Shi, "SCORAM: Oblivious RAM for secure computation," in *CCS*, 2014, pp. 191–202.
[36] C. Liu, X. S. Wang, K. Nayak, Y. Huang, and E. Shi, "ObliVM: A programming framework for secure computation," in *S&P*, 2015, pp. 359–376.
[37] J. Bater, G. Elliott, C. Eggen, S. Goel, A. N. Kho, and J. Rogers, "SMCQL: secure query processing for private data networks," *PVLDB*, vol. 10, no. 6, pp. 673–684, 2017.
[38] N. Volgushev, M. Schwarzkopf, B. Getchell, M. Varia, A. Lapets, and A. Bestavros, "Conclave: Secure multi-party computation on big data," in *EuroSys*, 2019, pp. 3:1–3:18.
[39] Q. Ye, H. Hu, X. Meng, and H. Zheng, "PrivKV: Key-value data collection with local differential privacy," in *S&P*, 2019.
[40] C. Sahin, T. Allard, R. Akbarinia, A. E. Abbadi, and E. Pacitti, "A differentially private index for range query processing in clouds," in *ICDE*, 2018, pp. 857–868.
[41] N. M. Johnson, J. P. Near, and D. Song, "Towards practical differential privacy for SQL queries," *PVLDB*, vol. 11, no. 5, pp. 526–539, 2018.
[42] J. Bater, X. He, W. Ehrich, A. Machanavajjhala, and J. Rogers, "Shrinkwrap: Efficient SQL query processing in differentially private data federations," *PVLDB*, vol. 12, no. 3, pp. 307–320, 2018.
[43] G. Cormode, S. Jha, T. Kulkarni, N. Li, D. Srivastava, and T. Wang, "Privacy at scale: Local differential privacy in practice," in *SIGMOD*, 2018, pp. 1655–1658.
[44] T. Wang, J. Blocki, N. Li, and S. Jha, "Locally differentially private protocols for frequency estimation," in *USENIX Security*, 2017, pp. 729–745.
[45] R. Chen, H. Li, A. K. Qin, S. P. Kasiviswanathan, and H. Jin, "Private spatial data aggregation in the local setting," in *ICDE*, 2016, pp. 289–300.
[46] N. Wang, X. Xiao, Y. Yang, J. Zhao, S. C. Hui, H. Shin, J. Shin, and G. Yu, "Collecting and analyzing multidimensional data with local differential privacy," in *ICDE*, 2019, pp. 638–649.
[47] T. Wang, B. Ding, J. Zhou, C. Hong, Z. Huang, N. Li, and S. Jha, "Answering multi-dimensional analytical queries under local differential privacy," in *SIGMOD*, 2019, pp. 159–176.
[48] F. X. Standaert, "Introduction to side-channel attacks," in *Secure inte-grated circuits and systems*. Springer, 2010, pp. 27–42.



Jing Yan received her B.S. degree from Xi¡¯an University of Technology, Xi'an, China, in 2019. She is currently working toward the M.S. degree in computer science at the School of Computer Science and Technology, Xidian University, Xi'an, China. Her research interests focus on security and privacy issues in large-scale data management.

Zhao Chang received a BS degree in computer science and technology from Peking University in 2013, and a PhD degree in computing from University of Utah in 2021. He currently works as an associate professor in the School of Computer Science and Technology at Xidian University. His research interests focus on security and privacy issues in large-scale data management.

Ke Cheng is a lecturer in the School of Computer Science and Technology at Xidian University. He received his B.S. and M.S. degree from Anhui University, Hefei, China, in 2015 and 2018, respectively. He received Ph.D. degree in computer science and technology from Xidian University in 2022. His research interests include cloud computing security, data security, and privacy protection.

Shuguang Wang is a senior engineer in Shandong Institute of Standardization, Jinan, China. He received his B.S. and M.S. degree from Shandong University, Jinan, China, in 1998 and 2010, respectively. He is studying for a doctorate in the School of Computer Science and Technology at Xidian University. His research interests include smart city, cyber security, and data security.