PRIVACY-PRESERVING QUERY PROCESSING BASED ON TRUSTED EXECUTION ENVIRONMENT AND ACCESS PATTERN OBFUSCATION TECHNOLOGIES

HAN ZIYANG

PhD

The Hong Kong Polytechnic University

2022

The Hong Kong Polytechnic University
Department of Electronic and Information Engineering

Privacy-preserving Query Processing Based on Trusted Execution Environment and Access Pattern Obfuscation Technologies

Han Ziyang

A thesis submitted in partial fulfillment of the requirements for
the degree of Doctor of Philosophy
March 2022

CERTIFICATE OF ORIGINALITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgment has been made in the text.

Signature: _____

Name of Student: Han Ziyang

Abstract

The thesis involves three research works in the field of privacy-preserving query processing. They focus on the research problems of memory level security and privacy of data querying services in the cloud hosting environment. In such a scenario, the proposed schemes consider not only the direct attacks tampering with the data and the data processing but also the threats from semi-honest adversaries in cloud platforms that attempt to derive sensitive information for inference attacks through analyzing the access pattern leakage. Motivated by these security goals, three privacy-preserving schemes are designed based on different principles and for different types of queries that comprise the body of the thesis. The first work proposes memory-secure DBMS adaptation encapsulating a bare SQL processor into the trusted execution environment (TEE) and optimizes the existing Oblivious RAM scheme to efficiently shuffle the access patterns generated in retrieving data blocks from untrusted memory for processing inside TEE. The second work provides a perturbation mechanism in a two-tier index to obfuscate the access pattern on the trapdoors of the fuzzy keyword search over encrypted document database. The TEE technology is employed to encapsulate the plaintext secondary index which is sensitive and conceals the obfuscation process. The third work gives a middleware solution to obfuscate access frequency patterns for general queries without leaking sensitive information of individual queries in a harsher threat model in which the query boundaries are exposed to attackers. Different from the former two schemes, it introduces a K-isomorphism perturbation mechanism on the query requests while not over the data storage and query processor. In each of these works, adequate literature is reviewed, and the most related works are involved in comparative evaluations. The thesis unifies the three works under a common background to summarize the research outcomes in the Ph.D. program and gives a prospect of future works.

Publications Arising from the Thesis

1. H. Zheng, H. Hu, Z. Han "Preserving user privacy for machine learning: local differential privacy or federated machine learning?", in *International Workshop on Federated Learning for User Privacy and Data Confidentiality in Conjunction with IJCAI* (2019).

2. H. Zheng, H. Hu, Z. Han "Preserving user privacy for machine learning: local differential privacy or federated machine learning?", in *IEEE Intelligent Systems* (2020).

3. Z. Han, H. Hu, "ProDB: A memory-secure database using hardware enclave and practical oblivious RAM", in *Information Systems* (2021).

4. Z. Han, H. Hu, Q. Ye, "ReFlat: A Robust Access Pattern Hiding Solution for General Cloud Query Processing Based on K-isomorphism and Hardware Enclave", in *IEEE Transactions on Cloud Computing* (2021).

5. Z. Han, Q. Ye, H. Hu, "OTKI-F: An Efficient Memory-secure Multi-keyword Fuzzy Query Protocol", in *Journal of Computer Security* (2022).

6. L. Tang, Q. Ye, H. Zheng, H. Hu, Z. Han, B.N.F. Law, "Stateful-CCSH: An Efficient Authentication Scheme for High-Resolution Video Surveillance System", in *IEEE Internet of Things Journal*(2022).

Acknowledgments

There are always a lot of 'thanks' to say when a long-last journey reaches its end. Recall the 6 years, I have encountered many difficult periods and uncountable surges of negative emotions that I could hardly overcome alone. Before my admission to the Ph.D. program I doubted whether I managed to survive the upcoming stress and complete my studies. In the middle of my studies, I was deeply anxious and frustrated about making research works admitted by the reviewers. In the last two years, tough personal affairs broke out, and I felt unable to concentrate on research. It is a great fortune that I have a considerate and visionary supervisor, Dr. Hu Haibo. First of all, he guides me to explore the research gaps when I was puzzled in seeking a worthy subject. Then, throughout my program period, he offers proficient and insightful advice with patience and urge me to improve my weak point all the way to cultivate me as a satisfactory researcher. Last but not the least, when I am trapped in difficulties, he always encourages and allows me enough time to shape up. Then, I would like to thank my parents. They did the same things from remote except for the academic advice, and they even take over most of my parental responsibilities to my daughter. I am grateful to all the colleagues in ASTAPLE Lab, Dr. Ye Qingqing, Hardy, Tang Li, etc., for their warm help. It is my pleasure to work with you and looking forward to joining our follow-up cooperations. Moreover, I am lucky to meet so many new friends here in Hong Kong and I really enjoy the hours in company with you. As space is limited, please forgive me for not listing your names here. A special appreciation is given to Ms. Li Yingying, my psychological consultant, for the lessons of self-awareness learned from you and for your efforts in rebuilding me up from inside. I also would like to thank all the related funds for supporting my Ph.D. program.

Table of Contents

List of Figures

List of Tables

2.02.0

Chapter 1

Introduction

1.1 Background

The essence of information services, including distributed storage, query services, information sharing, and machine learning is big data processing. This trend is becoming more eminent as more and more traditional services switch to data-driven models, e.g., the DaaS model [98][120]. As such, data become the core resource in this surge of industry revolution. On the other hand, data owners need necessary infrastructures to monetize the value of their data. In most cases, they need to outsource the data to external servers for high computing power and fast communication speed (e.g., a content delivery network, CDN) to perform computation-intensive and low-latency tasks. However, outsourced data usually contain sensitive information directly associated with the owner's interests and user privacy. As such, security threats are inevitable in such an

outsourcing environment, jeopardizing the trust between users and services. In recent years, many data leakage events are reported, such as those in Google Docs and the hidden malware found in Amazon Web Services (AWS), causing privacy and security concerns from a wide range of users and stakeholders.

In the scenario of data outsourcing and cloud hosting, the attacks can be generally classified into two streams based on the threat model. The first stream assumes the adversaries are malicious, who can compromise the hosts for unauthorized access to the outsourced data or data processing programs [42][80][7]. The adversaries of this kind usually aim to directly disclose the data, tamper with the data integrity, or maliciously intervene in the program logic. The other stream assumes the adversaries are semi-honest, and only eavesdrop on the access patterns on data storage. The access patterns can be generated by all the memory operations and further used to infer valuable information. Known as inference attacks [5][110][56], they are effective even when the data content is concealed in an encrypted form. Therefore, they are easier to be conducted and more difficult to be detected. For better understanding, the following examples explain how different types of access patterns can be used to explore sensitive information.

- (1) The adversary records the runtime memory operations of a database for a period of time. Then he compiles the statistics of block-wise access frequencies. According to the results, he can tell which data blocks are more frequently accessed. With auxiliary information, such as the natural distribution of the access over the data blocks, he can further disclose their plaintext content.

- (2) If the memory access pattern of an individual query request remains the same and distinguishable from others, the adversary can identify the request when the same pattern is observed. In the case when the requests are limited to their owner data, the identity of user can be revealed.

- (3) The adversary monitors the store engine of the relational database and captures an alternate access pattern on the memory space of two entity tables. He can infer that a join operation between the two tables is under processing with high likelihood.

The above examples show that the access pattern leakage should not be ignored in the design of secure query protocols. This thesis thoroughly studies the aforementioned two kinds of security threats, namely the query processing tampering and the inference attacks based on access pattern leakages. The thesis then provides efficient solutions to address these threats for typical query services in the data outsourcing environment, especially for cloud platforms. In the following sections of this chapter, I introduce the research progress in this area and give an outline of research outcomes.

1.2 State-of-The-Art Techniques

The privacy-preserving query processing has attracted the academic force of information security for decades. Now I introduce the current research progress in three parts based on the security threats they are mitigated.

1.2.1 Encryption Schemes

To fence the data confidentiality from direct attacks through unauthorized access control, diverse cryptographic approaches are widely adopted. However, these encryption schemes may decrease data utility in practice, e.g., ciphertexts are not able to be searched or computed in external servers. With this concern, new cryptographic primitives are proposed, such as searchable encryption and homomorphic encryption. Searchable encryption schemes facilitate the secure query services without revealing the plaintext data in an outsourcing environment. Since the representative searchable symmetric encryption (SSE) [27] and Public-Key Encryption with Keyword Search (PEKS) [12] were proposed, dynamic SSE [58] techniques have emerged to fill the gap in data updates, verifying the correctness of encrypted search [58], and accelerating range queries by order-preserving encryption (OPE) [2]. However, these schemes share the same disadvantage of high computational cost and failing to effectively repeal malicious data tampering and injection attacks on the application programs.

1.2.2 Trusted Execution Environment

To prevent data tampering and privilege escalation attacks even when the host is compromised, hardware-based security is proposed [46][52][6]. The principle is to seal a runtime, namely the trusted execution environment (TEE), with a set of key preset inside CPU chips and registered with a centralized attestation platform held by the CPU

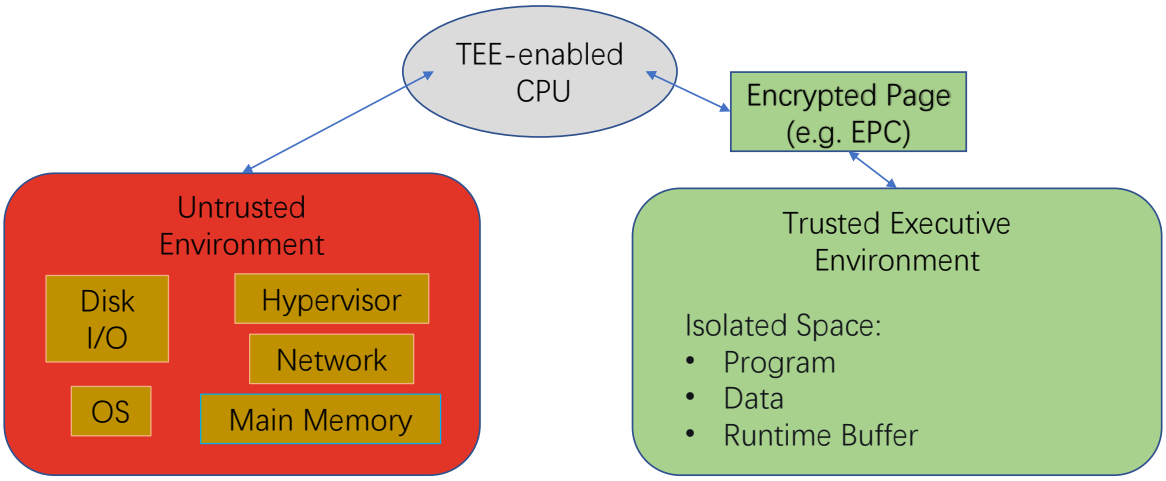


Figure 1.1: Trusted Execution Environment

manufacturers. To this end, the TEE is isolated from the operating system for an application to safely run its code and lock its sensitive data [25]. Even though the attackers manipulate privileged processes, OS administrators, and VM hypervisors, they can not further tamper with the data and code in TEE, as shown in Figure 1.1. There is a trend to support TEE in their new generation CPUs, such as the Intel flagship production Software Guard Extensions (SGX) [52] in Skylake microarchitecture and ARM TrustZone [6]). In the thesis, Intel SGX is applied in implementation and experiments, and a detailed explanation is given as follows. A TEE in Intel SGX technology is called an 'Enclave', which is executed in processor reserved memory (PRM) pre-configured in BIOS. The sealed application code segments and accessory data in the enclave are encrypted and signed by a key locked inside the Hardware Security Module (HSM). To initialize the enclave, the enclave pages containing program code and data are loaded into Enclave Page Cache (EPC) and hashed. The result is then transferred to ring 3 code before being

handled by interrupts and exits. Next, the OS process of an application invokes 'ECREATE' to turn the coded EPC page into the SGX Enclave Control Structure (SECS) and the enclave instance is born. Once created, the secure function calls (e.g. 'ECALL' and 'OCALL') are the only channels to communicate between enclave space and outside untrusted memory, while inside the enclave, the data are safe to be processed in plaintext. During its life span, a 'QUOTE' of enclave status is signed by the platform's Enhanced Privacy ID (EPID) for attestation service. The local and remote attestation service allows other enclave instances on the same host and the remote users to check with Intel Attestation Service (IAS) for ensuring they are communicating with a secure enclave application running with a trusted Intel processor so that they can safely exchange data or use data services. The procedure of remote attestation is illustrated in Figure 1.2. At the beginning stage of remote attestation, a challenge session is performed for mutual authentication between enclave client and remote user. The remote party then requests the enclave for the extended group ID (EPID-GID) of the corresponding EPID. After the EPID-GID is received, it is passed to IAS to fetch the Signature Revocation List (SigRL). The remote user then validates SigRL and requests for the report of 'QUOTE' to the enclave. During the above procedures, the two parties also complete the secret keys sharing through a Diffie–Hellman key exchange (DHKE). The shared secret is then used to produce a message digest and attached with the 'QUOTE' for verification. After verifying the message, the remote user extracts the 'QUOTE' and sends it to IAS for attestation. The IAS then verifies the cryptographic status of the enclave with the information given in 'QUOTE'. If matched, IAS informs the remote user that the enclave

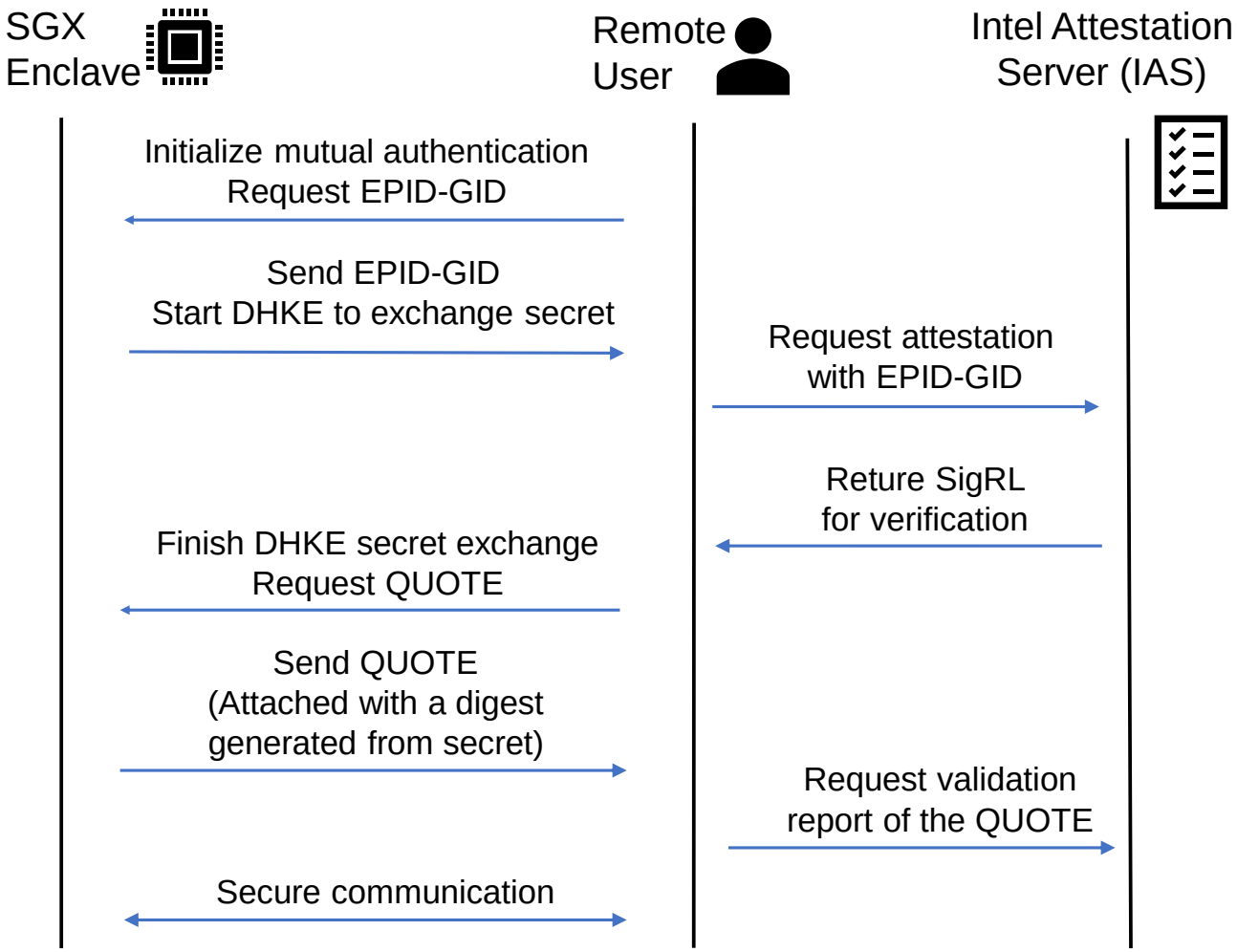


Figure 1.2: Intel SGX Remote Attestation

is trusted, otherwise, returns an error message. Despite the advantages, this technology as well as other TEE schemes show limitations in some respects. Above all, disk I/O operations and syscalls are forbidden in the enclave. Secondly, the maximum heap size for an SGX enclave is only about 90MB [75] in some mainstream OS platforms and the page swaps (i.e. inter-relocation) between the enclave and main memory are required to undergo extra validation and encryption before committing. Therefore, it is infeasible to

distribute large-scale data or run a high memory occupation program inside. Moreover, TEE is still vulnerable to some side-channel attacks as proposed in recent literature.

1.2.3 Access Pattern Obfuscation Schemes

To address the analytic and inference attacks assisted by the access pattern leakage, many protocols are proposed and continuously enhanced by the follow-up studies. The key idea is adding noises to obfuscate the patterns, so that to make the inference inaccurate, i.e. to reduce the success rate of inference towards the same as a random guess. The first kind is named private information retrieval (PIR). The original work of PIR [22] replicates the records in database, and restores the real results from the downloaded augmented query results in client-side. Some improved PIR protocols keep the database unchanged but restore real results using cryptographic approaches, such as the quadratic residues, etc. Another type of obfuscation scheme is the Oblivious RAM (ORAM) [37]. It shuffles the memory storage with randomness after each data retrieval and meanwhile performs a client-side search for real requests. The random perturbation in server storage allows the Oblivous RAM to hide most of the access patterns except the access intervals. The ORAM schemes gain more popularity than PIR protocols in academia, as the randomness employed in ORAMs are more reliable than computational hard problems (have the absolute solution) on which the PIR schemes rely when the computing power largely grows in near future. However, there is a tradeoff between higher robustness and the high cost in the server-side. Many variants and optimized ORAM

protocols [38][96][28] effort to improve the performance, while unfortunately, there is still distance from practical use. A detailed introduction of ORAM and representative ORAM protocols can be found in Chapter 3.

To provide a more practical solution to conceal the access patterns for different types of queries, a rich body of existing works involve obfuscation schemes designed according to the characteristics of specific data and query processors, e.g. KV pairs, keyword search, and relational database. To be best of my knowledge, except for downloading the entire database to a local client, there is yet perfect obfuscation schemes completely all kind of access patterns including the trivial ones. Hence, it is important to define the sensitive access patterns to be protected on demand.

1.3 Thesis Contribution

The thesis consists of privacy-preserving schemes for three distinct queries. The first work (see Chapter 3) implements a memory-secure adaptation of relational DBMS. Leveraging the TEE technology, a minimized SQL processor is located in Intel SGX to achieve a tamper-proof runtime memory. For loading the data into the enclave application without leaving sensitive access patterns, an Oblivious RAM protocol is applied, and its client is deployed in the enclave alongside the SQL processor. That said, the SQL is safely processed in the TEE, and the data is retrieved from untrusted main memory through the ORAM. The ORAM protocol optimizes the Path ORAM on its efficiency

of data retrieving and meanwhile offering the additional capability of inter-table access pattern hiding. This adaptation also mitigates the access pattern leakage in the data persistence stage of the store engine with a Probabilistic Lazy Persistence mechanism.

The second work (see Chapter 4) addresses the index disclosure problems on fuzzy keyword search by substituting the tradition trapdoor index with a two-tier index structure. In the system design, a small secondary index is preloaded into the SGX enclave as meta-data and an edit-distance-based obfuscation mechanism is proposed to obfuscate the access pattern to keyword index in untrusted memory. Moreover, a trend-aware cache is provided to effectively reduce the cost of data retrievals from enclave to external memory.

The third protocol (see Chapter 5) proposed is a middle-ware solution to hide the access frequency pattern for general queries. It virtually re-organizes the data blocks into a K isomorphic structure inside the TEE and duplicates the query with the same isomorphism operation before dispatching to the query processor in untrusted memory. By doing this, both the access frequency on each block and the query requests are always obfuscated with at least K isomorphic duplications. Therefore, on one hand, the block-wise access frequency distributions are smoothed and the ranking and order of the frequency are concealed. On the other hand, the request intention of individual queries is also protected from the adversary in the proposed threat model where the query boundaries are disclosed. In addition, two working functions are given for point and range queries separately to achieve a higher degree of perturbation on access frequency patterns.

1.4 Thesis Structure

Chapter 2

The review of literature and Related works.

Chapter 3

Proposed research work: Memory-secure database adaptation using hardware enclave and practical oblivious RAM.

Chapter 4

Proposed research work: Efficient memory-secure multi-keyword fuzzy query protocol.

Chapter 5

Proposed research work: Robust access pattern hiding module for general queries based on k-isomorphism and TEE.

Chapter 6

Conclude the thesis and list plan of the further works.

Chapter 2

Review of Related Literature

In this chapter, I review the literature and related works associated with the subjects in the thesis. To present a systematic introduction of them and the relations to the proposed schemes, they are categorized by the research directions in the following sections.

2.1 Optimizations of Oblivious RAM

In the subject of memory-secure database adaptation, a rich body of literature about Oblivious RAM is surveyed. Since the idea was first invited by Goldreich et al. in 1987 [37], generations of ORAM schemes are proposed to provide optimizations mainly on the protocol performance. TP ORAM [95] and Path ORAM [96] achieve $O(\log N)$ client cost measured by amortized number of blocks accessed per client operation. Goodrich et al. [38] achieves $O(\log^2 N)$ access bandwidth overhead, and Circuit ORAM

[107] further reaches $\omega(\log N)$ bandwidth blowup. More recently, many ORAM schemes trade server computation cost for communication cost, such as Path PIR [73], Ring ORAM [85][86] and Onion ORAM [28]. Among them, Bucket ORAM [33] with additive homomorphic encryption is remarked for providing single roundtrip (i.e., one single client-server interaction) with $O(1)$ bandwidth blowup. Balanced in bandwidth and storage cost, and get benefit from its simplicity in implementation, the Path ORAM becomes the most popular Oblivious RAM protocol and attracts numbers of follow-up studies to optimize its efficiency. A detailed introduction of Path ORAM can be found later in Chapter 3.2. As a representative extension work, PrORAM [115] operates directly at the block level by locating shared blocks on the same path of the Path ORAM tree and retrieving them as "super blocks" in one pass. The proposed Sap ORAM provides optimization from another aspect. It achieves the performance improvement leveraging the features of join query across tables (relations) without intervening the randomness of block position on ORAM tree, thus furthest preserves the obfuscation capability of Path ORAM. On the other hand, although based on Path ORAM, the path sharing mechanism in SaP ORAM can also be adapted to other ORAM schemes for database workloads. With regard to adopting ORAM schemes in the multi-user scenario, another track of research works [57][60][18][109][17] effort to achieve high-speed concurrent access over the ORAM server. The early ones, represented by [57][60][18], lay emphasis on the security of concurrent retrieval and the synchronization latency. Leveraging identity-based signature, shuffle correctness proof mechanism, as well as the blockchain technologies, the latter ones [109][17] provide traceable and unforgeable ORAM update

to deal with arbitrary modifications from malicious clients.

2.2 Secure Query Processing Using TEE

Trusted Execution Environment (TEE) technology is widely adopted in applications to provide tamper-proof query processing. The related literature is discussed from three tracks as follows.

Database Applications Using TEE. In respect of SQL queries, Bajaj et al. proposed TrustedDB [9], which uses IBM 4758 PCI [46] to implement tamper-proof query processing. CryptSQLite [108] encapsulates the SQLite engine in an Intel SGX enclave to achieve confidentiality with a modest performance drop. More recent work, ObliDB [32], further enhances the performance of point query to $7-22$x faster than the existing encryption-based oblivious database. StealthDB [40] and EnclaveDB [82] identify access pattern attacks in untrusted memory/storage and propose cryptographic solutions using secure hardware. They differ from the proposed adaptation in security boundary, optimization of access pattern approaches, as well as high coupling adaptations with hardware enclave, ORAM and disk storage. Other works, such as [3], [15], Qshield [20], and VeriDB [121] are also surveyed for acquiring inspirations in adopting hardware enclaves in the database for mitigating privacy issues and referencing their methodology in tackling performance loss in different database systems. Among them, Qshield [20] is highlighted for their innovation on multi-user control mechanism that notably simplifies

the process of multi-user authentication in TEE. Therefore, this improvement further enables the multi-user function in TEE-based secure database systems.

General Query Processing Using. With the increasing availability of trusted hardware, they are found used in other query services other than relational databases. [62] and pRide [70] encapsulates location-based query processing with IntelSGX enclave. [97] and [74] implement the prototype keyword search protocol using IntelSGX. In parallel with the proposed works in the thesis, Rearguard [97], [111] and SPEKS [114] migrate the search and indexing computation into the isolated buffer of the Intel SGX enclave. However, they are different from the proposed work in the thesis in supporting fuzzy search, oblivious primitives used for access pattern hiding, and practical concerns of enclave memory space. [4] and [69], leverage the secure hardware enclaves to guarantee the confidentiality and integrity of user-profiles in social network discovery. [8] and [39] is also commented for implementing the privacy-preserving modules with Intel SGX enclave for IoT information exchange, which paves a new application scenario for hardware enclave technology.

Combining use of ORAM and Hardware Enclave. In parallel with the proposed memory secure database adaptation in the thesis, other protocols [88][74][31][99] that combine the two cryptographic primitives in the design begin to emerge. ZeroTrace [88] provides additional security against software side-channel attacks on SGX enclave, i.e., the oblivious position map access using a novel assembly-level library in their proposed ORAM controller. Their contribution is stand-alone and can be complementary to my

work. Both Oblix [74] and ObliDB [31] realize the existence of access pattern leakage of the depth/size of data structure applied in index search even with hardware enclaves and further give more efficient solutions to this problem than the naive worst-case padding. Pro-ORAM [99] improves the system throughput by using multi-threading Melbourne Shuffle [76] with SGX enclaves while the proposed work fully utilizes the features of SQL queries.

2.3 Privacy-preserving Fuzzy Keyword Search

In the subject of privacy-preserving fuzzy keyword search, I survey the related literature that is also devoted to addressing the security and privacy issues in the area of keyword search. Majority of the literature extend the fundamental studies on keyword exact/fuzzy search [12][102][66][27][84] with security improvements. They can be generally divided into two tracks. One seeks performance enhancements upon cryptographic primitives for different demands of queries. For multi-keyword conjunctive fuzzy search, I remark [23][13][105][103] for their notable progress on query performance leveraging diverse technology dependencies. Among them, [23] optimized Bed-tree [117] with privacy-aware feature and [103] used the locality-sensitive hashing (LSH) [48] and Bloom Filter [11] to build index. For rank-ordered fuzzy search, Wang et al. initially gave a baseline solution in [106]. Fu et al. addressed its limitation on 'one-letter-mistake' later in [35]. More recent work [29] further improved the processing speed with the aid of term frequency–inverse document frequency (TF-IDF). Among the studies of preferred

keyword query, [90] introduced keyword weight in relevance scoring and reached acceptable efficiency. A follow-up solution [34] applied a history-oriented user interest model to achieve efficiency and personalization yields. Nevertheless, the exposure of runtime memory makes their solutions vulnerable to privileged attack methods [16][119] in cloud platforms. The other track focuses on exploring security improvements with various tools. By optimizing the architecture of service, [47] traded communication overhead between two cloud servers for the security improvements. By adopting user authorization and access control, the representative works, [67] and [91] reduced the data owner's privacy leakage resulting from search results by delegating search capabilities to data users using Hierarchical Predicate Encryption (HPE). However, they are incapable of large datasets or multi-keyword queries due to the slow processing and remain compromised with 'semi-honest' adversaries who possess extra knowledge about searching manners of users. In contrast, the proposed scheme in the thesis simultaneously achieves both query performance and highly complete memory security.

2.4 Access Frequency Pattern Hiding for General Query Processors

In my research work of designing an access frequency pattern hiding scheme for general query processing, I study massive previous works that address the access frequency pattern leakage of diverse categories of queries over encrypted data. I remark

[112][81][64][78] in hiding the frequency patterns of keyword searching and [63][72][41] for their solutions in concealing access frequency distributions of general database queries against passive attackers. Among them, the most similar scheme to my design in Chapter 5 is PANCAKE [41]. It creates replicas towards the frequently accessed data (i.e. key-value pairs) and generates dummy access over them to make the distribution appear to be flat. My work differs from it in three aspects: First, all the components of the proposed scheme run inside hardware enclave while do not rely on any intermediate proxy which may generate additional risk of compromise. Second, my design maintains the real-time access pattern in Frequency Snapshot which shares the same vision as adversaries. This feature eliminates the uncertainty of distribution prediction based on the histogram that applied in PANCAKE. Third, PANCAKE currently serves for KV-pairs storage while the proposed work is designed for general point and range queries. The idea of introducing dummy queries is also found in a more recent work [89], where Sepehri et al. achieve constant overhead of 7 to 13 such fake queries per real one in smoothing the frequency ranking pattern. Their disadvantage is that a local cache is required in their design. [116][68][19] are also remarked for using similar duplication mechanism as mine to obfuscate access frequency patterns. The main difference in implementation is that they imposed the duplication mechanism directly on data storage and assume that the adversary does not understand the program logic of their obfuscation protocols. However, in many cases, the semi-honest hypervisors or compromised OS administrators are able to analyze all the intermediate block-wise memory operations, including memory copies, lookups, and references, other than simply recording the block-wise access

frequency distributions. Therefore, the adversary still holds the possibility to crack the real requests. Intuitively, he may monitor the logic entrance of obfuscation protocol or the point that the original request is resolved and then distinguish the requested blocks from the protocol inputs or decoding results respectively.

Chapter 3

Memory-secure database adaptation

3.1 Introduction

With the prosperity of database outsourcing, an increasing number of data owners and service providers host their database management systems (DBMS) of web and mobile applications in cloud platforms [1][83] for higher cost-performance. However, emerging attacks as introduced in Chapter that target on cloud infrastructures become a major threat [59][87] harming the reliability and credibility to data users and owners. On one hand, the hosting platforms face traditional attack methods which allow adversary to obtain unauthorized access to oursourced data. On the other hand, they suffer from inference attacks [5][110][56] exploiting the access pattern of all levels of memory activities of VM. The access pattern statistics used by inference attacks, such as access frequency on memory addresses, access operation type (i.e., read and write), and disk

I/O throughput over a period of time, are resistant to traditional database encryption [77][104][56][55]. In respect of the store engine, the adversary can even learn the structure of a tree-based index by mapping the sequences of block accesses to traversal paths in this tree. Furthermore, such attacks do not require full access to the memory space, they are easy to perform and difficult to be detected. Motivated by this, the first subject of the thesis is devoted to providing a practical solution to address the above concerns in conventional DBMS.

Although the principle of inference attacks is introduced in Chapter 1.1, to depict its consequences and feasibility in real-world cases, a preliminary experiment is conducted. I adopt VMware vSphere ESXi, a leading hypervisor used by many cloud services, on a bare-metal machine. Then install MySQL 5.6 database on its VM image and run 5 TPC-H [100] OLAP SQL queries ($\#Q1, Q6, Q15, Q19, Q20$[1]) each by 50 runs. The integrated analytical tool ESXi vCenter is used to monitor the number of disk reads, small-range seeks, medium-range seeks on this VM and also calculate the small-to-medium seek ratio. Then the success rate of such an inference attack is evaluated based on these parameter readings.

Table 1 shows the average results over the 50 runs. The 4 parameter in the table columns is involved as the features for classification. The first 40 runs are used as training data to build a naive Bayes classifier, and the rest 10 runs are regarded as testing data to reidentify their query IDs (sensitive). The accuracy of this attack is around 84%, which

[1]These queries are randomly selected to be listed from the queries which can not be easily distinguished from massive test results over all TPC-H queries under given attributes.

Table 1: Disk Access Patterns of TPC-H Queries

Query	Read Requests	Small	Medium	S/M Ratio
1	2813	53832	2371	22.89
6	2812	45579	10688	4.31
15	5638	91622	21016	4.40
19	2918	56046	2283	24.55
20	15460	235021	18627	12.67

is much higher than 20% by random guess.

As introduced in Chapter 1.2.3, the Oblivious RAM is a well-studied solution for such inference attacks. Nevertheless, a critical issue to adopt ORAM in the cloud database, however, is that the untrusted VM can only serve as data storage (i.e., an ORAM server), while most of the DBMS components, such as the query processing engine, must be hosted in the data owner side (i.e., an ORAM client). This will pose not only intensive computation on the data owner but also a large volume of network traffic from and to the VM, which negates the purpose of a cloud solution.

Fortunately, with the advances in hardware-based security, a trusted execution environment (TEE), such as Intel SGX [52][54] and ARM TrustZone [6]) becomes a mandatory component in modern CPUs. This emerging technology enables us to **push the ORAM client into the cloud VM** so that the communication between ORAM client and server incurs memory access only. With this key idea, in this work, a practical DBMS adaptation is proposed, namely the ProDB. It employs a memory access obfuscation mechanism that is immune to inference attacks. The adaptation is in a minimal design of a traditional DBMS that separately runs on the Intel SGX Enclave [52] and on the untrusted memory, connected by an Oblivious RAM protocol. Notice that, challenges in

applying this TEE technology to real-world applications are: (1) limited computing and memory capacity in the SGX Enclave, which is also true for other TEEs, such as ARM TrustZone, and (2) I/O inefficiency due to the frequent block re-encryption by ORAM. ProDB optimizes the resource allocation, block I/O, and ORAM access frequency to address these challenges. To this end, I further propose a SQL-aware Path Oblivious RAM protocol [96] named SaP ORAM that is tailored for processing SQL queries. It has the following two features.

- **Probabilistic Lazy Persistence.** SaP ORAM probabilistically lowers the I/O consumption of those dirty blocks due to ORAM re-encryption, by introducing modest randomness in the lazy persistence process.
- **SQL-aware ORAM Path Sharing.** SaP ORAM organizes relevant tables that are often jointly accessed into a single ORAM instance. As such, multiple block accesses to these tables can share the same path of a single ORAM access. SaP ORAM optimizes this task by applying the maximum weighted matching algorithm [30][36] on the history query-table graph.

To summarize, the main contributions of this thesis work are as follows.

- I identify and resolve access pattern monitoring attacks by hypervisors using a "hardware+software" (or more precisely "enclave+ORAM") solution ProDB, which optimizes the allocation of limited hardware resources and the use of ORAM.
- A novel ORAM protocol, SaP ORAM, is proposed to significantly enhance the

efficiency over the classic Path ORAM by introducing probabilistic lazy persistence and SQL-aware path sharing for a practical database workload such as the TPC-H benchmark [100].

- The security functionality of access pattern hiding is rigorously analyzed in SaP ORAM and series of experiments are conducted to demonstrate the system performance of ProDB.

The rest of the chapter is organized as follows. An introduction of Oblivious RAM and Path ORAM is given in Section 3.2. The system model and the design of ProDB are presented in Section 3.3. The SaP ORAM protocol is shown in Section 3.4. Security analysis of SaP ORAM and ProDB is given in Section 3.5, followed by the experimental results and discussion in Section 3.6. Section 3.7 summaries this chapter. Some related notations used in this chapter are listed in Table 2.

3.2 Oblivious RAM

In the section, I introduce the concept and principle of Oblivious RAM, which is applied as the obfuscation scheme for access pattern leakage in the work.

Oblivious RAM [37][92][95][96][28][33][107] is a privacy-preserving data retrieve protocol for a data owner to safely access a remote storage in an untrusted environment while hiding her access pattern. It uses random permutation, shuffle of memory cells, and symmetric encryption on data to hide the original access sequence.

Table 2: Notations and Symbols

Symbol	**Definition**
Path ORAM Parameters:	
L	depth of tree (for Path ORAM and SaP ORAM)
Z	bucket size
B	block size
SaP ORAM Evaluation Metrics:	
λ	# of tables combined in a SaP ORAM instance
R_d	# of retrieve rounds occur in query requests
ω	computational cost of processing a block on ORAM client
ψ	efficiency gain in one pair of table
Ψ	overall gain of all table pairs in database
Υ	intra-table skewness of table pairing plan
Φ	inter-table skewness of table pairing plan
Database Parameters:	
$H_{K_h}(\cdot)$	MAC function of incoming queries with hash key K_h
$E(\cdot), D(\cdot)$	encryption and decryption function of query requests
K_u	symmetric key for encrypting user requests
T_i	database table index
N_i	# of blocks of table T_i
P	database page size
$J_{i,j}$	# of joint accesses of table T_i and T_j in history
S_i	# of non-parallel accesses of table T_i in history

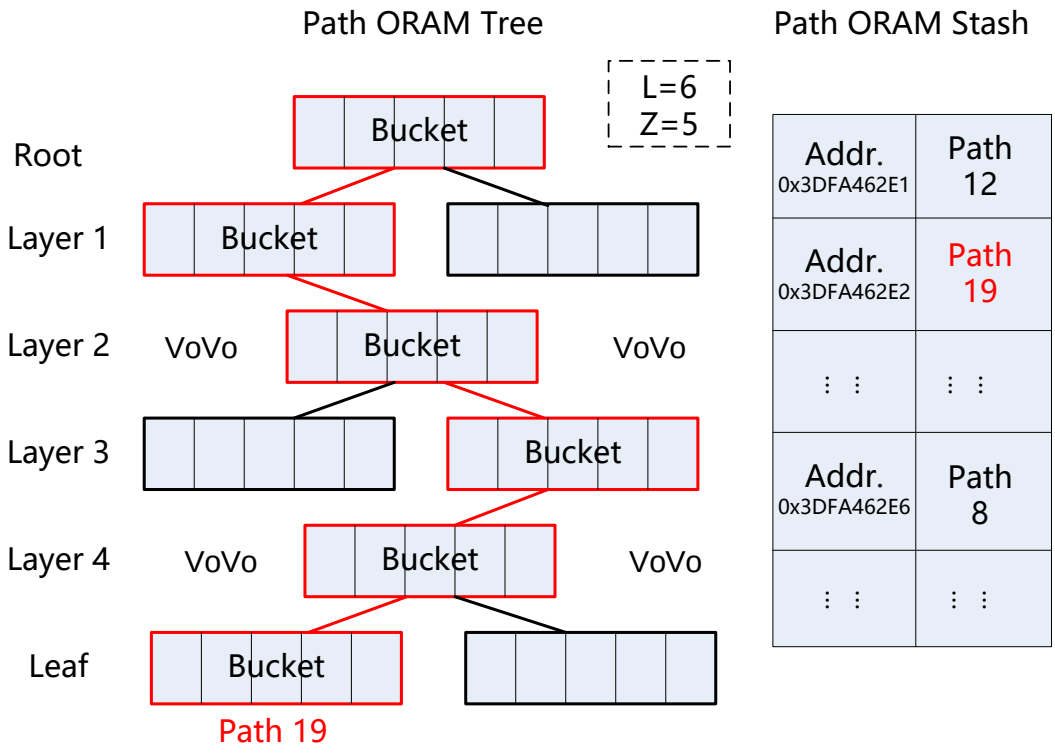


Figure 3.1: ORAM Path and Stash

As the proposed SaP ORAM is based on Path ORAM [96], I briefly introduce the latter in the following.

As shown in Figure 3.1, in Path ORAM data blocks are lodged in a tree structure. Each node on the tree is called a 'bucket' with a fixed size of Z blocks. A 'path' refers to all the buckets from a leaf node to the root. At the client-side, a position map stores the mappings of block addresses to paths. Due to the randomness of path mapping and the mechanism of path write-back (mentioned later), after each ORAM access, some candidate blocks may fail to be passed back to the ORAM server. In Path ORAM, a client cache, called stash, is set to lodge these overflowed blocks.

In each ORAM access, the client first lookups the position map to find the path assigned

to the target block, then notifies the server to send that path and merges them into the stash. After executing read or update operations, ORAM client re-maps the target block randomly to another path and re-builds each node on the path. The latter is achieved by traversing the stash to find candidate blocks whose assigned paths intersect with the retrieved path at each layer of the tree (see Figure 3.1). In this step, if more than Z candidate blocks are available for a bucket, the surplus ones have to remain in the stash. Otherwise, in the case that the candidate blocks in the stash are insufficient, the path will be padded with dummy blocks. As the last step, the client re-encrypt and writes the entire path back to the server.

3.3 ProDB Architecture

3.3.1 Overview

Security Model

Since the underlying threats comes from both the unauthorized data access and analytic attacks over memory which is different from conventional terms, the 'memory-secure DBMS' is formally defined as follows.

Definition 1. *(Memory-secure DBMS): Let $\vec{Q}$ denote a set of SQL queries executed by a processing algorithm P and R denote their results. In this process, P leaves access pattern ξ in the main memory. An adversary can conduct an attack function f that either*

tampers $\vec{Q}$, P and R for their interests or exploits I_p and ξ for sensitive knowledge. A DBMS is memory-secure if and only if it satisfies the following two conditions. (1) $\vec{Q}$, P, and R are inaccessible to any attack f that can access the main memory space. (2) For any attack f that attempts to infer $\vec{Q}$ or R, learning ξ only negligibly increases the success rate, or formally, $prob(f(\vec{Q})|\xi) = prob(f(\vec{Q})) + \epsilon_1$ and $prob(f(R)|\xi) = prob(f(R)) + \epsilon_2$, where ϵ_1 and ϵ_2 are negligible.

Remark 1. *Notice that, the access pattern leakage can not be eliminated in the whole site of memory space. In this work, the security boundary of access pattern leakage is restricted under the level of relation data access and the intention of the SQL queries. Other pattern leakages are regarded as less sensitive, such as whether the two queries access the same database instance, and they are beyond the scope of my study.*

System Model

As shown in Figure 3.2, assume that a database owner (e.g. a data service provider for business purposes) outsources its database on a virtual machine (VM) in the cloud hosting platform. To satisfy the second condition of memory-secure DBMS, an ORAM scheme is employed to obfuscate the memory access pattern ξ generated by the operations of the store engine. To satisfy the first condition, a TEE (e.g., Intel SGX enclave) is assumed to be available in the VM so that query processing and ORAM clients (collectively called ProDB Core) can run in it while the ORAM servers run in the regular operating system (i.e., untrusted space) of the same VM to interact with the external

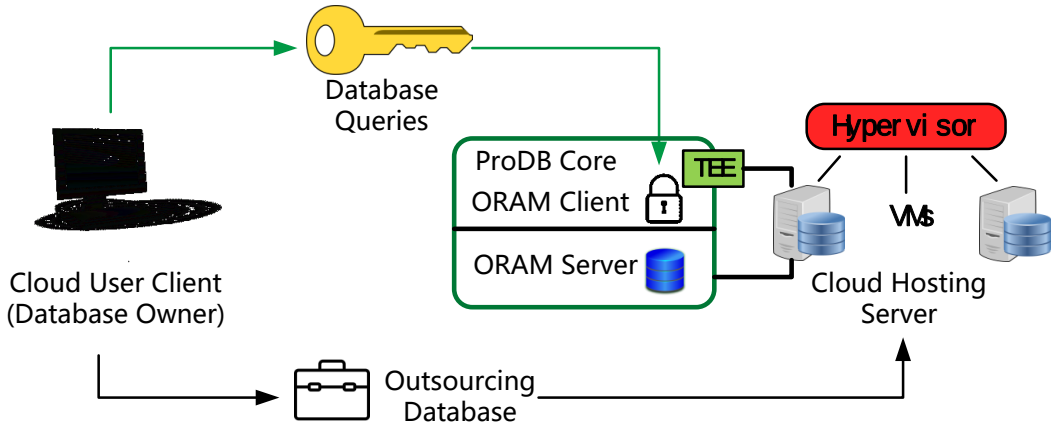


Figure 3.2: System Model

storage.

Performance Metric

As the ORAM cost is the predominant cost (w.r.t. CPU, I/O, and time) in ProDB, the evaluation focus on this part. Let ω denote the cost for an ORAM client to retrieve one single block in the path from the ORAM server, which also includes the overhead incurred by TEE for public-key attestation and function calls. Therefore the cost of one ORAM round is $LZ\omega$, where Z is the bucket size and L is the depth of the tree. If there is only one ORAM server, then the overall ORAM cost for a given query is the sum of costs for all ORAM rounds to access all blocks for this query, i.e., $R_d LZ\omega$, where R_d is the number of ORAM rounds. As such, the objective of ProDB is to minimize R_d while still satisfying memory-secure DBMS.

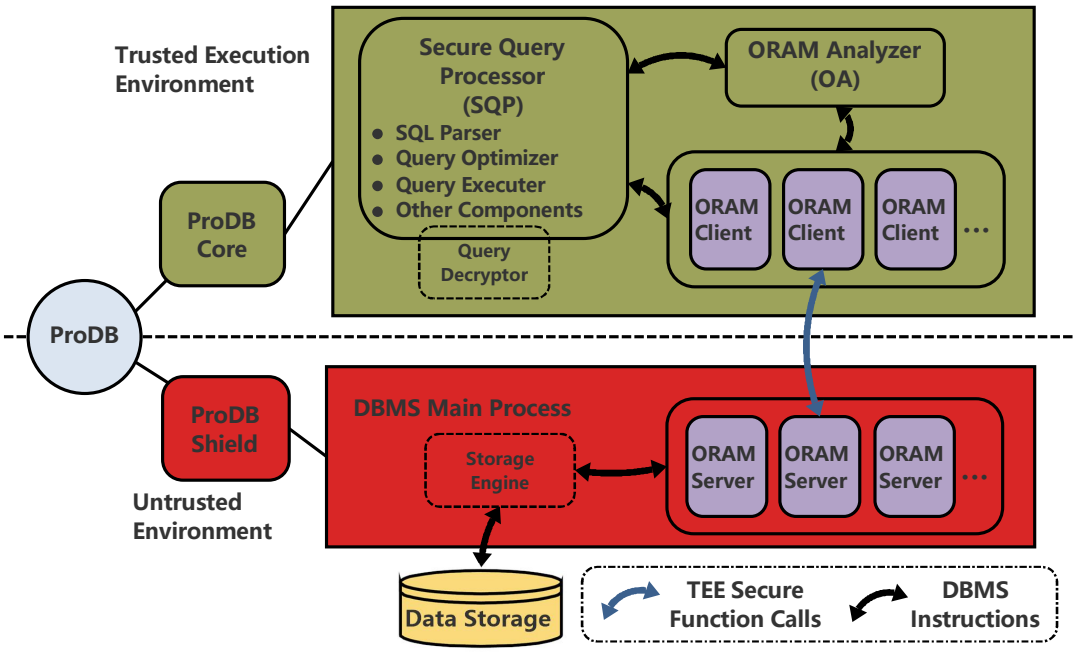


Figure 3.3: ProDB Two-Tier Design: Core and Shield

3.3.2 A Two-Tier Design

As shown in Figure 3.3,[2] The proposed adaptation is composed of 2 tiers, namely the ProDB Core and ProDB Shield, which run inside and outside a TEE, respectively.

ProDB Core

ProDB Core consists of the SQL Decryptor, Secure Query Processor, ORAM analyzer, and ORAM Clients.

SQL Decryptor. When a SQL query arrives at the ProDB Core, it is in an encrypted form. The SQL decryptor will decrypt it in the TEE so that the untrusted hosting

[2]To clarify, here do not depict those components in a conventional RDBMS that are not re-designed in ProDB, such as the concurrency control unit and database logging.

environment cannot learn about the query.

Secure Query Processor (SQP). SQP is the component to process queries in ProDB. It inherits those software modules from a conventional RDBMS, such as SQL parser, query optimizer, and query executer. The only difference is the I/O access. In a conventional RDBMS, the query executer directly sends I/O instruction to load data blocks from the main memory or the disk. In ProDB, the SQP loads data blocks from ORAM clients, which interact with ORAM servers outside TEE. The latter loads data blocks from storage engine if they are missing from the main memory.

ORAM Analyzer (OA). ProDB leverages SaP ORAM to securely access untrusted memory for data blocks without leaking access pattern. SaP ORAM, further discussed in Section 3.4, is a historical SQL-aware ORAM protocol. That is, how tables are mapped to ORAM instances is determined by the historical SQL workload. In a nutshell, tables that are frequently accessed together should be allocated to the same ORAM instance so that a single round of ORAM access can retrieve multiple blocks from these tables. In the ORAM Analyzer, historical statistics on SQL queries are retained for each table T_i as two pieces of meta-data. The first, denoted by $J_{i,j}$, is the number of joint ORAM accesses with another table T_j within the same batch of queries (e.g. transactions). The second, denoted by S_i, is the number of non-parallel ORAM accesses on this table T_i. These sets of meta-data are employed to find the optimal pairing plan of tables (as detailed in Section 3.4.2).

ORAM Clients During query execution, SQP accesses data blocks in the untrusted

environment through ORAM clients. There is a one-to-one mapping between an ORAM client and an ORAM server. Each ORAM server is initialized by its corresponding client.

ProDB Shield

As shown in Figure 3.3, ProDB Shield is the part of ProDB that works in the untrusted memory. It consists of the DBMS Main Process and ORAM Servers.

DBMS Main Process. It serves as the entry point of queries and the carrier of TEE program, i.e., the ProDB core. Besides these purposes, to function as a DBMS, it includes other conventional DBMS components such as storage engine and connection manager.

ORAM Servers. An ORAM server instance is organized in a tree structure (see Section 3.4) and can accommodate the data blocks of more than one table. As instructed by SQP, the corresponding ORAM client sends read/write requests of data blocks to the ORAM server, who then safely writes back updated data blocks to disks through the storage engine under an ORAM-to-Disk mechanism introduced in Section 3.4.1.

3.3.3 Overall ProDB Work-flow

ProDB Core and ProDB Shield collaborate with each other to provide memory-secure query operations. Figure 3.4 shows the main steps of query processing, among which inflow and outflow instructions of TEE are transmitted through secure channel by issuing

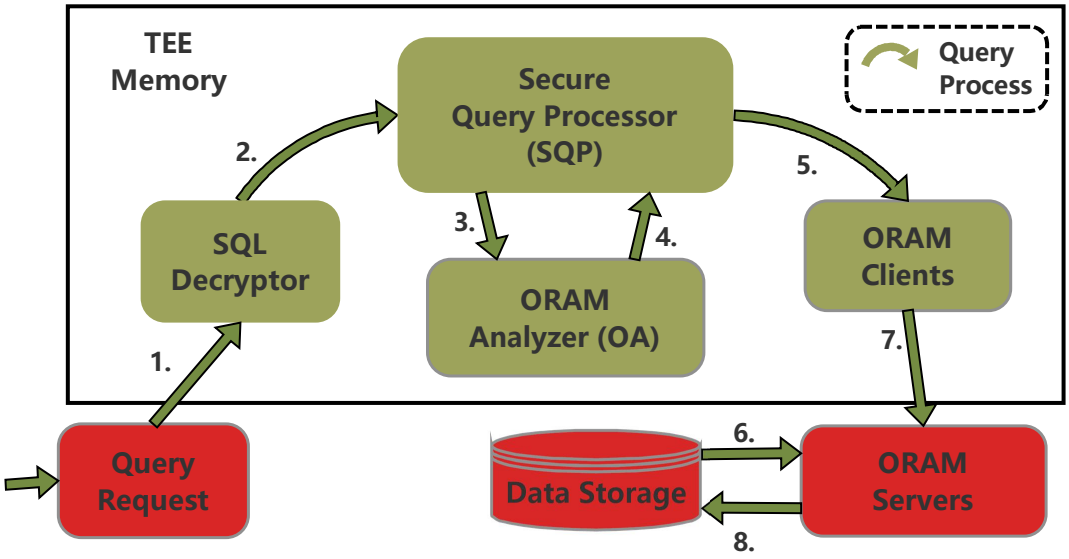


Figure 3.4: ProDB Query Process

secure function calls of TEE (e.g., ECALL and OCALL in Intel SGX).

Algorithm 1 shows a detailed workflow of processing a batch of query requests T. As defined in Table 2, a secure query request is encrypted by symmetric encryption E with key K_u, and a MAC with key K_h is appended to ensure data integrity. So the complete request is $T = H_{K_h}(E_{K_u}(S))||E_{K_u}(S)$. After being validated on hash value (line 3), SQL decryptor decrypts the SQL strings S contained in the workload (line 6), then forwards it to Secure Query Processor (SQP, line 7). ORAM Analyzer exports SaP ORAM table pairing plan (see Section 3.4.2) to guide the initialization of ORAM instances (line 8). The construction steps of the optimal table pairing plan will be given later in Algorithm 2. Subsequently, each SQL string s is parsed as s^* and optimized to generate execution plan qp (line 10-11). Meanwhile, the historical query records are updated. If the associated pages of qp do not exist in ORAM instances, SQP notifies

store engine to load them from disk (line 12). The query engine then executes the rest non-I/O tasks in qp and wraps the plaintext result set res (line 13). When all the queries are handled, SQP invokes probabilistic lazy persistence (PLP) (see Section 3.4.1) to perform persistence jobs and flush related dirty pages into disk storage (line 15). SQP finally re-encrypts and hashes result set by E_{K_u} and H_{K_h}, and sends it back to the query initiator (line 16).

Algorithm 1 ProDB Query Process

Input: $T = H_{K_h}(E_{K_u}(S))||E_{K_u}(S)$
Output: Result set: $Rs = H_{K_h}(E_{K_u}(res))||E_{K_u}(res)$
1: Initiate return value: $Rs \leftarrow \emptyset$
2: $H \leftarrow H_{K_h}(E_{K_u}(S)))$, $Q \leftarrow E_{K_u}(S)$
3: **if** $H_{K_h}(Q)$!= H **then**
4: **Return**
5: **else**
6: $S = D_{K_u}(Q)$
7: SQP Invoke: $doQuery(S)$
8: $buildOram(OA\_getPairing())$
9: **for each**: SQL string $s \in S$ **do**
10: $s^* = parse(s)$
11: $qp = opt(s^*)$ and $OA\_update(s^*)$
12: Invoke $oram.loadData(qp)$
13: $res \leftarrow res \bigcup SQP\_execQuery(qp, oram)$
14: **end for**
15: SQP Invoke: $persistence()$
16: **Return** $Rs = H_{K_h}(E_{K_u}(res))||E_{K_u}(res)$
17: **end if**

3.3.4 Limitations of ProDB

In most database management systems, persistent data are organized and stored in certain file formats, such as heap files and sequential files. These files exploit the charac-

teristics of a disk and thus optimize the I/O performance. However, when ProDB loads these blocks, the original data organization in these files is lost due to the shuffling by the ORAM server. By design, ProDB cannot address this issue for security purposes. Furthermore, currently ProDB does not consider those advanced DBMS features, such as concurrency control, rollback mechanism and database logging.

3.4 SaP ORAM

In this section, the features of access pattern obfuscation protocol, SaP ORAM, are demonstrated. The proposed scheme addresses two efficiency gaps between existing non-recursive Path ORAM and practical use in DBMS, that said, the privacy problem in persistence and the compatibility of processing SQL queries. In what follows, I discuss them separately.

3.4.1 Probabilistic Lazy Persistence

In a conventional RDBMS, the storage engine constantly writes modified dirty pages in the buffer back to disks. Since the Path ORAM protocol needs to re-encrypt the whole path of blocks back to the ORAM server, all these blocks are essentially modified as shown in Figure 3.5(a). This will significantly degrade the I/O performance of ProDB. The **probabilistic lazy persistence** mechanism of SaP ORAM addresses this problem by making the following two changes on the Path ORAM client.

Tagged Position Map. The 'tag' is appended as a new attribute to the original Path ORAM position map. The bijective 'address-path' mapping is thus expanded to a triple mapping as 'tag-address-path'. "Tag" is used to identify the underlying storage medium (e.g., a YD file of MySQL-ISAM or a filegroup of SQL Server) for the block. For new data blocks, a special tag *empty* is used. Since this attribute is only used in the ORAM client that runs on a TEE, it generates no security issues to ProDB.

Update List. An Update List is maintained in each SaP ORAM client. It stores those blocks that are updated by SQL query rather than by the ORAM re-encryption. When an ORAM client needs to write persistent media (e.g., after a transaction), the persistence procedure is invoked. For each block in the list, it uses the tag in the position map to locate the storage medium for that block and requests the storage engine to perform the write.

However, as shown in Figure 3.5(b), if an ORAM client only sends the real updated blocks for persistence, not those by the ORAM re-encryption (i.e., not logically modified), an adversary can learn from store engine about these blocks and infer the query. To address this, a probabilistic dirty-block-generation procedure is provided to fabricate an expanded and obfuscated update list for persistence. As shown in Figure 3.5(c), the procedure randomly adds blocks that are modified by re-encryption from the position map into the update list. Notice that, the 'dummy updated blocks' are randomly picked from all the pages that have been loaded into ORAM. Therefore, the obfuscation remains to be independent with distribution of real updates over pages. It is obvious that

the more re-encrypted blocks the more effective this obfuscation becomes. However, it is at the cost of degraded I/O performance. To model such cost, let H denote the number of real updated blocks and H' denote the number of re-encrypted blocks. As they are randomly selected from an ORAM tree with depth L, the probability that an adversary succeeds with a random guess of all real updated blocks is

$$prob_b = 2^{-(H+H')}. \tag{3.1}$$

I/O in the storage engine typically use pages rather than blocks. To convert the above to pages, let the page size P be a multiple of the block size B. As such, the total number of pages in the ORAM tree is $g = \frac{2^L \cdot B \cdot Z}{P}$, where Z is the bucket size for ORAM trees. Further, assume that the real updates and dummy updates are both evenly distributed among all pages, thus we obtain $g_R = g \cdot (1 - (\frac{g-1}{g})^H)$ as the expected number of real dirty pages in the Updated List and $g_D = g \cdot (1 - (\frac{g-1}{g})^{H+H'})$ as the expected number of dirty pages after adding H' dummy blocks. Then we could derive the probability of identifying the real dirty pages through

$$prob_p = 2^{g \cdot (\frac{g-1}{g}^{(H+H')} - 1)}. \tag{3.2}$$

The following illustrates a real-life example of this probability.

Example 1. *Under a practical setting that* $P = 4B = 16Kb$, $L = 7$, $Z = 4$, *and the expected overhead of lazy persistence is restrained to* $\leq 1x$, $2x$ *number of dirty*

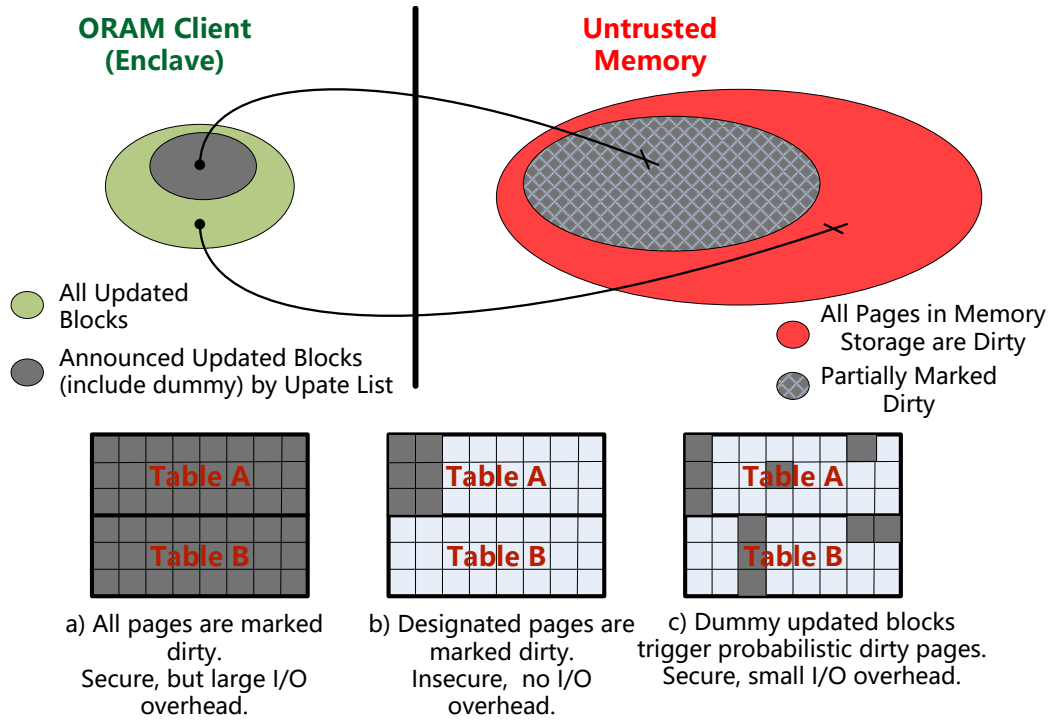


Figure 3.5: Probabilistic Lazy Persistence

pages, i.e. $g_D \leq 2g_R, \quad g_D \leq 3g_R$. *Under such constrains, let* $H = 30, \quad 40$ *to derive corresponding* $\max H' = 66, \quad 135$ *and* $\max H' = 102, \quad 207$ *respectively for 1x and 2x additional cost. The probability that an adversary knowing the distribution of updated blocks to dirty pages succeeds with a random guess of the actual updated pages suffices* $prob_p \leq 2^{-48.4}, \quad \leq 2^{-59.9}$ *for 1x additional cost and* $prob_p \leq 2^{-75.4}, \quad \leq 2^{-91.9}$ *for 2x additional cost.*

This shows that a moderate number of dummy blocks (e.g., the same number as the real dirty blocks) is sufficient for a secure obfuscation.

3.4.2 SQL-Aware Path Sharing

Path ORAM and other enhanced tree-based ORAMs still incur high computational and I/O costs for a practical DBMS as ProDB. While it is an active research field to design new ORAM schemes that optimize the performance of single block access, in this work, an orthogonal perspective by reducing the number of ORAM rounds is initially proposed.

In this subsection, with the core idea of fetching more than one block from the tree path during one ORAM round, the **SQL-aware path sharing** mechanism is proposed. The ORAM Analyzer keeps track of those blocks that are frequently accessed together and places as many of them in the same ORAM tree path as possible. Hosted in TEE, it collects the access history of each table in the form N_i, $J_{i,j}$, S_i, where N_i is the number of blocks in table T_i, and $J_{i,j}$ is the number of co-occurrences of block accesses to table T_i and T_j in a batch of queries, while S_i is number of access times to table T_i counted exclusively (as defined in 3.3.2). Based on this history, data blocks from λ tables are merged into a single ORAM instance. In the ideal case when all data blocks in the path are requested, the amortized ORAM cost can be reduced by up to $(\lambda - 1)\omega$.

For ease of presentation, in what follows I only discuss the condition that $\lambda = 2$, i.e., each ORAM instance holds up to 2 tables A and B, each of which accounts for s and t blocks, respectively. Let M and N $(M \geq N)$ be their numbers of block accesses in a query workload. If table A and table B have their individual ORAM instances, the depths of ORAM trees are approximated as $\log \frac{s}{Z}$ and $\log \frac{t}{Z}$, respectively. As such, the

ORAM cost C using plain Path ORAM is:

$$C = Z(M \log \frac{s}{Z} + N \log \frac{t}{Z})\omega \tag{3.3}$$

Recall that ω is the unit cost to access one block and Z is the bucket size. The security proof of this path sharing technique is presented in Section 3.5.1.

If SaP ORAM is adopted instead, A and B are stored in one ORAM instance. When a batch of SQL queries are executed, SaP ORAM retrieve paths that contain multiple target blocks from both A and B. Let K ($M \leq K \leq M+N$) denote the number of SaP ORAM access rounds. Then the ORAM cost C' using SaP ORAM is:

$$C' = ZK(\log \frac{s+t}{Z})\omega \tag{3.4}$$

Then I define *efficiency gain* ψ as the difference on ORAM costs between SaP ORAM and plain Path ORAM for the query workload, i.e., $\psi = C - C'$.

Definition 2. *(Efficiency Gain ψ):*

$$\psi = C - C' = Z\omega \cdot \log((\frac{s^M t^N}{Z^{M+N}}) \cdot (\frac{Z}{s+t})^K)$$

The following theorem depicts that to maximize ψ is equivalent to minimize K.

Theorem 1. *(K Negative Correlation): Given the number of tuples s and t, bucket size Z and the number of access rounds M,N, maximizing ψ is equivalent to minimizing K.*

Proof. The key observation is that $s + t > Z$ always holds for Path ORAM and its variants, because a rather small Z (e.g.,$Z = 3$, 4 or 5) is usually chosen[96]. As shown in Definition 2, given that $0 < \frac{Z}{s+t} < 1$, the expression of ψ suffices monotone increasing function as K decreases. □

To minimize K, it always expects to find a shared path in ORAM tree holding target blocks of both two tables, so that it can access them in one pass. Now I give the definition of such a path as follows.

Definition 3. *(Gaining Path): In processing a query with M block accesses to table A and N blocks accesses to table B, a tree path is called a gaining path if it contains at least* 1 *target block of table A and* 1 *target block of table B simultaneously.*

Each gaining path reduces K by 1. The ORAM client looks up the position map and picks a gaining path to retrieve if there exists one. Otherwise, a non-gaining path is chosen, which only reduces M or N by one. The following theorem shows that the best efficiency gain can be achieved when M and N are the closest.

Theorem 2. *If the total size of two tables, $M+N$, is fixed, a smaller intra-table skewness $\Upsilon = |M - N|$ will lead to a higher efficiency gain ψ.*

Proof. Let M^* and N^* be the remaining number of target blocks not accessed yet. When the ORAM client picks a path to retrieve for the next round, the probability that it can find a gaining path is $Pr = \frac{Min(M^*,N^*)\cdot Max(M^*,N^*)\cdot 2Z}{s+t}$. Since $Min(M^*, N^*) \cdot Max(M^*, N^*) = \frac{(M^*+N^*)^2-(|M^*-N^*|)^2}{4}$, and M^*+N^* is a constant value, Pr must increase

as $|M^* - N^*|$ decreases. To prove this, the relation between $|M^* - N^*|$ and $\Upsilon = |M - N|$ is shown as below. After each retrieval, the change on $|M^* - N^*|$ satisfies a stochastic walk as:

$$|M^* - N^*|' = \begin{cases} |M^* - N^*| + 1 & \text{with prob. } \frac{1-Pr}{2} \\ |M^* - N^*| & \text{with prob. } Pr \\ |M^* - N^*| - 1 & \text{with prob. } \frac{1-Pr}{2} \end{cases}$$

The initial values are $M_0^* = M$ and $N_0^* = N$, so the expectation of $|M^* - N^*|$ after a finite random walk is $E(|M^* - N^*|) = |M - N| = \Upsilon$. As such, I prove that a smaller $|M - N|$ increases the probability of finding a gaining path in each round of retrieval. Therefore, the efficiency gain of the transaction ψ will also increase. □

Now we can generalize the above theorem to the whole set of tables in a database. That is, we can pair up tables whose cumulated block accesses in all historical queries are close (i.e. small Υ under fixed $M + N$). The objective is to achieve the highest overall gain as defined below:

Definition 4. *(Overall Gain) The overall efficiency gain* $\Psi = \sum_{i=1}^{x} \psi_i$ *is the aggregated ORAM cost saving for whole batch of queries. Here* ψ_i *denotes the efficiency gain for* i*-th table pair (there are* x *pairs in total). Essentially,* Ψ *is the expression of yield on performance metric* $R_d \cdot \omega$ *for given set of query requests.*

Nonetheless, the above paring scheme does not take $M + N$ into account. Therefore,

a combined metric, namely, the evaluation factor of two tables, is used to determine if two tables are suitable for paring.

Definition 5. *(Evaluation Factor) The evaluation factor α_{ij} of two tables i and j is calculated as $\alpha_{ij} = \frac{{J_{i,j}}^2}{S_i+S_j}$. Here S_i, S_j and $J_{i,j}$ are the statistics of the historical queries defined in Section 3.3.2. Specifically, S_i, S_j indicate the number of non-parallel accesses to single table and $J_{i,j}$ is the number of joint accesses to table T_i, T_j within the same batch.*

The higher the evaluation factor, the more gaining paths the two tables can generate, and thus the more suitable to pair them. However, the optimal table pairing plan (OTPP) needs to enumerate all combinations of table pairs in a database. To efficiently model this problem, an analogy is conducted to convert the problem to the maximum weighted matching problem in graph. In such model, vertex set V is used to denote tables and the edge set E is used to indicate the joint access between two tables. Then the OTPP problem is equivalent to finding the maximum weighted matching on this graph $\overrightarrow{G}(V, E)$, where the weight of edge (i, j) is the evaluation factor α_{ij} of two tables. Unfortunately, state-of-the-art solutions to the maximum weighted matching problem in a general graph, such as [30] and [36], still require quadratic complexity on $|V|$. In what follows, an efficient heuristic-based approximate algorithm is provided to select a suboptimal table pairing plan (STPP).

The entire STPP procedure is shown in Algorithm 2. The greedy algorithm sorts the evaluation factors in descending order (line 3). It then traverses them in this order and

Algorithm 2 Suboptimal Table Pairing Plan (STPP)

Input: All evaluation factors: $\alpha_{ij}, \forall i, j \in U$
Input: Unpaired table set: S_V
Output: Bi-table Pair set: *sttp*

$sttp \leftarrow \emptyset$
Initialize vertices set with table set U : $S_V \leftarrow U$
Sort α_{ij} in descending order
for each: α_{ij} from top **do**
  if i and $j \in S_V$ **then**
    if $\alpha_{ij} \neq 0$ **then**
      Put pair $i \leftrightarrow j$ into *sttp*
    else
      Put mono-table pair i and j into *sttp*
    end if
    Remove i, j from S_V
  else
    if i or $j \in S_V$ **then**
      Put mono-table pair i or j into *sttp*
    else
      Continue.
    end if
    Remove i or j from S_V
  end if
end for
Return *sttp*

repeatedly adds the two vertices with the highest evaluation factor as a pair to *sttp* (lines 4-7). The corresponding two tables are then removed from the unpaired table set S_V (line 11). At the end of an iteration, all isolated tables (i.e. vertices) are regarded as mono-table pairs each of which will be allocated an ORAM instance. (lines 9 and 14). The iteration terminates when S_V becomes empty, and *sttp* is returned (line 21).

The evaluation on how the amount of database tables and the randomness of meta-data affect the computational cost of STPP are shown in Section 3.6.4. Furthermore, the relation between the independent cost in STPP and the overall cost in handling a query request is also given in Section 3.6.4.

3.5 Security Analysis

The utilization of ORAM guarantees the memory access pattern of a data block is indistinguishable from others. Therefore queries executed in memory disclose no discriminative information to adversaries. Therefore, by introducing ORAM in untrusted memory, the second condition in security model hold for ProDB. In this section, the optimizations in SaP ORAM are theoretically proved not to compromise the security functionality of ORAM. A proof on the security of path sharing in SaP ORAM is given first followed by a discussion on the security of using multiple ORAM instances for a single database.

3.5.1 Security of Path Sharing

Path sharing allows a SaP ORAM client to fetch more than one target block in a single ORAM access. The same style of security analysis as in Path ORAM [96] is adopted to prove that this feature does not weaken the security level, so that adopting SaP ORAM in ProDB does not compromise the security model of secure-DBMS in Definition 1. Recall in Definition 1, the sequential access pattern with length m seen by the ORAM server is denoted by $\xi = (p_m, p_{m-1}, ..., p_1)$, where p_j is the path index in the ORAM tree of depth L. The randomness in re-mapping the paths guarantees all block accesses are statistically independent. Or formally, the probability of distinguishing the access sequence of ξ from that of another query is $Pr(\xi)_{Path} = (\frac{1}{2^L})^m$. In what follows, SaP ORAM is proved to hold the same or even lower collision probability.

Theorem 3. *In SaP ORAM, the path sharing mechanism ensures that* $\forall \xi$, $Pr(\xi)_{SaP} \leq Pr(\xi)_{Path}$ *always holds.*

Proof sketch. Let λ denote the number of tables that are merged into one ORAM instance and L_c denote the depth of the combined ORAM tree. Recall that in SaP ORAM, we greedily retrieve the path that can fetch most ($\leq \lambda$) target blocks. The path selection is executed inside an enclave space after the re-mapping step of previous retrieval round. As such, the path retrieval maintains the same randomness as Path ORAM. That is, the probability of distinguishing the access pattern ξ in SaP ORAM satisfies $Pr(\xi)_{SaP} = (\frac{1}{2^{L_c}})^m$. Further, due to the fixed bucket size Z, L_c must be larger

than that of any ORAM instance of a single table to accommodate extra blocks, i.e. $L_c \geq L$. Therefore, we can guarantee that $Pr(S_i)_{SaP} \leq Pr(S_i)_{Path}$.

3.5.2 Inter-table Security

Intuitively, there should be only one ORAM instance that holds all tables. However, this is infeasible in practice for the following reasons. First, since a DBMS needs to accommodate data into different storage media and locations (e.g., local disks, iSCSI, and NFS), the single-ORAM-instance design cannot capture the difference in their I/O characteristics. Second, the access frequencies of various tables are drastically different and a single-ORAM-instance design cannot optimize the I/O performance for "hot" tables. As such, the design of allowing each ORAM instance to accommodate a set of tables is employed. However, as the mapping of tables to these ORAM instances is not anonymous to ORAM servers, the table-level access pattern may still leak sensitive information, though in a coarser scale. In an extreme example, if each table is mapped to one ORAM instance, the alternate access of two ORAM servers can imply a nest-loop join between two tables.

SaP ORAM protocol partially alleviates this issue by the path sharing feature. Since data blocks of multiple tables are retrieved simultaneously through single path access to an ORAM server, their original access sequences to these tables can no longer be inferred. To illustrate this, the above extreme example is used. Let $Seq := ..., A_1(A, path_i, addr_1, op)$, $A_2(B, path_j, addr_2, op), ...$ denote the original retrieval pattern on the two ORAM in-

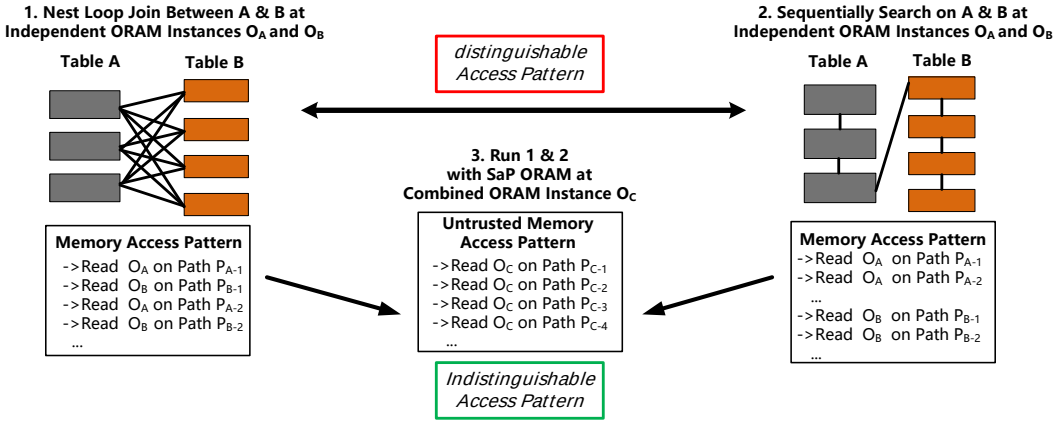


Figure 3.6: Access Pattern of Inter-table Queries

stances of a nest-loop join query on tables A and B. The pattern clearly shows the alternate access of tables A and B, which can be inferred by an adversary as a nest-loop join with high confidence. In SaP ORAM, since a batch of block accesses to tables A or B can be achieved by a single ORAM access, such an alternate table access pattern can no longer be observed. Furthermore, as shown in Figure 3.6, the same table access pattern can also originate from a sequential scan of tables A and B. Therefore, the confidence of the adversary to infer the query as a nest-loop join is significantly lowered.

3.6 Experimental Results

A prototype ProDB is implemented by refactoring those source codes with block access instructions (such as “ha_rnd_next()” in “execute_sqlcom_select()”) in MySQL. The ProDB core (i.e., client side of SaP ORAM) is written in C++, compiled to a DLL file,

and called by MySQL main program "mysqld" in "lex-unit-execute()" after the SQL string is parsed to LEX tree. The rest query execution procedures are removed (e.g. "unit-optimize" for optimizing the query and transaction control in "binlog") to simplify the process so that the experimental results will stay focus on the block access. In this section, I evaluate the performance of both ProDB and SaP ORAM through experiments. I first demonstrate the effectiveness of SaP ORAM to hide memory access pattern. Then I conduct a comparative study on the efficiency of SaP ORAM and Path ORAM. Finally, the overall performance of ProDB is depicted under various real-life TPC-H query workloads.

3.6.1 Effectiveness of SaP ORAM for Memory Access Pattern Hiding

I demonstrate the result of memory access pattern hiding against "curious" VM hypervisor and OS administrator for a set of different type TPC-H (1GB workload) SQL queries (Q4,Q17,Q21,RF1,RF2[3]). Maintaining the same settings as applied in the preliminary test shown in Section 1, the vSphere ESXi version 6.5 is deployed as hypervisor on host machine which has Intel i7 6700 CPU with 64 GB memory. Its VM runs a Win 10 Pro system with 2 CPU cores, 16GB memory, and MySQL 6.5 database.[4] To exclude the

[3]Other TPC-H queries are excluded because they contain compound operations, such as 'group by', which are beyond the current scope of ProDB.

[4]Since Intel SGX can only be applied to VM guests through KVM patches [53] at the time of writing, the set of experiments in this subsection runs on a plaintext database without affecting the result of access pattern hiding.

Table 3: Memory Access Patterns for TPC-H Queries - Direct Access

Query	**Mem. Reads(K)**	**Mem. Writes(K)**	**Peak Usage(MB)**
$Q4$	248	< 1	247
$Q17$	719	< 1	260
$Q21$	$1,295$	< 1	1007
$RF1$	68	34.52	159
$RF2$	101	30.66	159

impact of disk I/O on the query performance, the MySQL store engine is enforced to cache all TPC-H query workload in the memory by setting "innodb_buffer_pool_size" to 1GB. I use hypervisor tools and MySQL monitoring instructions to record the values of 3 memory activity features, namely, # of read/write requests to MySQL process buffer and the peak working memory usage in this VM. Tables 3 and 4 show the average feature values over 100 independent runs without and with SaP ORAM, respectively. We can observe that the values in Table 3 are more diverse and distinguishable, especially for SQL queries with updates (RF1, RF2). Then I launch side-channel inference attack by assuming an adversary has access to the same statistics as in these two tables and is monitoring the memory access of an unknown TPC-H query. Specifically, the adversary uses 80 runs as training data to build a naive-Bayes classifier over the 3 features, and then uses 20 runs as testing data to reidentify their query IDs in TPC-H. The success rates of this attack for each query without and with SaP ORAM are shown in Figure 3.7. From the observation, we can see this inference attack is very effective (65%+) without SaP ORAM. On the other hand, SaP ORAM effectively restrains the adversary from performing significantly better than a random guess, because it manages to narrow down the difference between queries in terms of memory reads, writes, and peak usage.

Table 4: Memory Access Patterns for TPC-H Queries - via SaP ORAM

Query	**Mem. Reads(K)**	**Mem. Writes(K)**	**Peak Usage(MB)**
$Q4$	$2,780$	$2,451$	893
$Q17$	$10,017$	$11,634$	1012
$Q21$	$15,363$	$14,268$	1022
$RF1$	$1,057$	824	897
$RF2$	$2,201$	$2,172$	997

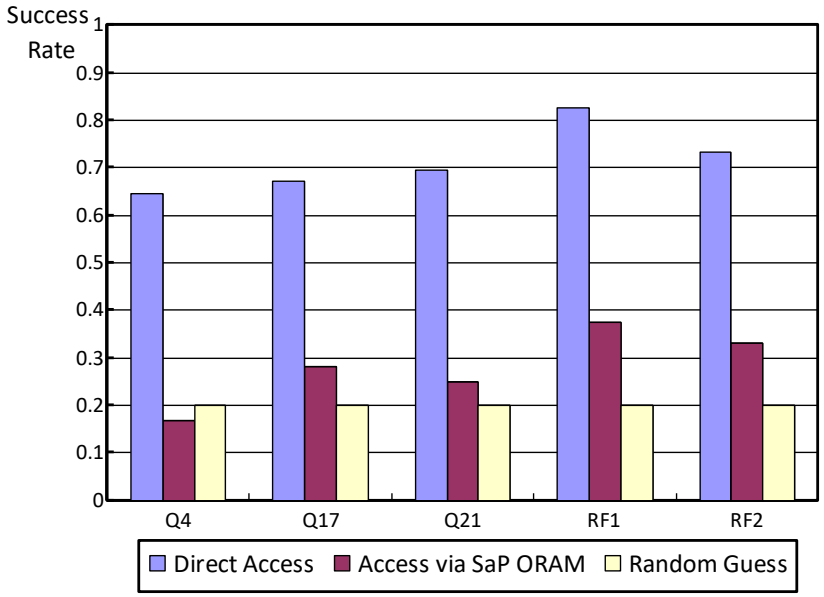


Figure 3.7: Success Rates of Inference Attack - Direct Access vs. via SaP ORAM

3.6.2 SaP ORAM Performance

In this subsection, the SaP ORAM performance is evaluated against Path ORAM in the presence of Intel SGX. The experiments are conducted on an Intel i7 6700HQ CPU with 16 GB RAM running Windows 10. The blocks in the $4Kb$ data array are sequentially accessed using three distinct Intel SGX Enclave programs sealed inside encrypted DLL files, namely, without ORAM, with Path ORAM, and with SaP ORAM. The latter two DLL files implement adopt the same ORAM parameter settings as $L = 8, Z = 5$ and $B = 4Kb$.

Each DLL file is executed for up to $1,000$ block retrievals and the amortized elapsed time of invoking the enclave program to access a block are measured (i.e., from the moment when the program enters the enclave to the moment when the return buffer from the enclave is received). The elapsed time under each DLL file is plotted in Figure 3.8. I observe that as more blocks are accessed, SaP ORAM achieves a steady 30% performance gain over Path ORAM mainly due to path sharing. On the other hand, compared with the direct access without any ORAM, SaP ORAM only introduces about 100% overhead to the elapsed time when a large number of blocks need to be accessed. Considering its security yield, this performance loss is acceptable when the security concern dominates the system design.

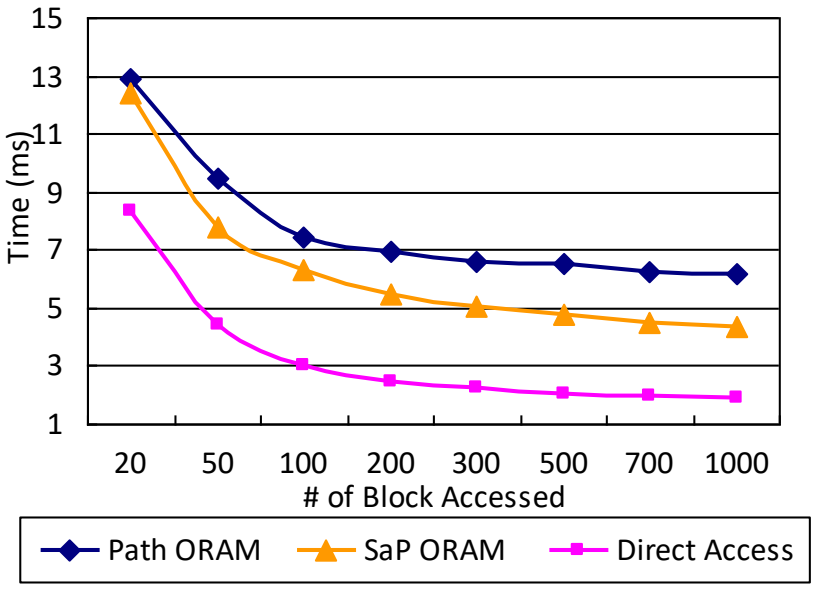


Figure 3.8: Amortized Elapsed Time - SaP ORAM vs. Path ORAM vs. Direct Access

3.6.3 ProDB Performance Under TPC-H Workload

This subsection includes the study on parameters of both path sharing and probabilistic lazy persistence mechanism reflecting on the ProDB query and persistence performance under various real-life TPC-H workloads.

Path Sharing

In this part, the overall performance of ProDB is evaluated under TPC-H 1G dataset in terms of amortized query elapsed time. The experiments focus on two TPC-H queries, namely, Q12 and Q14. Both are two-table join queries[5]. For Q12, the tables "orders"

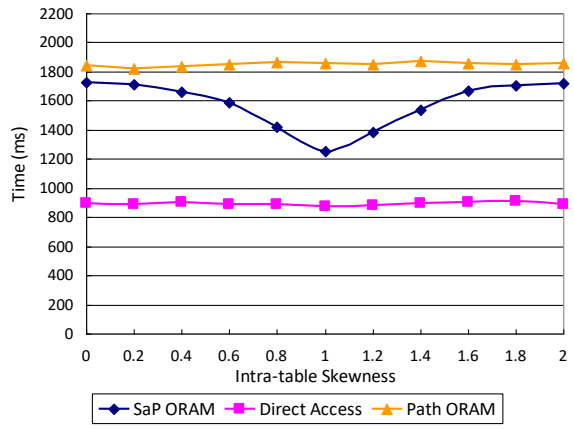
[5]Except for Q19, the other queries in TPC-H workload are not suitable to evaluate the pure effect of SQL-aware path sharing as they are associated with either 1 or more than 2 tables. Q19 is less representative in evaluating with different intra- and inter-table skewness. Therefore it is only used in the evaluation STTP performance

and "lineitem" are assigned as a pair and merge their blocks in one ORAM tree; for Q14, tables "lineitem" and "part" are designated as a pair and their associated blocks are merged into one ORAM tree. All rows in tables are fit in blocks (set $B = 4Kb$) as full as possible. The blocks are arranged in the ORAM tree structure for executing SaP ORAM accesses and in linear structure (Array) for direct non-ORAM accesses.

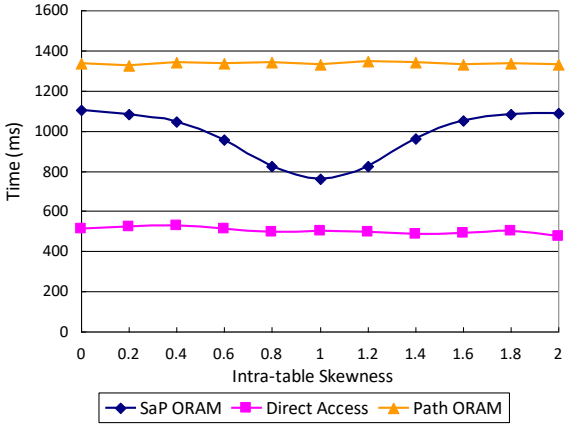
The 2 key internal parameters of path sharing that may affect the performance of ProDB are studied. The first is the intra-table skewness Υ. As defined in Theorem 2, Υ directly impacts on the efficiency gain ψ. In the experiment, Υ is normalized to $[0, 2]$ and $\Upsilon = 1$ indicates there is no distribution bias between the paired tables. The second parameter is the inter-table skewness Φ which evaluates the similarity between the pre-generated pairing plan and real query workload arrivals. Formally, the inter-table skewness Φ is the ratio of the largest number of block accesses on a pair of tables (A, B) to the total number of block accesses X. Recall that M and N denote the number of accesses to table A and B, respectively, so $\Phi = \frac{M+N}{X}$.

In the first experiment, Υ is varied by manipulating the scale of blocks in both tables. Figures 3.9(a) and 3.9(b) plot the amortized elapsed time of Q12 and Q14, respectively. I observe that when there is no distribution bias, i.e., $\Upsilon = 1$, ProDB achieves the lowest elapsed time. Nonetheless, even in the extreme cases where $\Upsilon = 0$ or 2, the elapsed time is increased by at most 40% and is still lower than Path ORAM. As such, it can be concluded that ProDB is robust with respect to Υ.

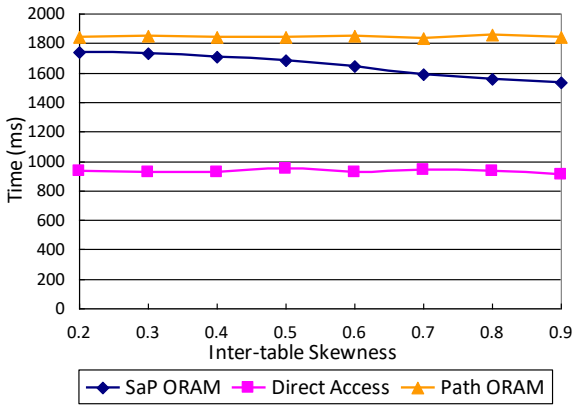
In the second experiment, the concentration ratio Φ is varied by adding random block



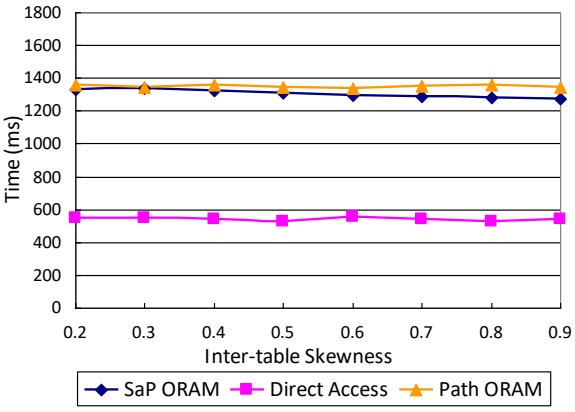
(a) Intra-table Skewness Υ - Q12



(b) Intra-table Skewness Υ - Q14



(c) Inter-table Skewness Φ - Q12



(d) Inter-table Skewness Φ - Q14

Figure 3.9: Impact of Parameters on Path Sharing Performance

accesses which are not related with the paired tables of Q12 and Q14, while maintaining the same total number of block accesses and the true Υ. Figures 3.9(c) and 3.9(d) plot the amortized elapsed time for Q12 and Q14, respectively. I observe that in both figures the elapsed time of SaP ORAM decreases linearly as Φ grows. Note that in Figure 3.9(d), SaP ORAM is close to Path ORAM because the true Υ of Q14 approaches 0, i.e., the number of blocks in table "lineitem" is much larger than that in table "part".

Probabilistic Lazy Persistence

This part demonstrates how the number of dummy updated blocks added affects the number of generated dirty pages for persistence. The experiment is performed on the prototype of ProDB, as previously introduced in Section 3.6. In prior, Innodb parameters are configured as "UNIV_PAGE_SIZE = 16Kb","UNIV_PAGE_SIZE_SHIFT = 14" and set ORAM block size $B = 4Kb$. In the experiment, 3 SQL statements are executed to update different requested rows[6] in table "orders" on their column "o_orderstatus" with TPC-H 1G database. By varying the number of dummy updated blocks (measured in the percentage of real updated blocks) in ProDB core, the number of modified (dirty) pages can collected through monitoring the Innodb buffer pool. As shown in Figure 3.10, the growth of dirty pages of same query fluctuates around linearity along with the increase of dummy updated blocks when the weight of dummy updates is small, while recedes to approximately sub-linear growth when the weight becomes large.

[6]The 3 SQL statements differ in their requesting monthly periods on column "o_orderdate" from 1994-07-01 to 1994-10-01.

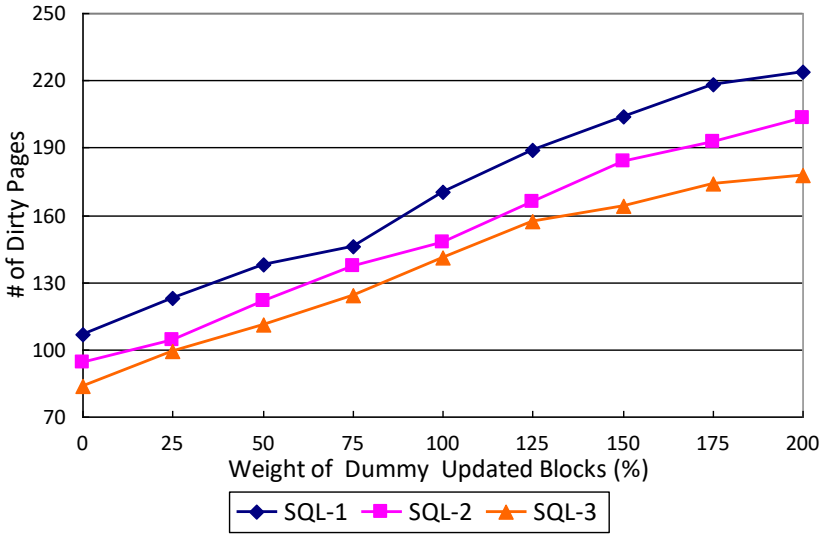
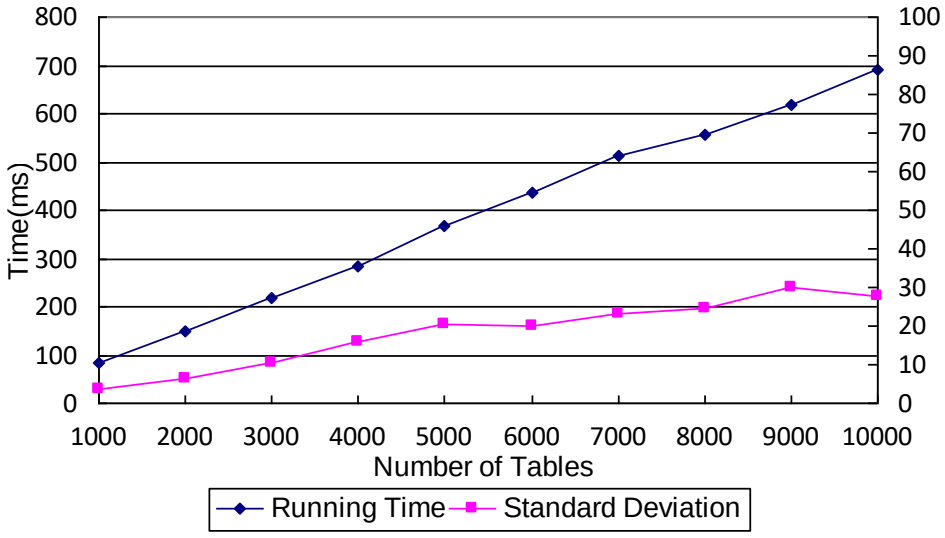


Figure 3.10: # of Dirty Pages Changing with # of Dummy Updates

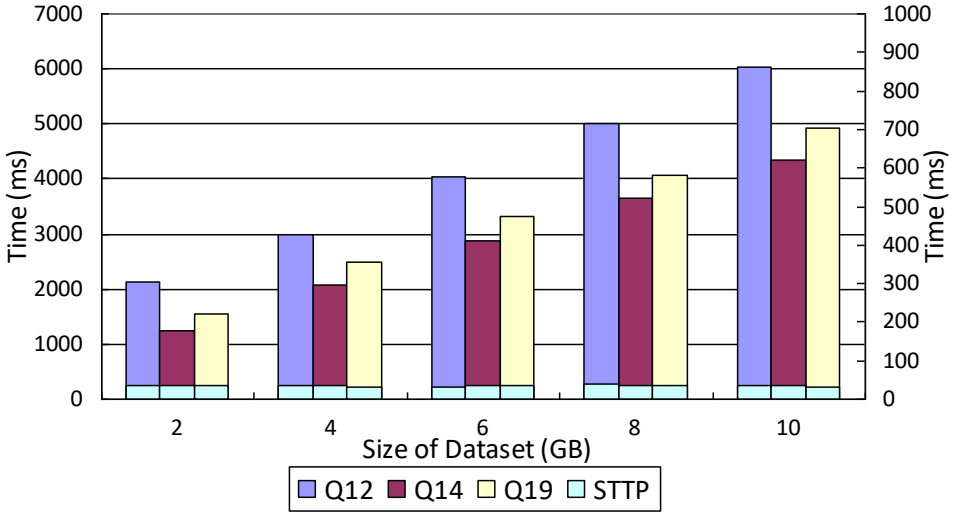
3.6.4 Suboptimal Table Pairing Plan

In this subsection, the efficiency of the suboptimal table pairing plan (STPP) is evaluated. In the first part, the CPU running time is measured with the number of tables varying from 1000 to 10000. The meta-data, i.e. S_i and $J_{i,j}$, are randomized in the experiment. Figure 3.11(a) plots the results. As each result is averaged by 200 trials, the standard deviations are plotted in the secondary y-axis in Figure 3.11(a). Based on the observation, the CPU running time is almost proportional to the number of tables even with the presence of random S_i and $J_{i,j}$.

In the second experiment, the TPC-H query workload is introduced to evaluate the time cost of STPP in the whole query processing. Figure 3.11(b) plots the STTP running time compared with the overall query elapsed time in Q12, Q14 and Q19 of TPC-H under various scales of TPC-H datasets (from 2G to 10G). It can be observed that STTP



(a) STPP Running Time Changing with # of Tables



(b) STPP Running Time vs. Query Elapse Time

Figure 3.11: STPP Running Time

consumes a fixed amount of CPU running time, and therefore the higher the total query elapsed time, the lower the percentage of STTP cost.

3.7 Summary

In this chapter, I propose ProDB as a minimal adaptation to a conventional database engine to practically improve the mutual trust between stakeholders in cloud databases. By leveraging hardware enclave, I propose SQL-aware Path ORAM (SaP ORAM) protocol to resolve access pattern attacks. It features probabilistic lazy persistence and path sharing that exploit the unique characteristics of database workloads. One key advantage of ProDB is that it does not need extra resources from the enclave and can co-exist with other optimizations on the database engine. As for full-fledged work in the future, I plan to enhance ProDB and SaP ORAM to support secure transaction management and concurrency processing in coordinating with DBMS.

Chapter 4

Privacy-preserving Fuzzy Keyword Search

4.1 Introduction

In the last chapter, the proposed DBMS adaptation applies TEE technology for memory access security and the Oblivious RAM for obfuscating the memory access pattern against inference attacks. However, the ORAM mechanism brings extra cost in data block retrievals, though the efficiency is improved in my design. To further mitigate this problem, in this chapter, I study a particular query, namely the fuzzy keyword search. The feature of fuzzy keyword search is to return the augmented results that are likely to be related to the inputs when the inputs are inaccurate or in the form of some typos. To this end, the purpose of studying this kind of query is to utilize this nature and

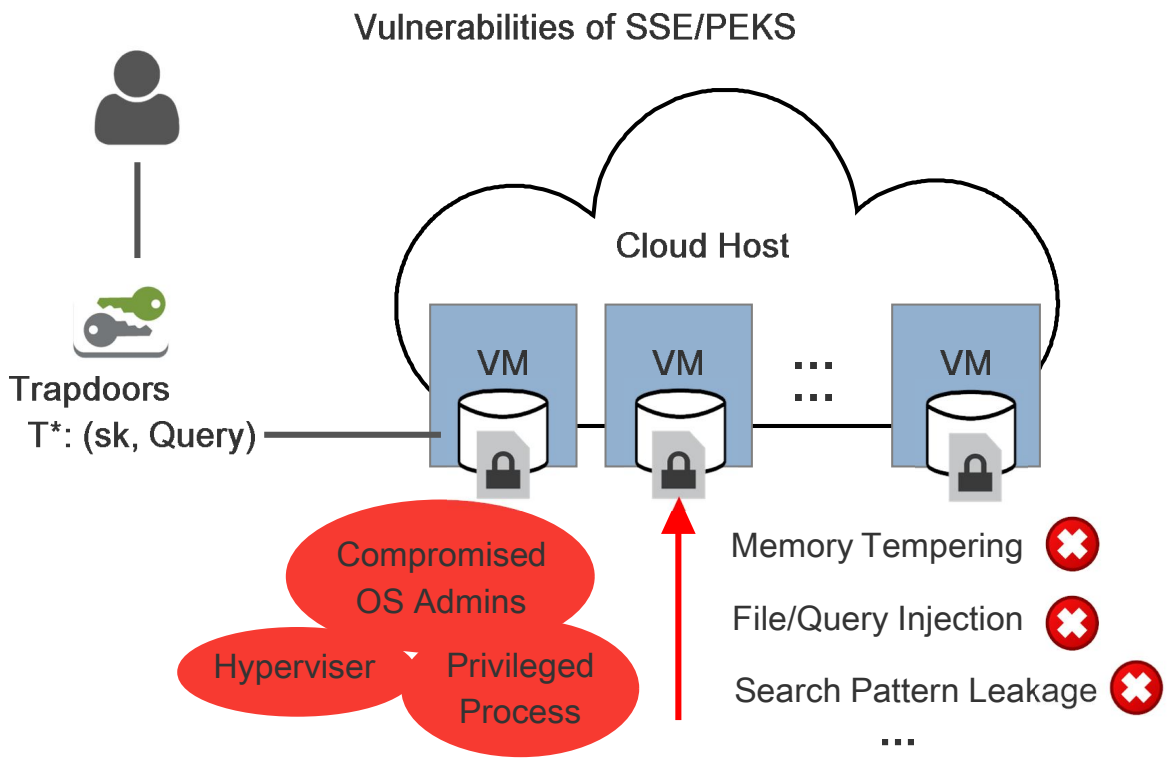


Figure 4.1: Unsolved Vulnerabilities of SSE/PEKS

combine it with an obfuscation scheme. Compared with an additional oblivious protocol, intuitively, this idea can reduce the overhead and maintain the protection on access patterns.

I propose a fast oblivious trend-aware keyword index with fuzzy search capability over the encrypted document data, named OKTI-F. To clarify the design goals of the proposed protocol, I decompose the research problem into three issues and introduce them as below.

- **Trustworthy multi-keyword fuzzy searching over encrypted data on cloud.**

  The fuzzy keyword searching offers convenience to data users in many aspects,

such as disposal of typos, uncertainty allowance in queries, and inspiring extra searching results. In many scenarios, such search services are privately delivered to authorized data users, e.g., company staff, organization members, and consortiums, through untrusted cloud platforms. Though massive efforts, represented by Searchable Symmetric Encryption (SSE) and Public-Key Encryption with Keyword Search (PEKS), have been developed to improve the security of query processes. However, the cloud platforms encounter new security challenges from time to time, as illustrated in Figure 4.1. To the best of our knowledge, the existing privacy-preserving fuzzy keyword search protocols merely take the compromised cloud infrastructures into account. The searching process for candidate fuzzy keyword trapdoors, which is the core step in fuzzy keyword search schemes, is vulnerable to such security threats. As different from the trapdoor entries which can be protected by encryption approaches and hash functions, if this searching process is intervened, the attackers can successfully tamper with the query result withouts being detected. Fortunately, recent research works of hardware-based security technologies [46][52][6] offer quasi-ideal solutions for these hidden dangers, although they are still imperfect in terms of practicality. Based on the features introduced in Chapter 1.2.2, inside TEE, the queries can be safely and efficiently processed over a plaintext keyword index, which can be further sorted for fast searching, only at the cost of an additional user-to-cloud attestation procedure. With the aid of IntelSGX, I first implement a fuzzy keyword index addressing such security threats from essence. The most challenging problem in the design

is the limitation of enclave memory. The main idea to solve the problem is to introduce a separate-type index structure to load the keyword indexes on demand, while only maintain a group of secondary indexes in the enclave to search such candidate trapdoors (as seen in Section 4.3.3).

- **Memory access pattern hiding in untrusted memory.** With the concern of large-scale datasets and the large number of demanded keywords, the enclave space can not cache the entire keyword indexes for fuzzy search. The coordination between the enclave and the untrusted memory is necessary. Consequently, the access patterns are left on main memory which can be monitored by semi-honest adversaries on cloud platform for valuable information. Such leakage, includes the access frequency to the same block, the occurrence of the same batch of keyword indexes, and other patterns which can help the attackers to explore sensitive information about user behavior, 'hotness' of the words, etc. Moreover, such analytical attacks are far more convenient to conduct than CPA-formed Keyword Guessing Attack (KGA) [16]. To overcome this, in this chapter an edit distance-based obfuscation scheme (as seen in Section 4.3.4) is proposed to prevent the adversaries from deriving useful information from observing the access pattern of untrusted memory.

- **Responses to the change of searching trend to accelerate real-time keyword queries.** From the observation of keyword searching, the query service shared in a relatively stable cluster of data users tends to show continuous iteration

of a set of hot keywords along with the timeline. To speed up the query processing associated with the current popular keywords, the proposed index scheme makes full use of the enclave memory to selectively cache the keyword indexes. The changes in the prevalence of keywords are recorded by the proposed Trend-aware Cache which periodically notifies the enclave to load in the popular fuzzy indexes and evict the aged ones. The detailed mechanism is introduced in Section 4.3.5.

To summarize, the main contributions of this chapter are as follows.

- Leveraging TEE technology, I design a secure protocol of fuzzy multi-keyword query to resist the threats in cloud environment.
- In concern of access pattern leakage, I propose a specialized ED-based Obfuscation Scheme(EDOS) upon fuzzy query processors to alleviate the threats of relevant analytic attacks.
- I optimize the protocol with novel mechanisms, i.e. Two-layer Fuzzy Index(TFI) and Trend-aware Cache (TAC), to enhance system practicality as well as efficiency.
- I conduct intensive experiments on the real-world datasets to demonstrate the security improvement and functional yields compared with the baseline solution.

The rest of the chapter is organized as follows. Some preliminaries of fuzzy keyword queries over encrypted are introduced in Section 4.2. I illustrate the proposed system design and the proposed feature mechanisms in Section 4.3, followed by the analytic

Table 1: Notations and Symbols

Symbol	Definition
Fuzzy Multi-keyword Search over Encrypted Data	
W	The set of exact keywords of given dataset
I_{inv}	Inverted Index of given W to
$ed(\cdot,\cdot)$	Edit distance from fuzzy keyword to exact keyword
T_{W_i}	Trapdoor function of keyword W_i
in OTKI-F	
d_λ	Distance scaler
d_q	The distance scaler of query
I_1, I_2	keyword index, secondary index
$\widetilde{I_1}, \widetilde{I_2}$	Encrypted form of I_1, I_2
$G_{i,d}$	fuzzy index group with edit distance d to W_i
FG_k	Searching results of k-th querying keyword
r	Recession rate of each non-matched time interval

discussions on security model in Section 4.4. The experimental results are demonstrated in Section 4.5. In Section 4.6, I conclude this chapter. Some notations related to this chapter are summarized in Table 2.

4.2 Fuzzy Keyword Search over Encrypted Document Data

In this section, I introduce the preliminaries of fuzzy keyword search to which I reference in this research work.

The keyword search over encrypted data allows authorized data users to issue queries to data storage via trapdoor functions while avoiding revealing the keyword-to-results mappings to potential adversaries. On the other hand, fuzzy searching brings extra

advantages in practice over the exact or Boolean keyword search [44] by providing resilience to the typos in keyword inputs. To support fuzzy searching, a straightforward solution is to expand each original keywords into augmented groups that include all similar keywords of them. In previous studies [94][66], the degree of similarity is defined as the edit distance (ED), whose value indicates the number of letter-wise differences to the original keyword. It is noteworthy that, the numbers of trapdoors required to support such fuzzy search increase exponentially with the pre-configured ED, i.e., the coefficient of growth on the number of trapdoors for alphabet text is $26^d \cdot \binom{d}{2l+1}$, where d is the edit distance and l is the length of original keyword. To reduce this number, Li et al. propose the wildcard fuzzy set [66]. As illustrated in Figure 4.2, they index the fuzzy keywords with the same edit distance d and the positions of letter-wise differences as a trapdoor. Though the wildcard fuzzy set and its variants have narrowed down the size of fuzzy indexes, it cannot scale well to large document databases where each index entry can contain extremely large number of encrypted document entries with fuzzy keywords sharing the same d and differential letter-wise position.

4.3 OTKI-F Protocol

In this section, I present the OTKI-F protocol for index-based multi-keyword fuzzy search using TEE. I first introduce the system model and overall algorithm, then provide detailed discussion of the three featured enhancements on privacy preservation and query performance.

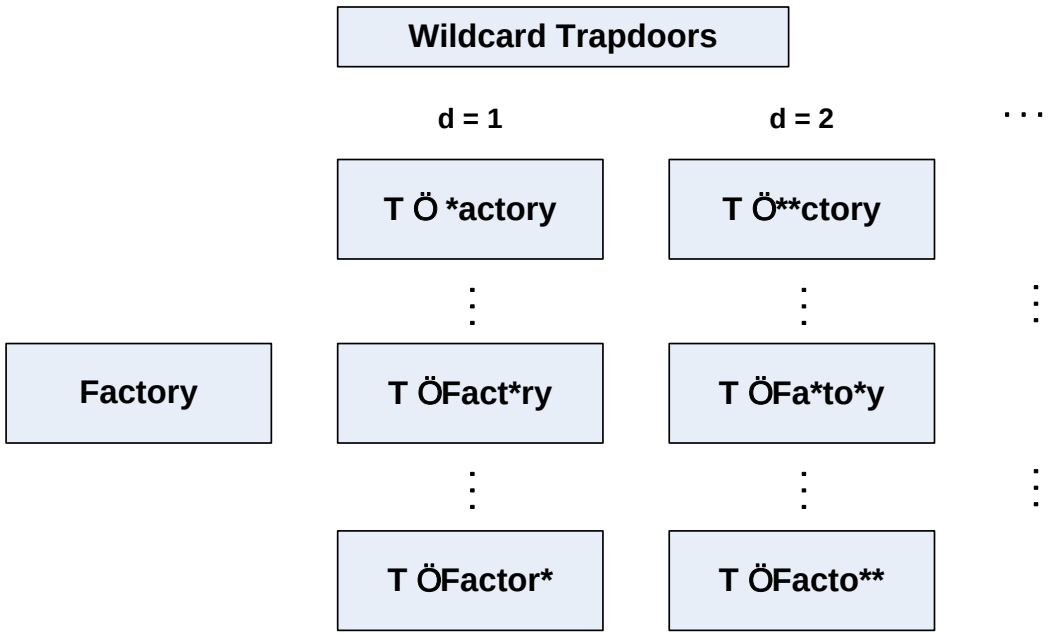


Figure 4.2: Wildcard Fuzzy Index

4.3.1 System and Security Model

As shown in Figure 4.3, the system model consists of three parties, i.e. data owner, data users, and cloud server. Data owner outsources her document database to the cloud server, which provides keyword search service [1] to data users. In this model, the data owner does not trust the cloud server and thus encrypts the database before outsourcing. An adversary can be either 'hostile' attackers or 'honest-but-curious' observers. A 'hostile' attacker can tamper with the query protocol to recover the encrypted data. On the other hand, an 'honest-but-curious' observer learns sensitive information from the memory access pattern of query execution.

[1] In this chapter, an assumption is that the keyword search only returns the topmost matched documents.

More specifically, a cloud server consists of a variety of components including the IntelSGX enclave. We trust the IntelSGX enclave and its validated communication channel with outside parties. All other components, such as the OS, hypervisor, other active processes as well as all levels of memory storage are not untrusted. Note that IntelSGX is known to be vulnerable to several side-channel attacks [14][61][45], but since most of them require strong assumption on the attacker and workload, they are not taken into consideration. To summarize, the memory-secure keyword fuzzy search can be defined as follows.

Definition 6. *(Memory-secure keyword fuzzy search) A fuzzy query σ_0 is processed under an untrusted environment E, leaving a series of access pattern ξ_0 and returning a result set R containing the keyword trapdoors. An adversary A has full access to all levels of memory in E. A fuzzy query processing protocol is memory-secure in E, if it simultaneously satisfies the following 3 conditions: (i) The result set R is always a correct answer of the trapdoor searching. (ii) During the search, A can only obtain the query statement of σ_0 and the result set R in an encrypted form. (iii) Let $\{\sigma_i\}$ denote a batch of such queries. For either two queries $\sigma_i, \sigma_j \in \{\sigma_i\}$, if they are semantically same, i.e. $\sigma_i = \sigma_j$, the probability that they show the same series of access pattern to A always suffices $Pr(\xi_i = \xi_0) < prob.$ where $prob.$ is a trivial probability.*

I will give the security proof of OTKI-F protocol against this definition in Section 4.4.

4.3.2 OTKI-F Overview

The proposed OTKI-F protocol consists of two phases, namely the index deploying phase and fuzzy keyword querying phase. As shown in Figure 4.3, the main building blocks of this protocol include keyword index, secondary index, trend-aware cache. The keyword index and secondary index form a two-layer hierarchical OTKI-F index to accelerate query processing. The keyword index is stored in an encrypted form in the untrusted memory, while the secondary index is stored in the enclave (i.e. the TEE of IntelSGX technology) as plaintext. The trend-aware cache is a data structure that offers the trend-awareness feature to further improve the efficiency of query processing under real query workloads. Algorithm 3 summarizes the main procedures in both phases, which are discussed in detail below.

A. Index Building Phase The data owner first pre-generates a one-way hash key K. Based on the inverted index of the document database, he first converts it to Wildcard fuzzy sets then further builds them into a two-layer fuzzy index as the structure shown in Figure 4.4. The detailed procedure of building the index will be presented later in Algorithm 4. Then, he invokes the SGX instruction "EENTER" to create an enclave instance and initializes the secondary index which is encrypted with the enclave secret key sk_0. After that, he hashes the keyword index with K and deploys it in the untrusted memory of cloud server.

B. Fuzzy Querying Phase The data user provisions with the enclave through a remote attestation and derives the user session key sk_u. In more detail, remote user first derives

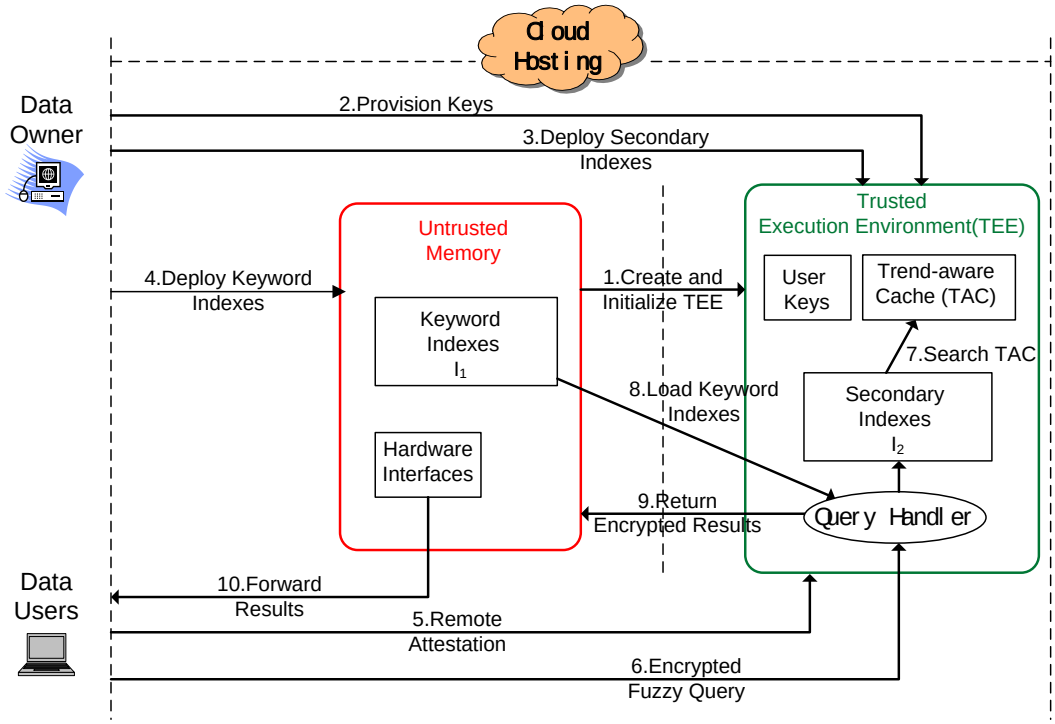


Figure 4.3: Processing Fuzzy Keyword Queries

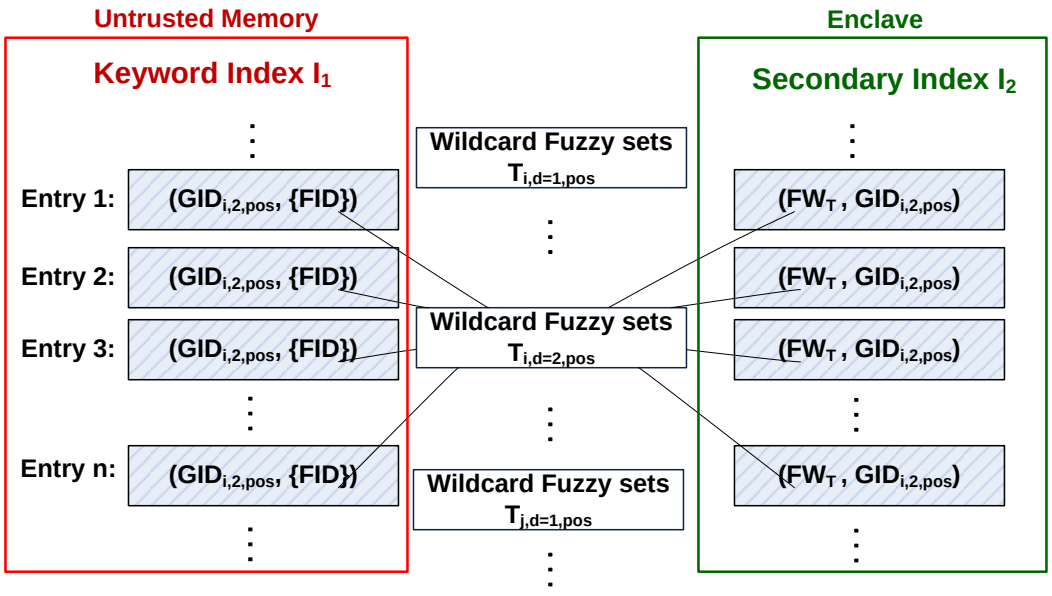


Figure 4.4: Two-layer Fuzzy Index

the enclave's EPID GID (extended group ID of EPID) and requests Intel Attestation Server (IAS) to retrieve the enclave's signature revocation information (SigRL) for verifying the enclave. Then it further checks the QUOTE Report (sgx_quote) with IAS to attest the enclave is at correct cryptographic status. After the remote attestation, data user's query request Q is encrypted with sk_u and sent to be decrypted inside enclave. When request Q is received by cloud server, it is processed by retrieving the two-layer index under the protection of ED-based Obfuscation Scheme (i.e., Algorithm 5). Briefly, the secondary index handles the query first and returns candidate fuzzy keyword groups whose original keywords are similar to the input query words. Afterward, the candidate keywords are searched in the trend-aware cache (TAC). If they are found, the final query results will contain those documents that have a particular keyword. Otherwise, the fuzzy keyword groups are retrieved from the keyword index. After combining the partial results from each input, the trend-aware cache is updated according to Algorithm 6.

Algorithm 3 Overall OTKI-F Protocol

Index Building Phase:

1: $HashKeyGen() \rightarrow K$.
2: Run Algorithm 4:
$I_2 \leftarrow buildI2()$. $\widetilde{I_2} \leftarrow Enc(sk_0, I_2)$.
$I_1 \leftarrow buildI1()$. $\widetilde{I_1} \leftarrow Enc(K, I_1)$.
3: Invoke $EENTER(\widetilde{I_2})$.
4: Send $\widetilde{I_1}$ to cloud server.

Fuzzy Querying Phase:

5: Data user sends query request T.
6: Run Algorithm 5 to execute query.
7: Run Algorithm 6 to update trend-aware cache to state ρ.
8: Adjust the cached keyword index entries according to ρ.

4.3.3 Two-layer Fuzzy Index

The two-layer fuzzy index is composed of an encrypted keyword index $\widetilde{I_1}$ outsourced to the untrusted cloud server and a secondary index $\widetilde{I_2}$ encrypted by the provisioned secret key and pre-loaded into IntelSGX enclave.

Algorithm 4 shows the construction process of this index. The data owner first traverses inverted index I_{inv} converting each of its entries to Wildcard fuzzy sets. A Wildcard fuzzy set is denoted as $\tau = (FW, \{FID\})$, where FW is the identifier of wildcard-based fuzzy set containing the information of differences to original keyword and $\{FID\}$ indicates the associated documents which contain the identifier. Then, each τ is assigned to a fuzzy index group $G_{i,d,pos}$ through a one-to-one mapping based on the edit distance d to the original keyword W_i and the position list pos of character differences. Note that the index groups of a given original keyword may intersect with other index groups. For example, a fuzzy set of original keyword "MERS" with edit distance $d = 2$ can overlap with the index group of the exact original keyword ($d = 0$) "SARS" on $\{FID\}$. Meanwhile, a user-defined threshold d_λ is introduced to discard those fuzzy keywords whose edit distances exceed this threshold. As such, each encoded group id $GID_{i,d,pos}$ serves as the key of keyword index I_1 and $\{FID\}$ becomes its corresponding index value. Then the algorithm hashes $\{FID\}$ of each index entry in I_1 with key K (e.g., HMAC-SHA256) into $\widetilde{I_1} = GID_{i,d,pos}, H_K\{FID\}$ before deploying the latter to the cloud database. To build the secondary index I_2, the algorithm uses the plaintext FW as the index key and $GID_{i,d,pos}$ as its corresponding index value. Before sealing it into the

enclave, the algorithm encrypts I_2 with the secret key sk_0 into $\widetilde{I_2}$.

Algorithm 4 Build OTKI-F Index

Input: Original keywords set W. Fuzzy scaler d_λ. Pre-generated hash key K. Owner's secret key sk_0. Document IDs $\{FID\}$. Inverted index of the documents I_{inv}. $I_1 \leftarrow \emptyset, I_2 \leftarrow \emptyset$.

Output: Output keyword index $\widetilde{I_1}$, secondary index $\widetilde{I_2}$.

1: **for each** entry in I_{inv} **do**
2: Convert it to Wildcard index entry $\tau = (FW, \{FID\})$.
3: **end for**
4: **for each** τ **do**
5: Assign τ to $G_{i,d,pos}$, $GID_{i,d,pos} = encodeID(FW, d, pos)$.
6: **end for**
7: **for each** $G_{i,d,pos}$ **do**
8: $I_1 \leftarrow I_1 \cup (GID_{i,d,pos}, \{FID\})$.
9: $\widetilde{I_1} \leftarrow \widetilde{I_1} \cup (GID_{i,d,pos}, H_K\{FID\})$.
10: $I_2 \leftarrow I_2 \cup (FW, GID_{i,d,pos})$.
11: $\widetilde{I_2} = Enc(sk_0, I_2)$.
12: **end for**
13: Return $\widetilde{I_1}, \widetilde{I_2}$.

4.3.4 ED-based Obfuscation Scheme

In processing user queries with the two-layer fuzzy index, access pattern leakage will occur when TEE retrieves the keyword index I_1 from the untrusted memory. Specifically, to limit the number of search results, many fuzzy search protocols have a threshold on edit distance $d_q, d_q \leq d_\lambda$ between word input and its feasible keyword. As such, the same access patterns on keyword index may be inferred as the same queries, which jeopardizes the privacy of users and compromises the security of keyword space. To guard against this, an ED-based Obfuscation Mechanism is proposed to make individual keyword queries indistinguishable by adversaries with polynomial time. As shown in

Algorithm 5), it resolves this issue by retrieving probabilistic supersets of requested keyword indexes from untrusted memory. An obfuscation parameter ϵ is used to determine the scale of such supersets and balance the trade-off between communication overhead and obfuscation degree. The security analysis of this mechanism will be presented in Section 4.4.2 and the discussion on the impact of ϵ on the performance will be shown in Section 4.5.3.

The ED-based Obfuscation Mechanism works as follows. An encrypted fuzzy search request T contains n querying inputs and an edit distance parameter d_q, denoted as $T = Enc(sk_u, Q := [Wq_1, Wq_2, ..., Wq_n], d_q)$. Once the server receives the request, it scans the key of plaintext (sorted) secondary index I_2, namely FW, with each querying word Wq_k for their matched entries. Except for the string match to identifier FW, the sequential search also references to the edit distance d between original keyword W_i and FW. If $d \leq d_q (\leq d_\lambda)$, it merges the entry value GID_s directly to the intermediate set $\{GID\}$, whereas the rest entries whose d suffices $d_q < d \leq d_\lambda$ are selected with probability $Pr = \epsilon^{(d-d_q)}$ before being merged into $\{GID\}$. The enclave retrieves $\{GID\}$ first in the trend-aware cache and then in keyword index $\widetilde{I_1}$ for the candidate document IDs. The matched entries of keyword index are aggregated in $\{GR\}$, which is a superset of the real query. A followup pruning by $d \leq d_q$ removes redundant elements whose original keywords deviate from input word Wq_k in a distance larger than d_q. As such the document IDs FID_j in G_j are merged into a partial result set associating with single querying word $\{FG_k\}$ accordingly. Finally, the enclave obtains the multi-keyword

searching results $\{FG\}$ by intersecting all $\{FG_k\}$.

Algorithm 5 Searching indexes with ED-based Obfuscation

Input: Encrypted query string
$T = Enc(sk_u, Q := [Wq_1, Wq_2, ...], d_q)$.
Intermediate set of group ids $\{GID\} \leftarrow \emptyset$.
Intermediate result document id sets $\{FG\}_k \leftarrow \emptyset$ of the k-th querying keyword.
Obfuscation parameter ϵ.
Output: Query result document id set $\{FG\}$

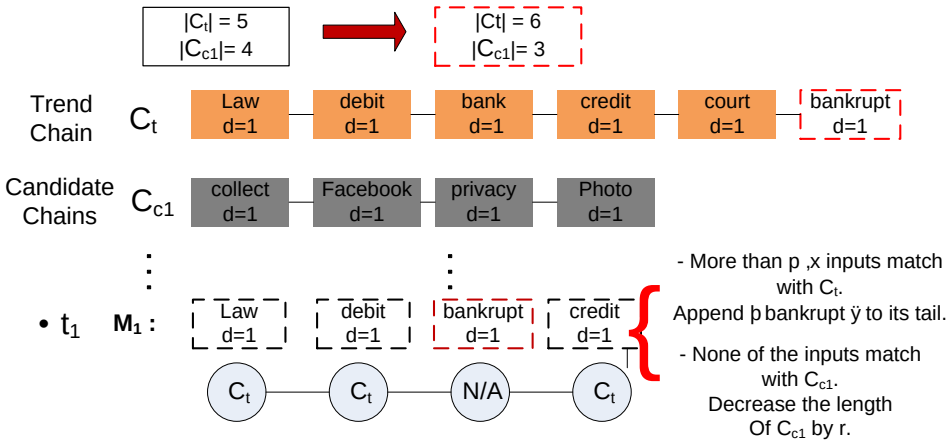
Decrypt T with corresponding sk_u provisioned in attestation session.
for each $Wq_k \in Q$ **do**
  Sequentially search Wq_k among I_2 for matches.
  for each matched I_2 entry **do**
    if $d \leq d_q$ **then**
      $\{GID\} \leftarrow G \cup GID_s$.
    else
      GID_s is merged into $\{GID\}$ at a probability $Pr = \epsilon^{(d-d_q)}$.
    end if
  end for
end for
Lookup each element of $\{GID\}$ in trend-aware cache.
if Cached keyword index entries $\{GC\}! = \emptyset$ **then**
  Merge $\{GC\}$ into $\{GR\}$. Remove matched GID from $\{GID\}$.
end if
Invoke $OCALL(searchI1())$ to retrieve rest elements of $\{GID\}$ in keyword index $\widetilde{I_1}$.
Merge retrieved entries of keyword index into $\{GR\}$.
for each loaded entry $G_j \in \{GR\}$ **do**
  if $d \leq d_q$ **then**
    $\{FG_k\} \leftarrow \{FG_k\} \cup FID_j$.
  end if
end for
$\{FG\} = \{FG_1\} \bigcap \{FG_2\} \bigcap \{FG_3\} ... \bigcap \{FG_m\}$.
Return $\{FG\}$ to data user.

4.3.5 Trend-aware Cache

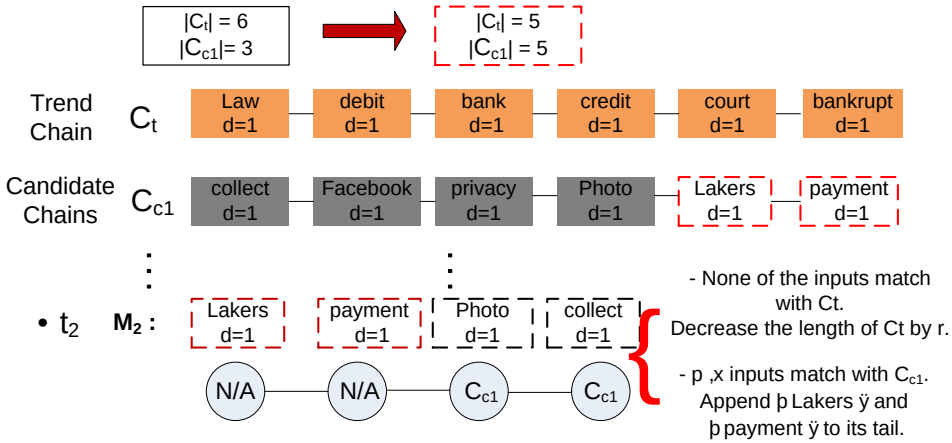
It is a common observation that keyword search exhibits a changing trend of prevalent words over time [43][118], for example, in a material e-library of editorial studio, and email search in an institutional email archive. To this end, caching the documents (more precisely the document IDs) which contain frequently searched keywords in an enclave heap can reduce the number of OCALLs to the untrusted memory to retrieve keyword index $\widetilde{I}_1$. Ideally, the keywords of all search requests can be cached. However, in practice, since the enclave has limited size and popular words change over time, such cache cannot always hit. To capture the change of "hot" keywords precisely and improve the utility of limited enclave cache space, a spectial mechanism called Trend-aware Cache (TAC) is provided.

Trend-aware Cache (TAC) is composed of one trend chain[2] C_t and s candidate chains C_{c_j} initialized as an empty chain $\diamond$. Each of these chains contains keywords that frequently co-occur in the keyword index retrieval. Hence each chain indicates a possible active theme among the conversations of users. In a nutshell, TAC always cumulates the lengths of the chains which may contain appearing prevalent themes, while reduces the lengths of rests. As such, the longest chain represents the most popular theme in keyword search services. As demonstrated in Algorithm 6, after each time interval ΔT, the aforementioned chains are updated based on the most recent index group retrieval

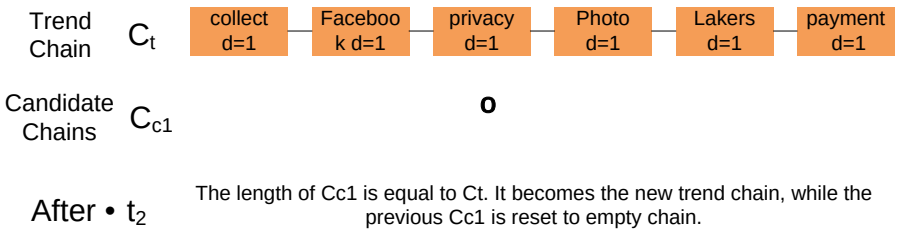
[2]The number of trend chains is not restricted. There also can be more than 1 prevalent themes coexist in the enclave cache. For simplicity, the setting of one single trend chain is used throughout the chapter as default.



(a) Δt_1



(b) Δt_2



(c) After Δt_2

Figure 4.5: Example of Trend Chain Updating

records and compete for the prevalent theme. The meta-data of records is the x most frequently accessed index groups, denoted as M_i. Then, it scans each chain with the indexes in M_i and counts the number of matches N_t and N_{c_j} for C_t and C_{c_j} respectively. The trend chain C_t is examined in prior. If it is empty or N_t is no less than a thresholding percentage p of x, the unmatched index group IDs are appended to the tail of the trend chain. In other cases, it continues to examine the candidate chains. Likewise, if either chain whose N_{c_j} is equal or greater than $\frac{x}{2}$ or is an empty chain, the unmatched index group IDs in M_i are appended to its tail. For all the chains, once appended with new IDs, its length $|C_t|$ or $|C_{c_j}|$ increases with $|N_t|$ or N_{c_j} respectively. Otherwise, the chains which are not appended undergo a deduction on their lengths for loss on the hotness. The number of deduction is determined by preset receding rate r. After scanning all the chains, the longest chain is copied to substitute the trend chain, while itself is restored to $\diamond$. An example of how the compositions of the chains change is shown in Figure 4.5.

4.3.6 Limitations

First, the proposed OTKI-F protocol does not cover the access to the actual document files. A secure and oblivious protocol for secure document file retrieval from untrusted sources is hence needed. Second, assume that the keyword index $\widetilde{I_1}$ can be loaded entirely into the untrusted memory for execution. However, if this assumption does not hold, such protocol will be vulnerable to those attacks leveraging the auxiliary knowledge already derived from disk storage. For example, the $\widetilde{I_1}$ is split into separated DB files

Algorithm 6 Trend-aware Cache

Input: Initialize 1 trend chain C_t, s candidate chains C_{c_j} as empty chain, denoted as $\diamond$. Thresholding percentage p, Receding rate r.

1: **for each** time interval ΔT_i **do**
2: Invoke $xtop()$ to record x most frequent accessed index groups as M_i, $|M_i| = x$.
3: Let N_t, N_{c_j} denotes the number of matched groups with C_t, C_{c_j} respectively.
4: **for each** existing chain C_t and C_{c_j} **do**
5: **if** $N_t \geq p \cdot x \quad || \quad C_t = \diamond$ **then**
6: Append unmatched group IDs U to the end of C_t.
7: $|C_t| \leftarrow |C_t| + |N_t|$.
8: **else**
9: **if** $N_{c_j} \geq p \cdot x \quad || \quad \exists \quad |C_{c_j}| = \diamond$ **then**
10: Append unmatched group IDs U to the end of C_{c_j}.
11: $|C_t| \leftarrow |C_t| - r$, $|C_{c_j}| \leftarrow |C_{c_j}| + |N_{c_j}|$.
12: **else**
13: $|C_t| \leftarrow |C_t| - r$, $|C_{c_j}| \leftarrow |C_{c_j}| - r$.
14: **end if**
15: **end if**
16: **end for**
17: **if** $\exists \quad |C_{c_j}| \geq |C_t|$ **then**
18: $C_t \leftarrow C_{c_j}$, $C_{c_j} \leftarrow \diamond$.
19: **end if**
20: **end for**

on disk with explicit identifiers, such as file names, and is loaded partially on demand. That said, adversaries can imply with the scope of keywords that likely to be searched currently. On the other hand, for the extremely large database or keyword space, the enclave may fail to load even the secondary index $\widetilde{I_2}$ in one pass. I plan to resolve this with some oblivious swapping mechanisms in the future.

4.4 Security Analytics

In this section, the security and privacy functionalities of protocol components is analyzed and the proposed schemes is proved to satisfy all three conditions in the definition of memory-secure keyword fuzzy search (as seen in Definition 6).

4.4.1 Condition *i* and *ii* Security

I now give the proof that query process of OTKI-F Protocol satisfies the Conditions *i* and *ii* based on the security standards claimed in the literature.

Theorem 4. *(Condition i&ii sufficiency): A keyword fuzzy search protocol satisfies Condition i and ii security, if each of its building blocks associated with keyword index (trapdoors) searching is inaccessible to the adversaries and the entries of keyword index are protected by acknowledged cryptographic schemes.*

Proof. **Sketch.** Since the adversary can not access the building blocks which are directly related to the search process of candidate trapdoors in the enclave, the search always

returns the correct results. Therefore Condition i holds. The utilization of encryption and hash function ensures the accessible memory space is always operated in encrypted form including query statement and results. Therefore Condition ii holds.

Recall that, there are three main components in the proposed protocol, i.e. keyword index (trapdoors), secondary index, trend-aware cache. Among them, keyword index $\widetilde{I}_1$ (trapdoors) is located in untrusted memory. As a community consensus, the applied HMAC-SHA256 hash function is resistant to collision attack and remains robust against length extension attack. The secondary index and trend-aware cache are sealed into IntelSGX enclave and the outgoing channels are protected by secure function calls [26] and the usage does not surpass the guaranteed security boundary of SGX as explained in Intel's manuals and reference books [50][49]. Hence, Theorem 4 holds for OTKI-F protocol. □

Remark 2. *Though the existence of attacks based on L1 terminal fault flaw, e.g. Foreshadow[101], has challenged the trust model of IntelSGX, it is still trustworthy in the cloud environment with the conditions that Intel microcode patches have been updated or the hyper-threading is turned off.*

4.4.2 Condition *iii* Security

This subsection proves that the query process of OTKI-F Protocol suffices the Condition *iii* based on the quantitative evaluation. Prior to the proof, a formally definition of the selective access pattern attack that may take place in the scenario of fuzzy keyword

search are given as below.

Definition 7. *($\widetilde{I_1}$ Access pattern attack) The adversary holds the oracle that the queries toward certain specified original keywords will burst in the coming time period. He then begins to monitor the index entries of $\widetilde{I_1}$ retrieved to be loaded to the enclave. From his perspective, the same access pattern for a retrieval means the same fuzzy search request. Based on this, the sensitive information he can further infer includes the original keywords of encrypted keyword indexes, the frequency of a particular query, etc.*

Theorem 5. *(Condition iii sufficiency): In OTKI-F Protocol, if the access pattern attack succeeds only with a trivial probability, the protocol satisfies Condition iii security.*

Proof. According to Definition 7, during the bursting period, the adversary actually regards the arriving queries as the same input to seek a match in corresponding access pattern on $\widetilde{I_1}$. Therefore, if the probability of the occurrence of the same series of access patterns is trivial with such oracle, the Condition *iii* in Definition 6 must also hold. □

In what follows, the success rate of the access pattern attack is formally modeled to show it is indeed trivial in practice, and therefore Theorem 2 holds. During the runtime, $\widetilde{I_1}$ is accessed for searching requested index group IDs based on the instructions issued from the enclave. The set of requested index group IDs forms the superset of feasible original keywords of querying word inputs as mentioned in Section 4.3.4. In the retrieval of secondary index I_2, the model do not attempts to predict the actual original keyword of a fuzzy input (namely, target keyword) which means the additional index groups contained in the superset consists of not only the $\widetilde{I_1}$ entries whose edit distance fall in

the span $d_q \leq d \leq d_\lambda$ but also the associate index groups of other original keywords whose edit distances to the particular input are less than d_λ. In the nature of ED-based Obfuscation Scheme, the more such probabilistic additional index groups retrieved from $\widetilde{I}_1$, the memory access patterns become more oblivious to the adversaries.

Now I evaluate the obfuscation imposed to adversary produced by protocol mechanism based on the probability *prob.* that two queries with same x querying words generate exactly the same access pattern on $\widetilde{I}_1$. The core factor of the probability is the numbers of additional index groups N_a involved in the super set. *prob.* is formulated as:

$$prob. = \frac{1}{2^{N_a}} = \frac{1}{2^{\sum_{i \leq x} N_i}} \tag{4.1}$$

where N_i is the total number of additional index groups of the i-th input word.

For each input word i, given the maximal edit distance d_λ of $\widetilde{I}_1$, the query distance allowance d_q, the set of index groups with distance equal to or under d_q is denoted as TG_i. The super set of TG_i containing all possible keyword index groups is denoted as SP_i, and the differential set of SP_i and TG_i is denoted as D_i. Then expand D_i as = $D_i = \{\Omega_0, \Omega_1, ..., \Omega_n\}$, where Ω_k indicates the k-th possible target keyword associated the input keyword. Let N_{Ω_k} denote the expectation number of additional index groups associate with the k-th possible target keyword after the probabilistic selection. Then N_i is calculated as:

$$N_i = |D_i| = \sum\nolimits_{k=1...n} N_{\Omega_k}. \tag{4.2}$$

Next, substitute the value of N_{Ω_k} with:

$$N_{\Omega_k} = \sum\nolimits_{d \in d_q < d \leq d_\lambda} \binom{d}{2l_k + 1} \cdot \epsilon^{d - d_q} \tag{4.3}$$

where l_k denotes the length of k-th possible target keyword. For each target keyword, d denotes the edit distance to the corresponding original keyword of the specific index group.

Finally, perform a stepwise substitution to N_i with the above expressions and derive the quantitative evaluation model of obfuscation functionality as the equation below:

$$prob. = x \cdot n \cdot \sum\nolimits_{d \in d_q < d \leq d_\lambda} \binom{d}{2l_k + 1} \cdot \epsilon^{d - d_q}. \tag{4.4}$$

In a practical setting, let $d_\lambda = 3$, $d_q = 1$, $n = 3$, $x = 3$, and fix l_k to 5 as close to the average length of English word (4.7) [71]. It shows that the success rate *prob.* of access pattern attack drops to $2^{-282.15}$ which is trivial in practical settings, when ϵ is set to a reasonable value, 0.3, to control the communication overhead. Hence, Theorem 5 holds for OTKI-F protocol.

4.5 Experimental Results

In this section, I evaluate the performance of OTKI-F as a memory-secure multi-keyword fuzzy search protocol. First, the size of OTKI-F two-layer index is demonstrated to show

its practicality and compatibility with Intel SGX. Then a thorough evaluation on the efficiency of OTKI-F fuzzy search algorithm under a variety of parameter settings is given. Third, the effectiveness of ED-based obfuscation Scheme which enhances the privacy of keywords is discussed. At last, I test the performance yield of trend-aware cache with simulated queries of dynamic trending keywords.

4.5.1 Efficiency of Building Index

In this subsection, the overall size of OTKI-F index is measured and compared with the naive Wildcard protocol [66]. The applied dataset is a truncated 'ENRONMAIL' containing 10,000 files to extract the keywords for indexing. With the same maximal fuzzy scale $d_\lambda = 2$, both OTKI-F and Wildcard indexes assemble the possible fuzzy words into their entries, respectively. In the experiment, the keys of Wildcard index entries are hashed with HMAC-SHA256, while the OTKI-F secondary indexes are measured in plaintext.

As shown in Figure 4.6, the secondary index of OTKI-F is about $11X$ smaller than the naive Wildcard index throughout the process. Further, according to this figure, it can be projected that an approximately 90MB enclave can accommodate a secondary index I_2 with around $10,000$ keywords, which is sufficient in practice as the latest Longman Dictionary of Contemporary English only identifies 9,000 English words as common ones.

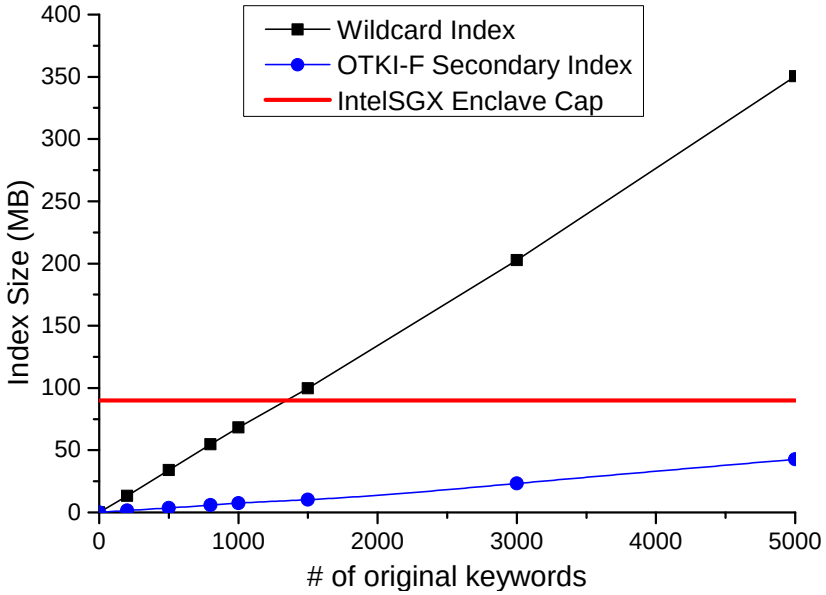


Figure 4.6: Index Size: Wildcard fuzzy index vs. OTKI-F secondary index

4.5.2 Performance of Fuzzy Keyword Search

In this subsection, I perform three sets of experiments to evaluate the impact of dataset and queries on the performance of two-layer index. The experiments are conducted on a desktop computer with Intel i7 6700HQ CPU, 16 GB RAM running Windows 10. An Intel SGX enclave instance is created in hardware mode and pre-configured with 'MAX_STACK_SIZE = 2MB' and 'MAX_HEAP_SIZE = 30MB'. 5000 annotated email files in the 'ENRONMAIL' dataset is used to build an inverted index and then convert it to an OTKI-F two-layer index. Then it stores and decrypts the secondary index I_2 in the enclave as plaintext meta-data. I_2 is then sorted with binary tree to speedup searching. For the three parts, the maximum edit distance d_λ is fixed as 2 for the first two experiments and fuzzy scale allowance d_q is set to 1 for all three. The trend-aware caching is disabled to avoid its impact on the performance measurements in the second

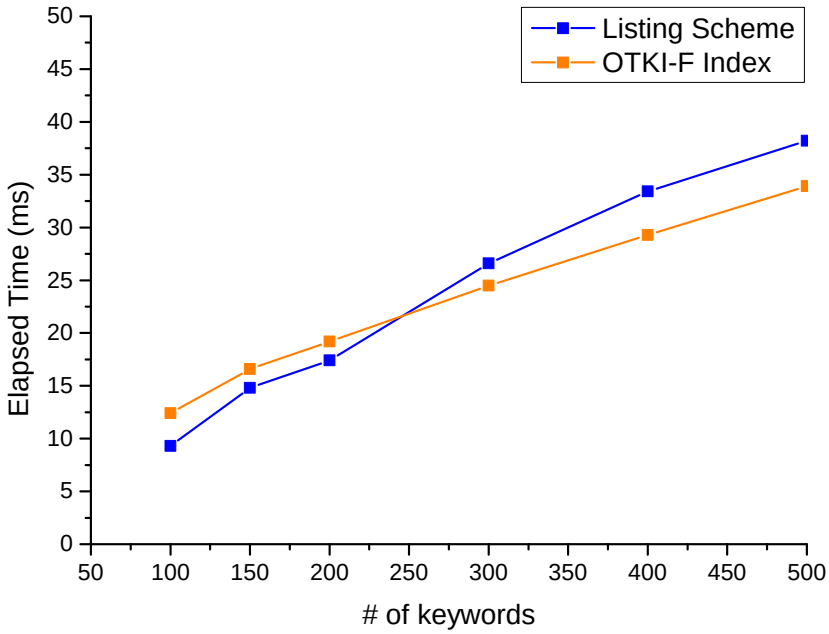


Figure 4.7: End-to-end searching time with changing keyword universe size and fixed d_λ

and third experiment.

In the first experiment, a single request search processing time is measured by varying the size of keyword universe. The elapsed time is counted from the issuing of query to the point that resulting document IDs returns. Since a large proportion of keywords of inverted index are sparse in mapping to documents (i.e., owns less than 5 mapped email files). Certain keywords are randomly removed from the keyword universe before it is converted to an OTKI-F index and the remaining number of keywords $|W|$ ranges from 100 to 500. Each query is composed of 3 wrongly spelled keywords, each with edit distance $d = 1$ to its target keyword.

The end-to-end searching time of OTKI-F index is compared with SSE-based listing approach mentioned in [66]. The listing approach indexes the encrypted document files

with trapdoors of fuzzy sets of keywords on different edit distances. In such baseline solution, the number of its LSH (i.e. locality-sensitive hash) of the is set to 30 and the LSH miss is set to 8 to ensure enough accuracy. As the number of original keywords increases, OTKI-F index gradually outperforms the listing approach when the number of original keywords grows as shown in Figure 4.7. I attribute this observation to two reasons: (a) The advantage of plaintext index search in enclave applied in the proposed protocol compared to the SSE-based search used in listing approach becomes obvious. (b) The trend-aware cache reduces the frequency of retrievals for $\widetilde{I_1}$.

In the second experiment, the number of input words n varys from 1 to 8 increasingly in 5 series of trial runs. Each series illustrates a different number of original keyword universe $|W|$ ranging from 100 to 500. As shown in Figure 4.8, the elapsed time shows a linear increase with the growth of n. Further, the gaps between the series also increase due to the logarithmic growth of elapsed time with the increase on $|W|$.

In the third experiment, the number of input words is fixed as $n = 2$ and the elapsed time under various maximal fuzzy scales $d_\lambda = 1, 2$ and 3 is measured. Figure 4.9 plots the 3 sets of results against the change of $|W|$ on X-axis. It is observed that the increasing rate on query processing time with growing $|W|$ tends to be linear when $d_\lambda = 3$, while remains sub-linear when $d_\lambda = 1$. I attribute this phenomenon to the exponentially increase on fuzzy index entries caused by the maximal fuzzy scale d_λ.

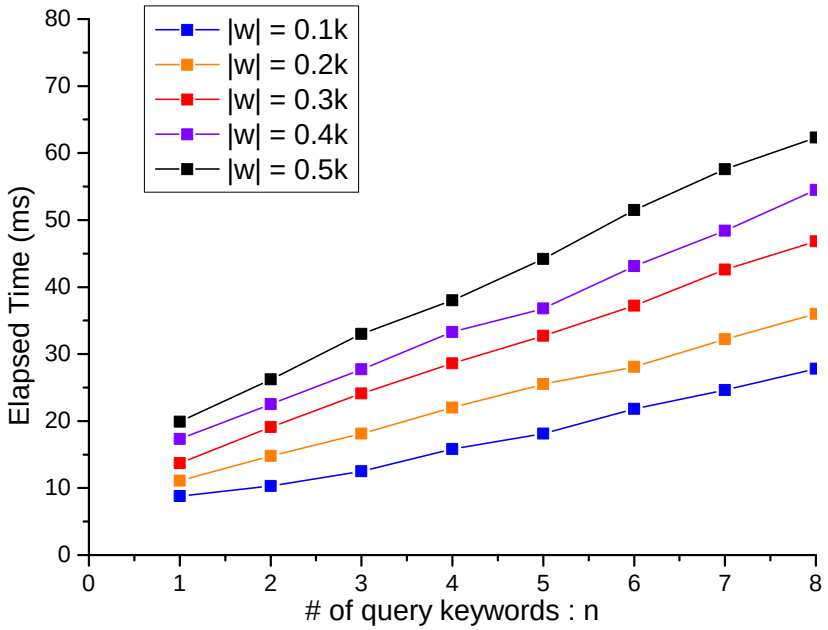


Figure 4.8: End-to-end searching time of multiple keywords queries

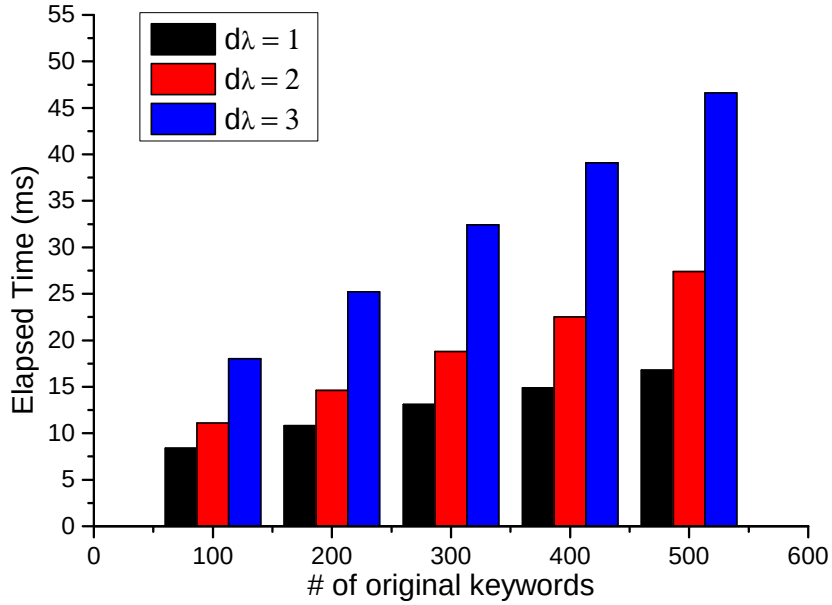
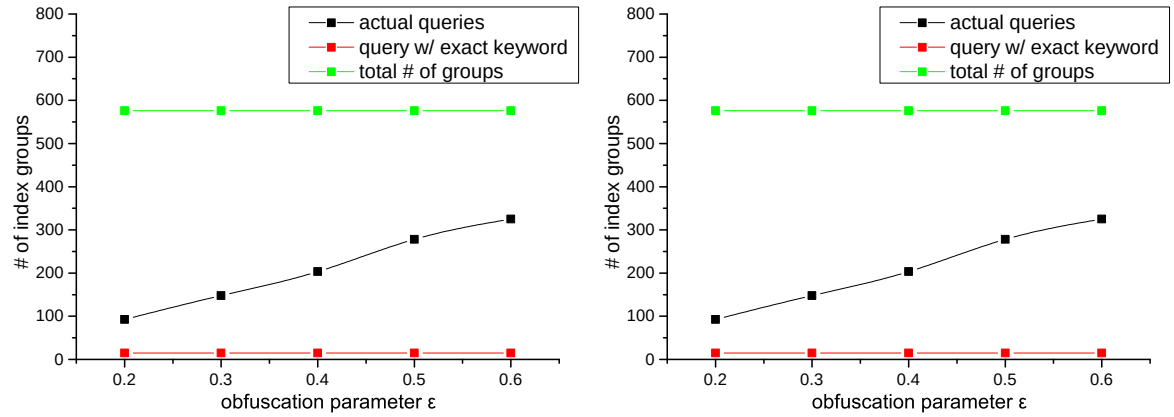


Figure 4.9: End-to-end searching time on changing d_λ

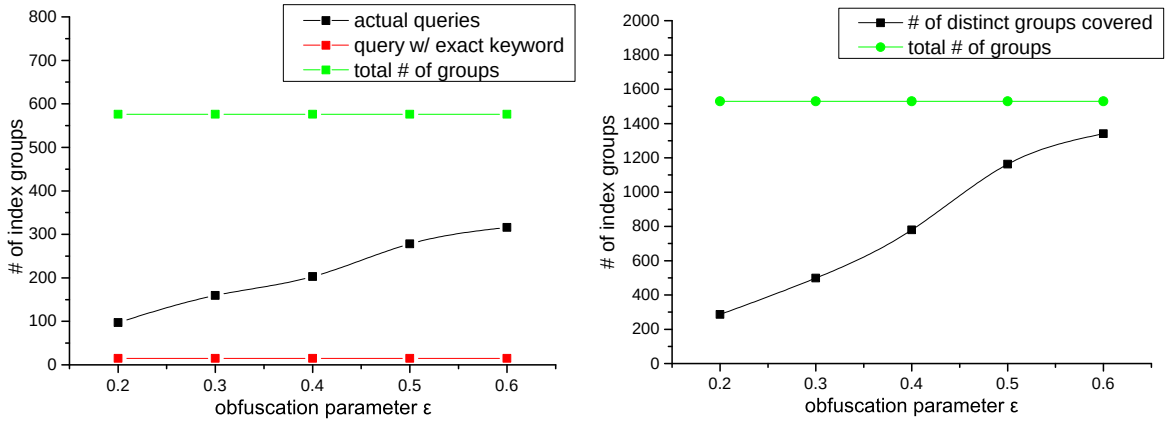
4.5.3 Effectiveness of ED-based Obfuscation Scheme

In this subsection, I analyze the degree of confusion incurred to adversary by the proposed ED-based Obfuscation Scheme. Recall the analytic model in Section 4.4.2, the effect of access pattern hiding on keyword index $\widetilde{I_1}$ is correlated with the number of additional index groups N_a accessed on $\widetilde{I_1}$. The actual value of N_a is directly impacted by the obfuscation parameter ϵ. To this end, I collect the average numbers of accessed $\widetilde{I_1}$ entries (i.e. index groups) with changing ϵ ranging in $[0.2, 0.6]$. A fixed query string containing three input words is used to perform a $d_q = 1$ fuzzy search against the simulated dataset (100 original keywords) with maximal fuzzy scale $d_\lambda = 3$. Each of the three inputs is a fuzzy keyword with $d = 1$ to its target keyword and does not overlap with others on index groups. Then collect the actual numbers of additional index groups N_a retrieved from $\widetilde{I_1}$ of each keyword. 5 trial runs are conducted for each ϵ. Figure 4.10 plots the actual value of N_a for each input word as well as the aggregated number of distinct index groups being accessed.

As shown in Figure 4.10, the rise of average N_a along with the increasing ϵ reflects a positive correlation between the degree of obfuscation and obfuscation parameter ϵ. It is also worth noticing that the number of distinct additional index groups accumulates fast to a relatively large coverage of theoretical N_a, which coincides with the analytical model in Section 4.4.2).



(a) d=1,MEETING (b) d=1,ASSETS



(c) d=1,PRIVACY (d) $d_q = 1$, Aggregated

Figure 4.10: # of accessed $\widetilde{I}_1$ entries with changing ϵ

4.5.4 Performance of Trend-aware Cache

In this subsection, the performance of Trend-aware Cache (TAC) is evaluated by measuring its cache hit rates from I_2 search results. First, the truncated 'ENRONMAIL' dataset used in Section 4.5.1 is sorted by date. For each date, top-6 frequent included keywords from the annotated emails of that day are selected as the seeds of search trend. Time interval ΔT_i is set to 1 day. The random $d = 1$ fuzzy words of the selected hot keyword are used in searchs[3]. Then the numbers of requested index groups after scanning I_2 are collected and the cache size is fixed to 4MB to calculate the hit rate. As for the TAC parameters aforementioned in Section 4.3.5, the number of candidate chains is set as $s = 3$, and the threshold percentage is set as $p = 50\%$ in identifying the potential trend. The receding rate is set to $r = 1$ to control the eviction speed. Two date spans from the database are randomly selected, each with 20 constant days. The average result values of each date span are obtained from 10 repeating trials.

As shown in Figure 4.11 (a) and (b), the cache hit rates maintain above 15% after several intervals in both date spans and show a peak at 54.5% and 62.4% respectively. Comparing with first-in-first-out (FIFO), a common cache eviction strategy, hit rates of enclave cache are improved by 58.2% and 103.7% on average.

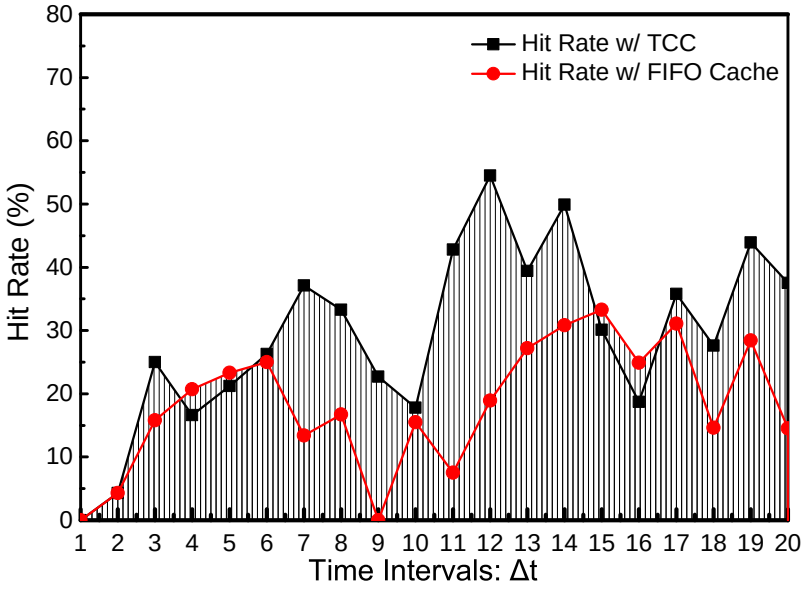
Furthermore, Trend-aware Cache receives higher performance yield when more trend chains are allowed to co-exist, especially for the datasets whose search trends concentrate on more than one theme, such as "WEIBO Hotwords" provided by [93]. As plotted in

[3]In this experiment, the search stops after deriving the results of secondary index search.

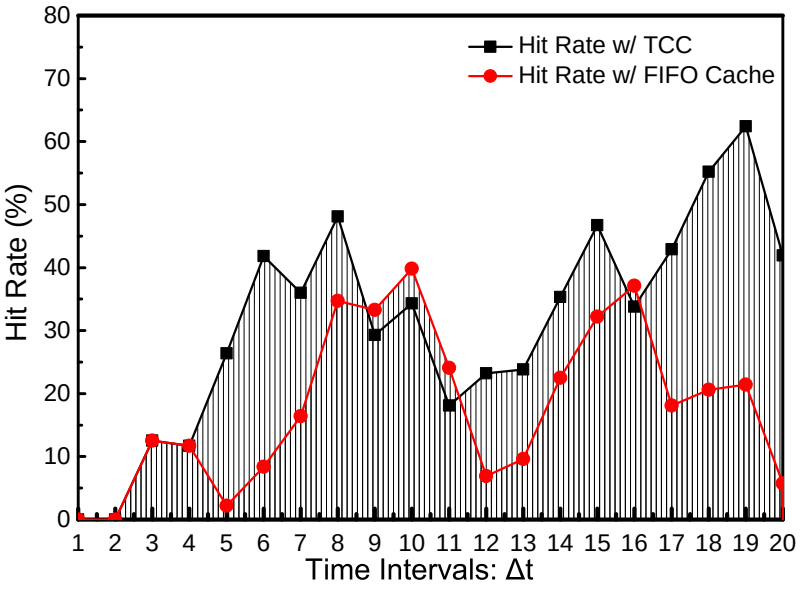
Figure 4.12, the sort 15 consecutive hourly hot word sets (Date: June 29th, 2020) are sorted and top-20 frequent keywords for each hour are selected to evaluate TAC performance of different numbers of trend chains. I set the threshold percentage $p = 20\%$ and keep the rest settings unchanged as the previous experiment. The resulting hit rates show a positive correlation with the number of trend chains on average values.

4.6 Summary

In this chapter, I propose OTKI-F, a memory secure multi-keyword fuzzy search protocol built on a separate type keyword index. Leveraging IntelSGX technology, it ensures the exclusive access of authorized stakeholders to second layer of index. The first layer is protected by cryptographic tool and a novel obfuscation scheme from memory tampering and access pattern sniffing respectively. I perform rigorous analysis on the claimed security and privacy functionality with the security definition. On the other hand, a trend-aware cache coordinator is employed to achieve moderate query performance or even higher for trend-sensitive queries. I evaluate system performance and internal parameters of OTKI-F with various simulation experiments and real dataset tests. The novel mechanisms proposed in this work can be solely applied to other applications offering significant enhancements. As for the extension of this study, I plan to keep updating with the progress of TEE technology and provide security solutions to different types of queries.



(a) November 2000



(b) June 2001

Figure 4.11: Cache hit rates change with constant queries

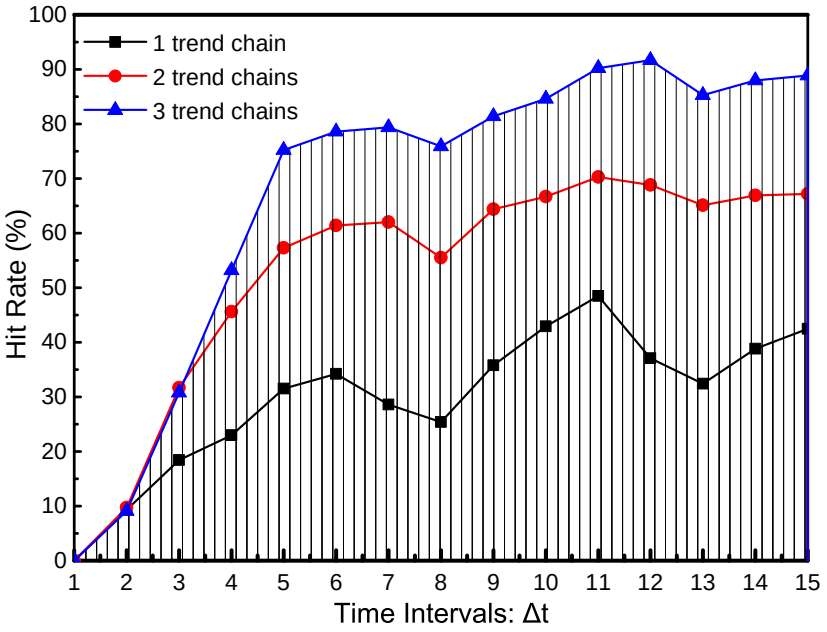


Figure 4.12: Cache hit rates change with the # of trend chains

Chapter 5

Access Frequency Pattern Hiding for General Query Processing

5.1 Introduction

In the previous chapter, motivated by the efficiency loss of using both TEE technology and a standalone obfuscation scheme (i.e. the Oblivious RAM) to achieve secure and privacy- preserving query processing, I explore the potential solution that combines the obfuscation approach with the origin query processing, that is, the fuzzy keyword search. Essentially, the main idea is to add randomness to the original result. Since the original result is already augmented during the fuzzy keyword search, it cost only limited incremental overhead in processing. However, after a wide survey into various query services, I find the idea is only adaptable for the inclusion searches (i.e. searching the entities that

include some specified elements), such as fuzzy searches and subgraph searches. That said, in other categories, the queries tend to return the absolute or proper (exact) search result according to the requests. Therefore, they still need standalone mechanisms that are specially designed for different queries to achieve the perturbation on data access patterns. To provide a general solution for different queries mentioned above, in this chapter, an ad-hoc secure module is proposed using the TEE technology to imposed the perturbation on the query requests prior to their processing stage. A line of proposed protocols [79][72][41] also studies from this perspective. They hide the access frequency patterns using the idea of injecting 'fake query' with the real accesses in an intermediate proxy to smooth the distribution. Detailed comments of these related work can be found in Chapter 2.4. Note that, although the system throughput is largely enhanced, their drawbacks still exist in essence. Above all, it is inevitable to be concerned about the additional risk of compromise generated in such a proxy. Secondly, a majority of them use the mechanism of 'query batching' which could reveal access patterns of individual real requests when the timing boundary of query processing is identical to adversaries who are familiar with the obfuscation algorithm, particularly, when the arrival of requests is sparse. They usually mitigate this crux by imposing extra latency on handling the real query potentially identified in such cases, such as the batch pending applied in [41]. Thirdly, previous fake-query-based schemes and oblivious access protocols either merely support point queries (i.e., key-value searches) as limited by their mechanisms or specially designed for particular order-preserving encryption (OPE) to process range searches. However, both two query types are highly demanded in real-world query ser-

vices (e.g., relational databases).

Table 1: Access Frequency Hiding Schemes

Schemes	**Reveal Individual Request**	**Query Types**	**Deploy Requirements**	**Performance (latency)**
Yang et al.[112]	**No**	Non-fuzzy keyword search (point query)	1.Rebuild on index 2.Client local cache	Low
Path ORAM[96]	**No**	Split range query as individual point queries	1.Re-organize data storage 2.Client local cache	Low
Mavroforakis et al.[72]	Risk when query boundary disclosed	Range query over MOPE (Modular OPE)	1.Trusted proxy Modificatio on query processor	N/A
PANCAKE[41]	Risk when query boundary disclosed (requires batch pending)	KV-pairs query (point query)	Trusted proxy	**High**
The proposed	**No**	**Both point and range query**	**Only Intel SGX enabled host**	**High**

Motivated by the three problems above, I propose the ReFlat module, a novel privacy-preserving module that efficiently hides the access frequency pattern of both point and range queries in the compromised and query boundary disclosed memory (CQBDM) (as defined in Section 5.3.1) inheriting the idea of 'fake query'.

To eliminate the demand of intermediate proxy, the core process of the proposed obfuscation scheme is capsulate with the hardware enclave provided Intel SGX technology [52][51][25], and its program code is run alongside the protected query processor in the compromised cloud platform. As introduced in Chapter 3.1, even in a fully compro-

mised cloud, it still guarantees that the adversary can not observe the buffered data, the obfuscation algorithms as well as the incoming query requests. However, due to the limitation of enclave heap space and the high cost of page swapping, it is infeasible to load the entire data storage and the query processor into the enclave for processing. Concerning this, the ReFlat module is designed to cost very small runtime memory and pre-loads only schema information of the data storage as meta-data.

In ReFlat module, a query duplication mechanism, K**-duplication**, is used to fabricate the fake queries. Initially, the data storage is virtually divided into K isomorphic segments, named **duplication Structures**, in the Intel SGX enclave. Then, each query request on a particular data block is duplicated to the mapped blocks in the other $K-1$ duplication structures. Concerning this, $K-1$ fake queries are generated to smooth access frequency patterns not only for statistics of overall access times on each data block during a given period but also for individual requests. The user finally receives the **Slip** encrypted by his credential to distinguish which segments compose the result set of real queries and which source from dummy queries. Different from other fake-query-based schemes, the proposed scheme does not batch the query. As such, the access patterns of individual queries are preserved in the cases where query boundaries are already disclosed to attackers.

To build an integrated solution for point and range queries, two working functions of K-duplication mechanism are elaborated, namely the **Frequency Distortion Function (FDF)** and **Query Reconstruction Function (QRF)**. Both of them provide

additional security to the basic K-duplication mechanism. Through FDF, the access frequency patterns left by point queries are perturbed with extra dummies, and QRF further hides query scopes of range search.

As an overview, in Table 1, the proposed scheme is compared with representative access frequency pattern hiding schemes, including oblivious data access protocols and fake-query-based schemes, in four concerning aspects: information leakage of individual query in CQBDM, query types they are applicable to, additional requirements in deploying to query processor, and protocol performance. Among them, the remarks of performance are given based on the detailed experimental evaluations as seen in Section 5.5.

To summarize, the main contributions of this subject are as follows.

- To formally describe the ability of the attacker, I issue a rigorous threat model, that is, compromised and query boundary disclosed memory (CQBDM), and describe the levels of information leakage it may incur.

- I propose the obfuscation module, ReFlat, to conceal the access frequency pattern in a robust manner against the defined leakage in compromised and query boundary disclosed memory (CQBDM). Meanwhile significantly reduces the latency and storage cost than the existing schemes that achieve equal security goals.

- The design of ReFlat does not intervene with the query processors, and consequently preserves the functionality of sorted storage and the trapdoors of searchable encryption. Besides, it can be conveniently adapted to any outsourced database

offering point or range query interfaces. The implementation eliminates the need for the intermediate proxy which may incur extra security risk itself and does not transfer any data storage into the hardware enclave.

- Comparative experiments and thorough evaluations on system parameters over real-world datasets are performed to prove the claimed effectiveness and performance enhancement.

The rest of the chapter is organized as follows. In Section 5.2, I introduce the cryptographic primitive used in the design of access pattern obfuscation scheme. I present the threat model and security definitions, as well as the system model in Section 5.3 followed by the detailed introduction about the K-duplication mechanism and the security analysis in Section 5.4. Experimental results and discussions are shown in Section 5.5. Section 5.6 summaries this chapter. Some related notations in this chapter are summarized in Table 2.

5.2 Data Storage and Query Isomorphism

In this section, I introduce the concept of query isomorphism which is one fundamental principle of this work.

The algebraic concept and properties of group isomorphism [10][65] is widely applied in addressing structural attacks over sub-graph query [21][113][122]. Likewise, we can model the isomorphism of query request patterns as follows.

Table 2: Notations and Symbols

Symbol	Definition
K	# of duplication structures (duplication factor)
B	universal set of blocks in a duplication structure
Q_a	all the queries pending for processing of single request after being disposed in ReFlat
$T(SK, *)$	trapdoor function of query generated with key SK
N_p	# of dummy point queries attached in FDF
Pr_0	the probability of random attached dummies in FDF can be selected from 0-accessed blocks
M	# of segments in one duplication structure
R	the query scope of a range query
R_{Seg}	the index scope of a segment used in QRF
R_d	the expanded query scope of a duplication structure used in QRF
ω	# of accessed times of particular blocks
α	pre-configured replica threshold in PANCAKE

Definition 8. *Given two data structures G and H containing the same number of data blocks, $|G| = |H|$. D_G, D_H denote the unique identity of data block in G, H. G and H are isomorphic if either of the two conditions is satisfied:*

(1) G and H are unsorted data structures. There exists a bijection f between D_G and D_H, where D_G, D_H are the universal set of the data blocks in G, H.

(2) G and H are sorted data structures each with circular links between its two ends. Let (D_G, D'_G) be the consecutive range covering the data blocks from D_G to D'_G. Let $B_{(D_G, D'_G)}$ be the data blocks covered in range (D_G, D'_G). Condition (1) holds, and meanwhile $f(D_G, D'_G) \in D_H, f((B_{(D_G, D'_G)}) = B_{f((D_G, D'_G))}$ always hold for arbitrary referred blocks of D_G , D'_G.

Based on this definition, it is straightforward to obtain that each point or range query

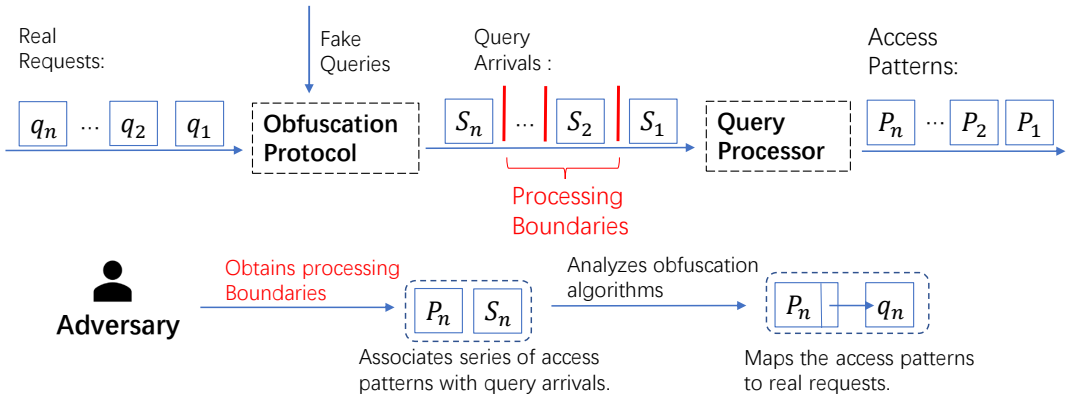


Figure 5.1: Threats of Compromised and Query Boundary Disclosed Memory

over G can be uniquely mapped to a query in H. This property is further applied in K-duplication mechanism, as presented in Section 5.4.

5.3 ReFlat Architecture

In this part, threat and security model as well as the system framework of ReFlat module are explained sequentially.

5.3.1 Threat Model

In the threat model of ReFlat, the encrypted data blocks are considered to be serialized on logical addresses. A search key on the searching attribute is applied to facilitate point queries or sorted to support range queries. In processing such queries, all the block-wise accesses can be accurately monitored by the adversary in compromised memory. Assume that the adversary possesses the knowledge of timing boundaries of each query,

i.e. starting points and endpoints of entire process over the timeline. That said, he always manages to associate a series of access patterns P_n captured from compromised memory with a query arrival S_n (as shown in Figure 5.1). In practice, this assumption is common to be reached when the real request arrivals are sparse or the data users are authorized to distinct portions of data storage in multi-user scenarios. In addition, the adversary acknowledges that query arrival S_n consists of a real query q_n and some fake queries generated from a known obfuscation protocol. Therefore, by grasping some specific logic in the given obfuscation algorithm, he could further successfully map a portion of access patterns in P_n to the real query q_n. For example, the last query of S_n in [72] is always the real request. To highlight the differences in adversaries' capability and target out of previous literature, the definition of compromised and query boundary disclosed memory (CQBDM) is given as below.

Definition 9. *(compromised and query boundary disclosed memory (CQBDM)) The entire memory level access patterns can be monitored by the adversary in both an aggregated perspective for sequence of queries and independent perspective for each individual query. The boundary of processing each real request is assumed to be disclosed to the adversary even though they are injected with fake queries as a sequence.*

Remark 3. *Although the memory is fully compromised, the adversary are unable to crack the encrypted structures.*

According to the definition, through convenient approaches or statistical tools, the adversaries manage to know: a) The number of access times for all the data blocks; b) The

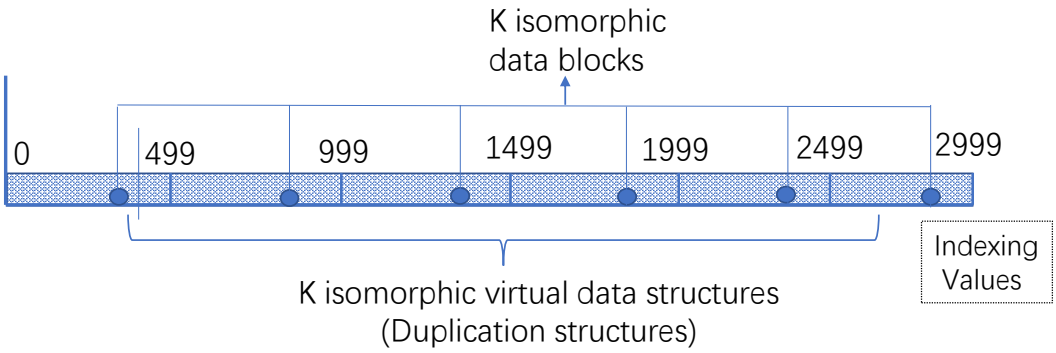


Figure 5.2: K-duplication Structures

logic of all memory operations (e.g. memory reference) associated with accessed blocks; c) A series of data accesses that generated by a sequence of queries is associated with a real request; d) Assume that he is managed to understand the algorithm and map it to the observed memory operations transparent; The model also assumes the attacker holds the auxiliary knowledge such as e) the natural request frequency patterns of the dataset in plaintext (e.g., flight with the lowest price are most frequently accessed).

Note that, the aim of an adversary over compromised and query boundary disclosed memory includes not only information of data storage, but also the information of individual query requests. Thus, the proposed threat model is also distinct from persistent active/passive adversary [63][41] in adversary's target.

5.3.2 Security Model

The proposed scheme aims to hide the actual request frequency patterns left by memory operations associated with queries. To clarify the security goals, four levels of leak-

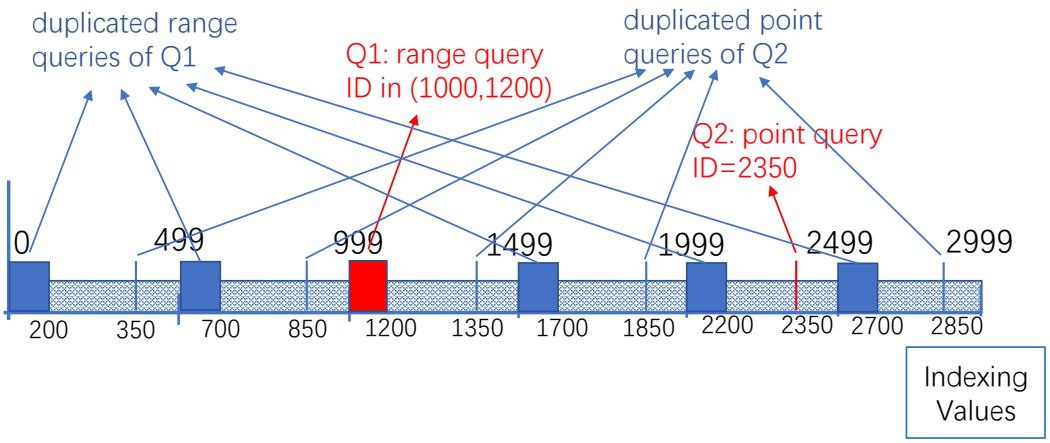


Figure 5.3: Query Duplication

age that can be obtained through analyzing the changes and statistics over the access frequency patterns are given as below.

- (L1-leakage): The pattern reveals the data blocks which are topmost accessed and their rank information.
- (L2-leakage): The adversary can identify the exact number of real access times of a particular data block.
- (L3-leakage): For individual point query requests over unsorted storage, from observing the change on request frequency pattern, the adversary knows which data block is actually requested or distinguishes whether two queries request for the same block.
- (L4-leakage): For individual range query requests over sorted storage, from observing the changes on request frequency patterns, the adversary can correctly find the requested ranges or successfully compare the scope of querying range of

each query.

Definition 10. *(K^*-Ordinary Request Frequency Pattern (ORFP)): In processing queries in full compromised cloud memory, a privacy-preserving protocol turns the access frequency pattern into k^*-Ordinary request frequency pattern $if.f$ the worst case occurrences of the above levels of leakage are all constrained under $\frac{1}{k}$.*

Remark 4. *Note that the CPA or IND-CPA approaches are beyond the scope of this work, i.e. the attackers are assumed to have no information about the real query contents of each particular query arrival. The security model also exclude other feasible side channel attacks, such as timing attacks.*

The principles of basic K-duplication mechanism and how it facilitates the ReFlat to satisfy the security goals is briefly demonstrated as follows.

As detailed in Section 5.4.1, the original data storage are initialized as K virtual structures bath on the indexing attribute values in the enclave of ReFlat module. The virtual structures are further initialized as isomorphic structures, i.e. duplication structures. As shown in Figure 5.2, each virtual block contained in a duplication structure can always be mapped to an isomorphic block sharing the same block-wise offset to its own head blocks in the other $K-1$ duplication structures. For ease of presentation, the K isomorphic blocks are named "mirrors". When each data block of the particular structure is queried (or covered by a range query), the block request is duplicated to its "mirrors". Consequently, the access frequency patterns in all the duplication structures are always isomorphic. As examples, the changes on request pattern of two queries (range query Q1

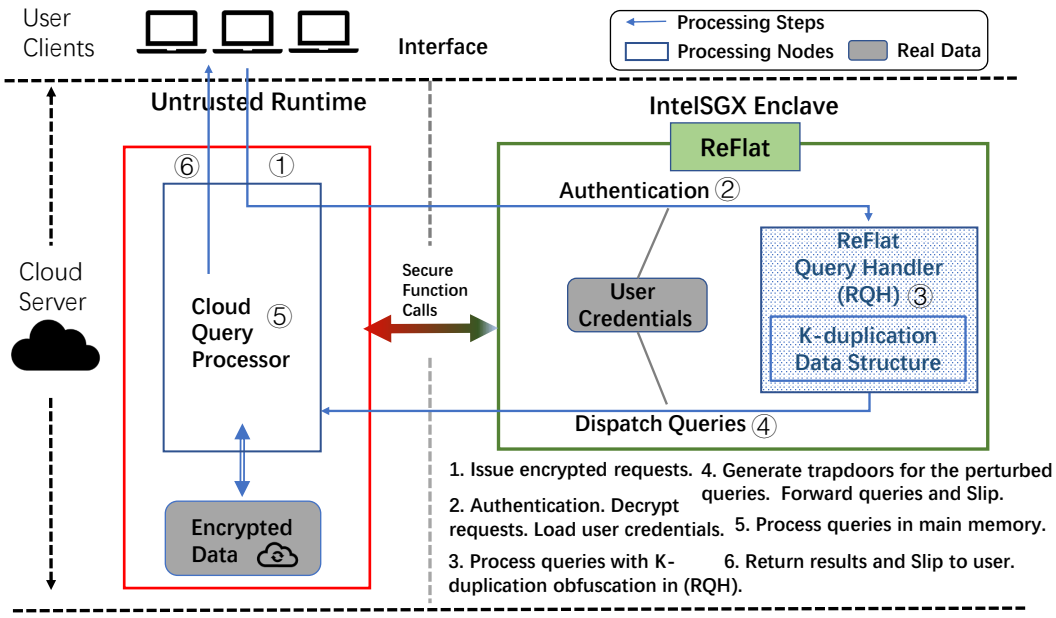


Figure 5.4: ReFlat System Design

and point query Q2) after processing the query duplication are depicted in Figure 5.3. ReFlat maintains a Frequency Snapshot (FS) displaying the real-time access frequency pattern of duplication structures to reflect the overall pattern of the data storage. From time to time, the request frequency pattern reflected on the Frequency Snapshot always synchronizes with the observation from the perspective of adversaries. Thus, based on the principles of the K-duplication structure and the working functions of K-duplication mechanism (as detailed in Section 5.4), the adversary needs a much higher random guess cost to distinguish the listed access pattern leakages. The theoretical proof that ORFP security is satisfied in ReFlat is provided in Section 5.4.4.

5.3.3 System Model

In ReFlat, the protocol workflow consists of three procedures, which are user authentication, ReFlat Query Handler, and Query Dispatch as shown in Figure 5.4. These steps run inside the Intel SGX enclave and exchange with the query processor in untrusted runtime through secure function calls provided by the SGX enclave.

User Authentication. After a remote attestation stage (omitted in Algorithm 7), user encrypt plaintext query Q with negotiated key K as $E(K, Q)$. ReFlat decrypts it to obtain Q inside enclave[1]. The user's credentials SK for performing searchable encryptions are also pre-fetched to enclave as meta-data.

ReFlat Query Handler (RQH). In this building block, both unsorted and sorted data storage is virtually divided into K-duplication Structures according to the preloaded meta-data describing the storage. The original query Q is processed through Frequency Distortion, K-duplication and Query Reconstruction based their types to generate dummy queries for perturbing the access frequency pattern.

Query Dispatch. All the queries including the dummy ones Q_a, before being forwarded to query processor in fully compromised memory, are encrypted with user credentials SK to generate trapdoors $T(SK, Q_a)$for processing under searchable encryptions. An encrypted Slip $E(K, S)$ is attached to notify the user which result sets of queries are used to restore the real query results of Q.

[1]Since the communication channel with user client is beyond the security boundary of ReFlat, no hash functions and cryptographic primitives are specified to be applied to the user queries.

Algorithm 7 and Figure 5.4 illustrate the workflow of processing a query with ReFlat module. Note that, for all the algorithm steps, there are no real data blocks or memory read/write instructions transmitted between ReFlat and untrusted query processor. The frequency snapshot is updated once a real or dummy query is issued. The point queries over unsorted storage and the range queries over sorted storage are handled in different K-duplication functions in ReFlat query handler.

Algorithm 7 ReFlat Work-flow

Input: Encrypted user request $E(K, Q)$
Output: Result Set and Encrypted Slip $RS \| E(K, S)$
1: Decrypts $E(K, Q)$ for Q in UA.
2: Searches enclave meta-data for user credentials SK.
3: RQH: Invokes $updateFS(Q)$ to Update Frequency Snapshot FS.
4: **if** Q is point query over unsorted storage **then**
5: RQH: Runs K-duplication under Frequency Distortion Function.
6: RQH: Expands Q with the generated dummy queries Q_d, $Q_a \leftarrow Q \bigcup Q_d$.
7: RQH: Invokes $updateFS(Q_a)$ and generate Slip $E(K, S)$.
8: **else**
9: RQH: Runs K-duplication under Query Reconstruction Function.
10: RQH: Expands Q with the generated dummy queries Q_d, $Q_a \leftarrow Q \bigcup Q_d$.
11: RQH: Invokes $updateFS(Q_a)$
12: **end if**
13: Sends queries Q_a and slip $E(K, S)$ to query processor.
14: Query Processor: Derives query results RS and returns $RS \| E(K, S)$.

5.4 K-duplication

In this section, the initialization process of duplication structure is introduced followed by the algorithm demonstrations of the two working functions of K-duplication, that is, Frequency Distortion Function(FDF) and Query Reconstruction Function(QRF). Then

I further explain how they coordinate to achieve the K^*-Ordinary Access Frequency Pattern (ORFP)(as seen in 5.4.4) in compromised and query boundary disclosed cloud memory.

5.4.1 Initialize K-duplication Structure

First, the schema information of encrypted data storage is pre-loaded into an enclave in plaintext as meta-data. This information is referenced for dividing the K-duplication Structures and building the Frequency Snapshot. For easy presentation, in the rest of this chapter, the trapdoors and indexes are assumed to be built over a consecutive numeric attribute of data blocks. The particular data attribute is named the indexing attribute and is used to uniquely represent corresponding virtual data blocks. Therefore, in the current design, the meta-data includes the total number of data blocks, the lower and upper bounds of the indexing attribute. ReFlat then evenly divides value span from the lower bound to the upper bound into K duplication structures in a virtual axis[2]. After that, each of the virtual blocks contained in a duplication structure is logically mapped to the "mirrors" to suffice the definition of isomorphic data structure (recall from Section 5.2).

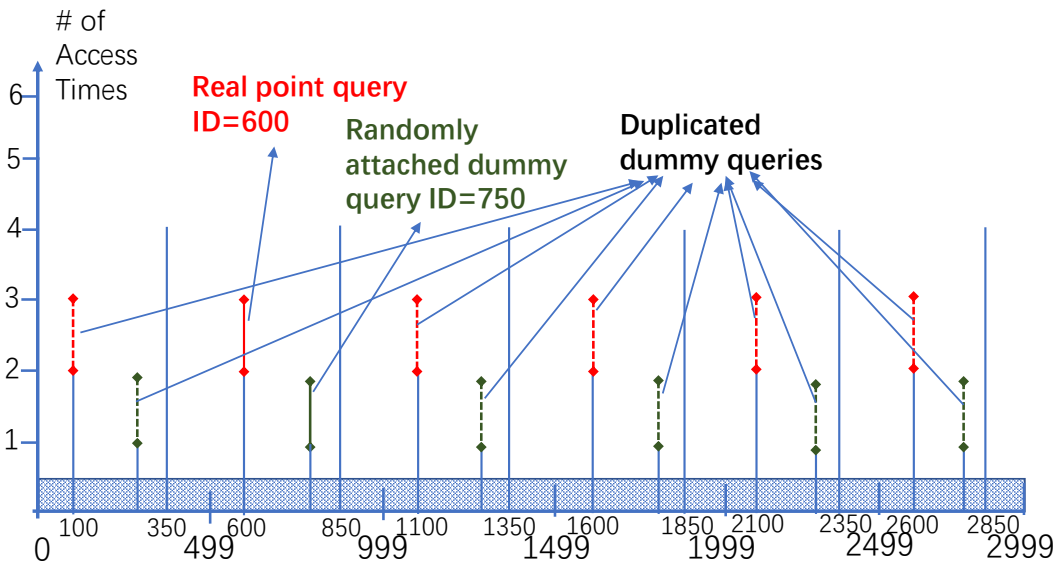


Figure 5.5: Frequency Distortion Function

5.4.2 Frequency Distortion Function

In this subsection, the Query Reconstruction Function (QRF) designed for processing point queries in ReFlat is demonstrated. Based on the query duplication mechanism introduced in the security model (as seen in Section 5.3.2), the preliminary K-duplication perturbs each ranking of access frequency with at least $K-1$ isomorphic block entities. However, it can not effectively hide the number of access times of a particular data block for point query Q_p. The reason is that the probability of two real point queries occurring at isomorphic blocks of different duplication structures is relatively low. Considering this, for each request, N_p dummy point queries are attached to the real query before being duplicated. The dummy queries are randomly selected from the blocks whose number of access times (ω) are larger than 0 based on the records of Frequency Snapshot (in

[2]The last structure is padded with faked value scope as the extension of the virtual axis to maintain the same scope as previous ones.

Figure 5.5). To prevent the real request on a data block that is never accessed from being distinguished in this setting, QRF imposes a small probability Pr_0 to let the dummy queries probably be selected from on the blocks whose accessed time remain as 0. Then both the real and dummy queries are duplicated into the rest $K-1$ duplication structures (as shown in Algorithm 8). Since the workload of query execution increase linearly with $K \cdot N_p$, in practice, K and N_p shall be configured according to the capability of the query processor.

Algorithm 8 K-duplication: Frequency Distortion Function

Input: real point query request: Q_p,
universal set of blocks in a duplication structure: B,
queries pending for duplicating $Q_C = \emptyset$.
Output: Perturbed request: Q^*
1: **if** $rand(0,1) \leq Pr_0$ **then**
2: Random select N_p blocks as Q_d from B into Q_C.
3: **else**
4: Remove the blocks with $\omega = 0$ from B.
5: Random select N_p blocks as Q_d from B into Q_C.
6: **end if**
7: $Q_1 = Q_C \bigcup Q_p$.
8: Duplicate Q_1 in all duplication structure.
$Q^* = Q_1 \bigcup Q_2 \bigcup ... \bigcup Q_K$.
9: Return Q^*.

5.4.3 Query Reconstruction Function

In this subsection, the Query Reconstruction Function (QRF) of K-duplication which is specialized for range queries over sorted storage is introduced. Compared with point query, range queries have a higher probability that the querying ranges of two requests that originally contain a common isomorphic range in different duplication structures

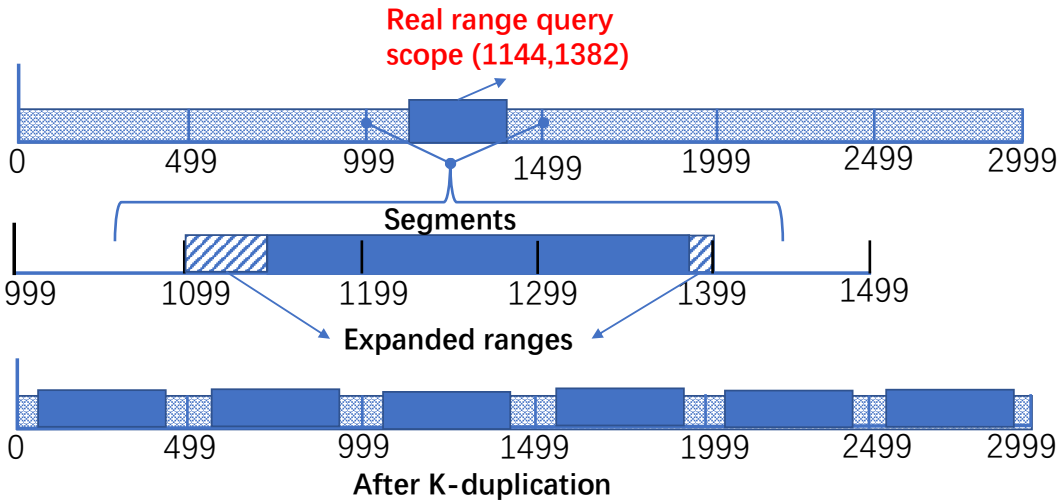


Figure 5.6: Query Reconstruction Function - First Stage

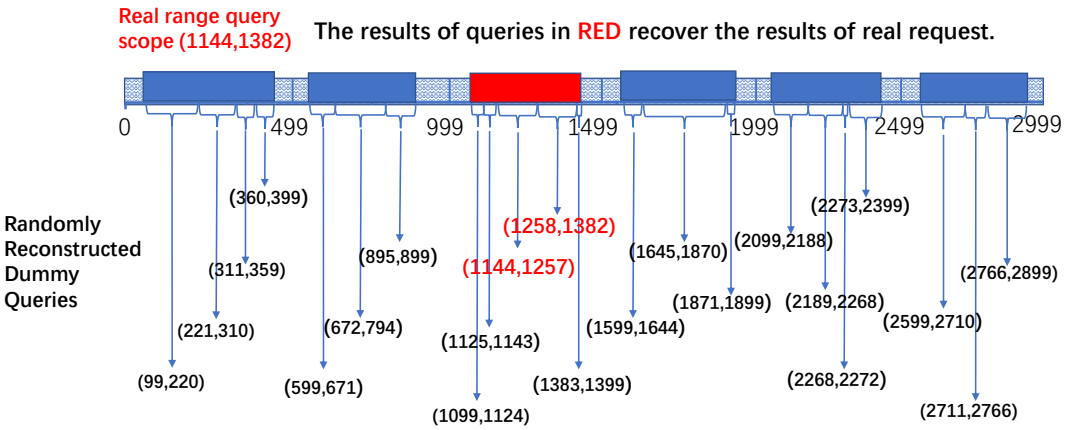


Figure 5.7: Query Reconstruction Function - Second Stage

to be overlapped after duplicating. However, it is obvious that the query ranges of distinct queries are less possible to be exactly the same in most cases. This phenomenon incurs the leakage on the query range of individual requests if the access frequency pattern is incrementally observed by adversaries. To overcome this, at the first stage, the duplication structures are split into M segments each of the same scopes of the indexing attribute. Then expand the query range R of real request Q_r that falls into a segment

Algorithm 9 K-duplication: Query Reconstruction Function

Input: real range query request: Q_r, its query range R. dummy range in a duplication structure $R_d = \emptyset$.
Output: Perturbed request: Q^*

```
Split each duplication structures into M segments.
for each duplication structure Z do
    for each segment Seg do
        if R_Seg ⋂ R ≠ ∅ then
            R_d = R_d ⋃ R_Seg.
        end if
    end for
end for
Execute K-Duplication and update R_d for each structure.
for each duplication structure Z do
    Segregate R_d into dummy queries Q_d.
    Q* = Q* ⋃ Q_d.
end for
Return Q*.
```

to be uniformed to cover the entire scope of the segment (as seen in Figure 5.6). At the second stage, after the expanded range is duplicated, the query range of each isomorphic structure is segregated randomly to reconstruct several dummy queries for processing (as seen in Figure 5.7). Note that, in segregating the ranges, it always guarantees that the real results of the requesting range can be recovered via the combination of the results of several fabricated queries. The combination is synchronized to user in the Slip. Algorithm 9 gives a step-wise demonstration of QRF.

5.4.4 Security Analysis

In this subsection, a theoretical analysis on the capability of access pattern obfuscation provided by ReFlat is displayed .

Theorem 6. *(Mitigation of L1-Leakage): ReFlat constrains L1-Leakage with probability under* $\frac{1}{K}$.

Proof. (*order*) At any time, the data blocks can be ordered by their access frequencies. From the perspective of adversaries, for each ranking in the order, there are nK blocks that have the same number of access times, due to the K-duplication mechanism. Therefore the probability of revealing the real order, Pr_{order}, suffices

$$Pr_{rankinformation} = \Pi_{i=1}^{m}(n_i K)^{-1} \leq \frac{1}{K} \tag{5.1}$$

, where m denotes the length of ordered data block sequence. □

Theorem 7. *(Mitigation of L2-Leakage): ReFlat constrains L2-Leakage with probability under* $\frac{1}{K}$.

Proof. (*number of access times*) After processing ν query requests (point/range queries) in compromised and query boundary disclosed memory, a data block is accesses for a times from the observation of adversaries. In ReFlat, each access may be caused by a dummy request produced by the duplication and frequency distortion (for point query). Thus, without further knowledge, the probability Pr_{times} that a is the real accessed time suffices

$$Pr_{times} = \frac{1}{a} \leq \frac{1}{K}. \tag{5.2}$$

, where $a \gg K$. □

Theorem 8. *(Mitigation of L3-Leakage): ReFlat constrains L3-Leakage with probability under* $\frac{1}{K}$.

Proof. (*requested block of individual query*) For each point query, N_p random dummy requests are generated to distort the access frequency. After executing K-duplication, $N_p \cdot K$ blocks are requested from the view of adversaries. The probability that he finds out the real requested block is

$$Pr_{point} = \frac{1}{N_p \cdot K} \leq \frac{1}{K}. \tag{5.3}$$

□

Theorem 9. *(Mitigation of L4-Leakage): ReFlat constrains L4-Leakage with probability under* $\frac{1}{K}$.

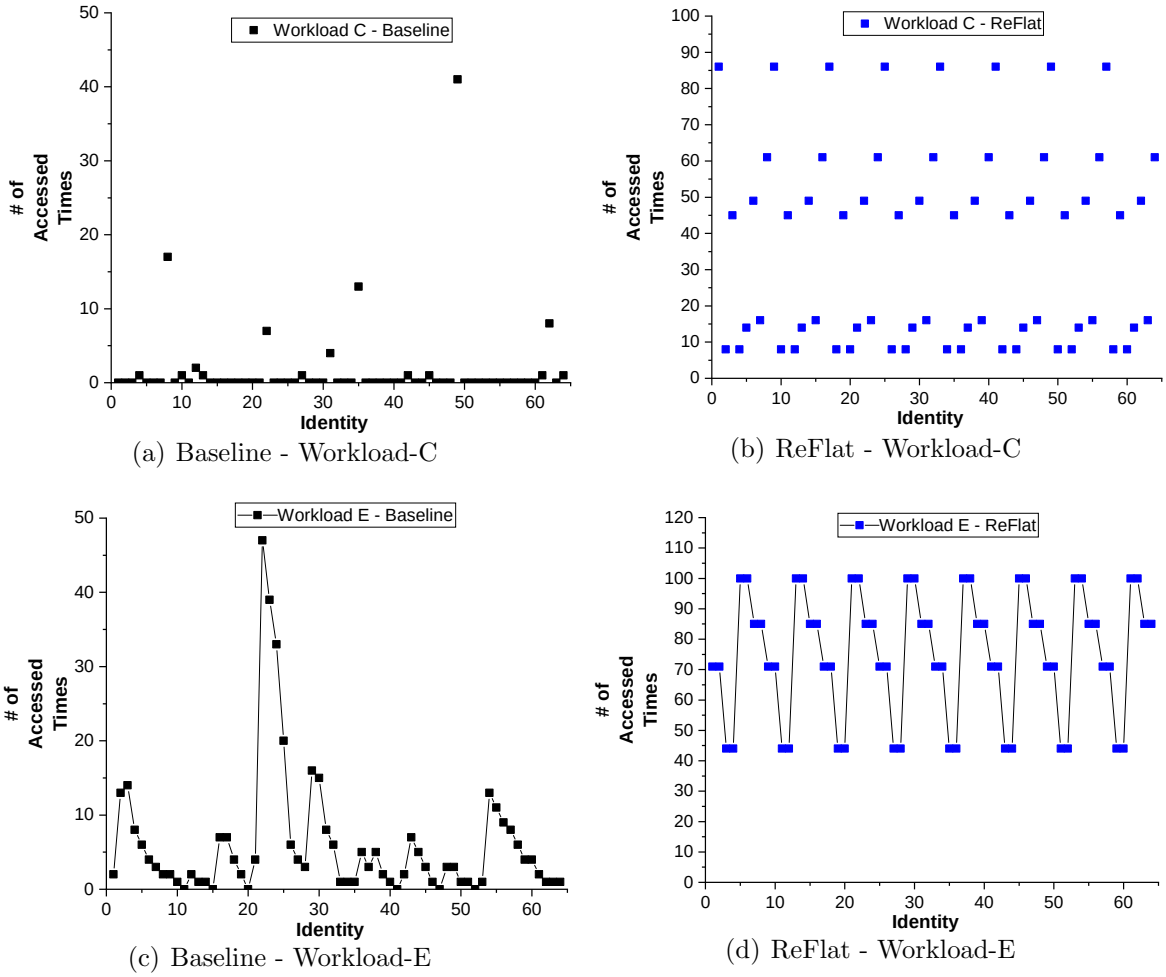
Proof. (*query range or range scope*) According to the Query Reconstruction Function, each range query is first expanded to align with the scope of the segment. Then for each of the K isomorphic structures, the expanded dummy ranges are randomly segregated to reconstruct N_{r_i} dummy queries. Only one combination out of the dummy queries can recover the real request range. Thus, the probability of recover the query range suffices

$$Pr_{range} = \Sigma_{i=1}^{K} \Sigma_{t=1}^{N_{r_i}} \binom{t}{N_{r_i}} \leq \frac{1}{K}. \tag{5.4}$$

□

Corollary 1. *ReFlat facilitates the query processing in compromised and query boundary disclosed cloud memory to achieve K^*-ORFP privacy preservation.*

Proof. Recall the Definition 10, since Theorem 6, 7 , 8, 9 hold for ReFlat protocol, we can say that K^*-ORFP is satisfied for the queries pre-processed in ReFlat. □



(a) Baseline - Workload-C

(b) ReFlat - Workload-C

(c) Baseline - Workload-E

(d) ReFlat - Workload-E

Figure 5.8: Perturbation on Request Frequency Patterns

5.5 Experimental Results

In this section, the experimental results of the proposed scheme are presented in 3 parts, namely the effectiveness of request frequency pattern hiding, the performance comparisons against other access frequency hiding schemes and the skewness on the internal settings of system parameters. All the experiments are conducted on a local machine (i.e. network latency ignored) with skylake i7-6700 CPU, 16G RAM. ReFlat program code is launched inside Intel SGX (v1.0) enclave instance running in the Hardware Mode. The "MaxStackSize" is pre-configured as 4MB for a single thread of ReFlat module throughout this section. To highlight the performance of access pattern hiding, all the data blocks and queries are processed in plaintext.

5.5.1 Effectiveness

In this part, ReFlat's security capability of frequency pattern obfuscation is evaluated using the YCSB dataset [24]. Every single record (data block) of the dataset is composed of 10 data fields, each with 100 bytes, and 8 bytes primary key. To illustrate the access frequency pattern over each block, 64 records are created and sorted by the primary key. The total size is around 63KB. I configure the two duplication functions of ReFlat as global duplication factor $K = 8$, distortion scaler $N_p = 2$ for frequency distortion function, the number of segments in each duplication structure $M = 4$ for query reconstruction function. For handling the basic point and range queries, a minimal

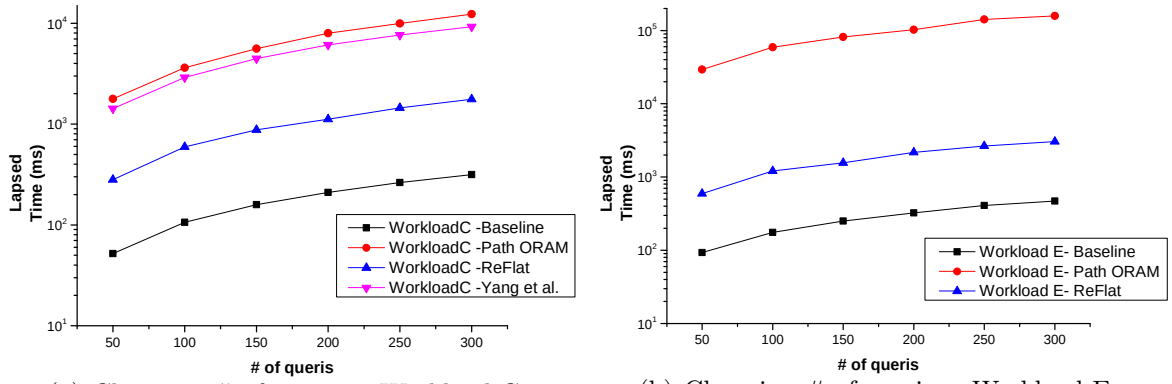
query processor is fabricated to resolve and execute such simple queries[3]. For point queries, it simply traverses the dataset to search for corresponding primary key. For range queries, it accesses the requested blocks by batch when the first block of the range is found.

W.r.t point query, 100 queries in the form of YCSB Workload-C are generated. For range query, YCSB Workload-E with 100 queries whose average querying scope is adjusted to around 4 records (blocks) is used. The adversarial program code counts the number of access times over each block based on whether it is actually included in the final result sets. The baseline solution for comparison directly process the query workload in the aforementioned tiny query processor. Figure 5.8(a) and (c) show the original access frequency patterns after executing the point and range queries through the baseline function respectively. The two figures reflect the same patterns as can be observed by adversaries in compromised and query boundary disclosed memory. Then process the same query workloads through ReFlat prior to the tiny query processor and plot the corresponding request frequency patterns in Figure 5.8(b)(d). It is obvious to find that, for each number of accessed times in the figure, there exist at least K blocks sharing the same value. Also, the access frequencies ω over individual blocks are randomly modified.

[3]The data storage is in plaintext to highlight the performance of access frequency pattern hiding mechanisms of all the schemes studied in the following experiments. They are all independent of what searchable encryptions are applied in the query processors.

5.5.2 ReFlat Performance

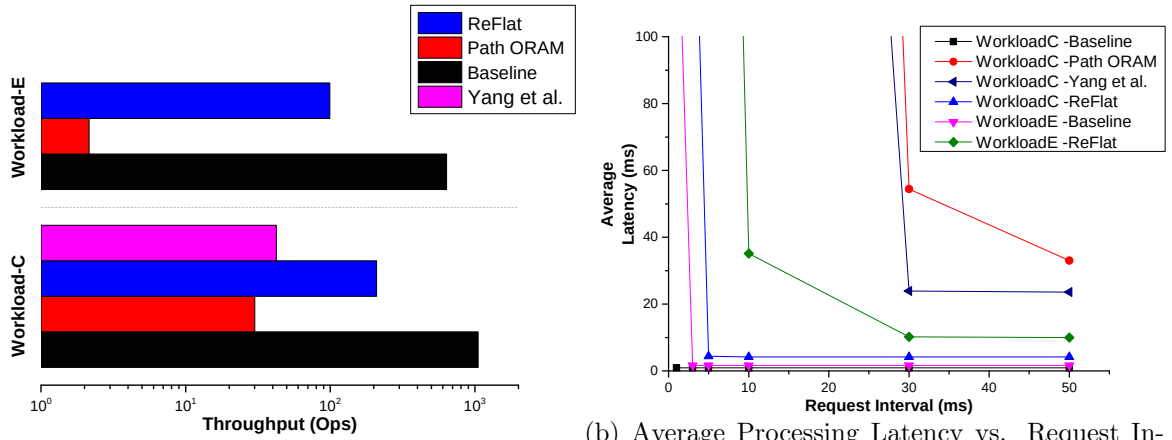
Comparing to Insecure Baseline and Oblivious Data Access Solutions



(a) Changing # of queries - Workload-C

(b) Changing # of queries - Workload-E

Figure 5.9: ReFlat vs. Baseline and oblivious data access schemes on processing time - changing # of queries



(a) Query Throughput

(b) Average Processing Latency vs. Request Interval

Figure 5.10: ReFlat vs. Baseline and oblivious data access schemes on overall performance

This part evaluates the performance of ReFlat in obfuscating the memory accesses by comparing it with bare insecure baseline (i.e. directly through the aforementioned tiny query processor) and two mainstream oblivious data access protocols, namely Yang et

al.'s storage shuffling schemes and the non-recursive Path ORAM [96]. The overall query processing time is recorded from receive time to the point when all query results are collected. Four experimental setups are prepared respectively for the insecure baseline, the proposed scheme, Yang et al.'s, and non-recursive Path ORAM. In processing through the insecure baseline, all the queries are executed in the aforementioned tiny query processor (as seen in 5.5.1) directly. In ReFlat setup, maintaining the same as the previous test, the query requests are handled by ReFlat with two working functions inside the Intel SGX enclave before they are executed in the tiny query processor. In the setup of Yang et al.'s approach, the primary keys of YCSB records act as the indexes of keywords database in its original design while searching. In both Yang et al.'s setup and Path ORAM setup, the server-side building blocks of their designs are embedded in the tiny query processor to replace the direct data retrieve operations. Their client-side building blocks are located in the local cache with the query processor so that the network delay can be ignored. Note that, the sorted indexes are not compatible with the security model of Path ORAM protocol, so the data accesses of range queries are treated as independent block accesses in Path ORAM setup.

In this experiment, the working size of YCSB dataset contains 128 records. In ReFlat setup, the duplication scale is configured $K = 16$, the distortion scaler is set to $N_p = 2$ for frequency distortion function, and the number of segments in each duplication structure is $M = 4$ for query reconstruction function. In Yang et al.'s setup, the number of lower-level index groups recorded in an index group is configured as $m = 4$ in building its

hierarchical index structure. In Path ORAM setup, the bucket size is set to $Z = 4$. For each trial, the same workload varying from 50 to 300 original queries are given to both setups. For point query, YCSB Workload-C is used. For range query, YCSB Workload-E is used and the average querying scope is adjusted to around 15. Note that, the trial runs of YCSB Workload-E are excluded in Yang et al.'s setup, since Workload-E is composed of range queries while the Yang et al.'s scheme is designed for point keyword queries.

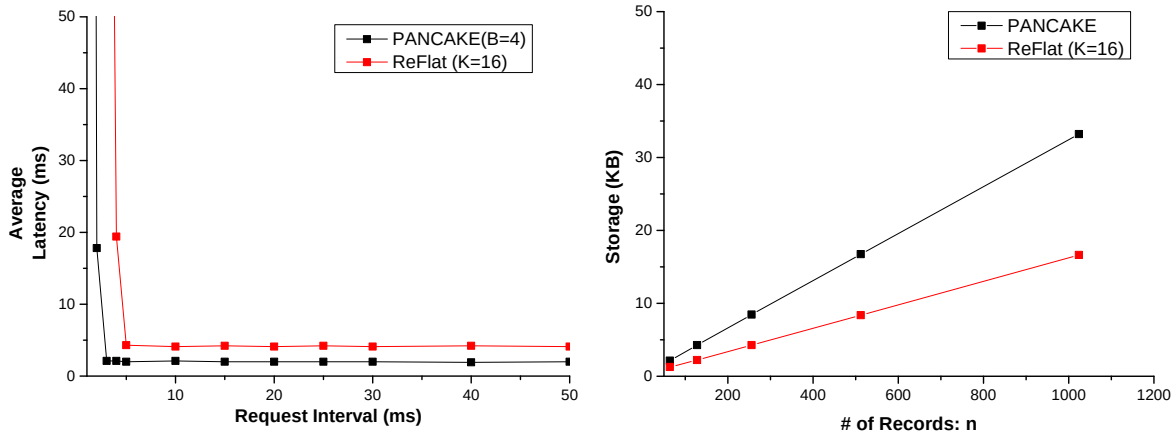
As plotted in Figure 5.9(a)(b), compared with the baseline, non-recursive Path ORAM is $34x - 39x$ slower on average than the insecure baseline in dealing with point queries and $295x - 346x$ slower in the range queries. Yang et al.'s approach shows $26x - 30x$ slower on average for point queries. With moderate duplication factor, $K = 16$, ReFlat outperforms Path ORAM with $6.7x$ faster on average for point query and $47.1x$ average faster for range queries reaching $5.3 - 5.7x$ and $6.2 - 6.7x$ response time of the baseline respectively for point and range queries. In comparison with the insecure baseline, the processing speed is moderate and acceptable for application scenarios without strict latency requirement. The higher efficiency yields in processing range queries can be attributed to the preservation of sorted indexes in ReFlat mechanism. It can be inferred that the increasing average querying scope will scale up the advantage of ReFlat in processing range queries. In Figure 5.10(a), the system throughput (single thread) of the four setups are exhibited in colored curves. In Figure 5.10(b), the average processing latency for each query are demonstrated with the request intervals varying from 1ms to 50ms. The latencies of the Path ORAM setup over Workload-E are hidden, due to the

throughput of processing range query with Path ORAM being extremely low.

Comparing to Existing Fake Query Injection Solution

To the best of my knowledge, ReFlat is the only wrapped up solution that provides request pattern hiding in the CQBDM threat model without the aid of stand-along proxy (with unknown security primitives and guarantees) or deferring the processing of queries. To evaluate the performance with state-of-the-art schemes, I reforge a sketched read-only prototype of PANCAKE and pre-set the query batching size as $B = 4$. Since a YCSB dataset with 2^8 records (i.e., KV-pairs, in PANCAKE) is used, thus it is reasonable to fix the replica threshold $\alpha = \frac{1}{n} = 1/128$. The settings of ReFlat remain the same as in Section 5.5.2. Both ReFlat and PANCAKE proxy are run in single thread mode. Since PANCAKE only supports point query, YCSB Workload-C is applied in the evaluation, and the average processing latency for each query is measured with time intervals between individual requests changing from 1ms to 50ms. The evaluation results are plotted in Figure 5.11(a).

For PANCAKE, the latency drops at the beginning (2ms interval trials) and turns to be stable alongside the increase of query interval reflecting the optimal throughput capability is near 500 operations per second under the settings here. For ReFlat, the latency sharply goes down with the increase of interval, and then levels off after reaching 5ms which confirms the average processing time for individual query in underlying settings. As illustrated in stable scope on the two curves, PANCAKE is $2.1x$ faster than the



(a) Average Processing Latency vs. Request Interval - YCSB Workload-C

(b) Protocol Storage Cost

Figure 5.11: ReFlat vs. PANCAKE

proposed ReFlat module.

I also evaluate the stand-alone storage cost both of schemes. PANCAKE uses fixed footprint to accommodate the $2n$ replicas of keys, each with 8 bytes. In ReFlat, the implementation uses 4 bytes to label the n records and the K duplication structures (fixed as $K = 16$). As shown in Figure 5.11(b), without enabling the UpdateCache, PANCAKE requires up to $2x$ storage space in the proxy than the usage of ReFlat in the enclave when n/K increases.

5.5.3 Changing value of K

With the changing value of duplication factor K, the processing time of 100 queries for both point (YCSB Workload-C) and range queries (YCSB Workload-E) is recorded. In processing the queries, YCSB dataset and the other parameters of both two working functions maintain the same value as previous experiments. Let K vary from 8 to 40

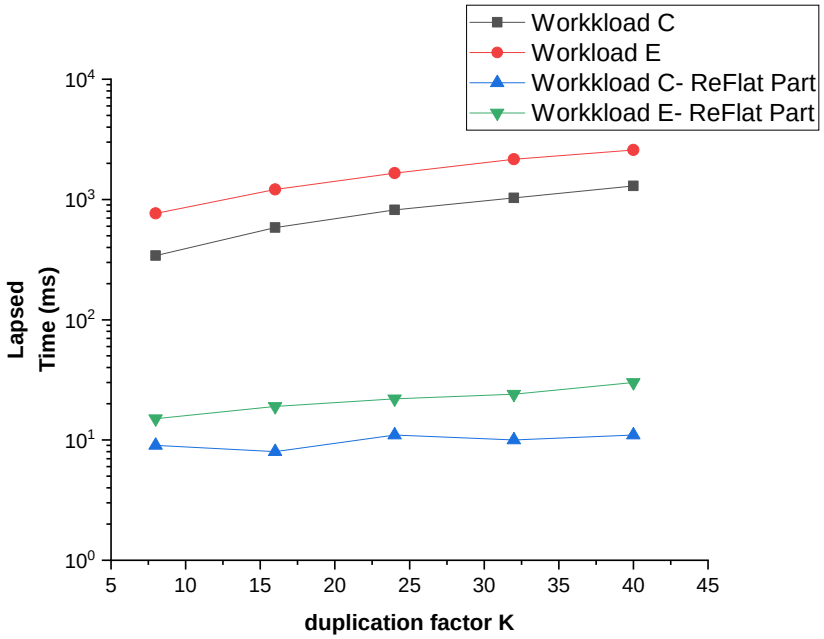


Figure 5.12: Overall and ReFlat Response time on Changing K

and calculate the amortized processing time for each query request. The changes on the amortized ReFlat response time are also collected for each request. As reflected in Figure 5.12, on one hand, the amortized time cost of processing single query shows a linear growth with increasing K for both point and range queries. This observation is in accordance with the increase in the actual number of dummy queries processed. On the other hand, the amortized response time of ReFlat module remains stable along with the growth of K, while slightly increasing in running range queries. The reason for the difference is that the increasing number of duplication structures leads to extra time usage for reconstructing the dummy queries over the duplicate ranges.

5.6 Summary

In this chapter, I formally defined the access frequency leakage carrying sensitive information of query requests that can be obtained in compromised and query boundary disclosed memory in the cloud environment. Then I propose the counter solution, namely ReFlat, whose K-duplication mechanism notably mitigates the leakage for point and range queries over encrypted data storage. Running inside the Intel SGX enclave, ReFlat realizes K^*-ORFP privacy preservation without modifying the underlying query processor and eliminates the security risk of involving intermediate proxy. ReFlat significantly outperforms ORAM solutions on overall efficiency and simultaneously achieves higher robustness and smaller cost on extra buffer space than existing schemes based on fake query injection.

Chapter 6

Conclusions

The thesis thoroughly studies the current progress of privacy-preserving query processing from aspects of both idea and methodology. From the massive survey works, I carefully select three existing research problems to explore and provide my research outcomes by proposing robust and practical solutions. In each proposed scheme, the threat model and security model are clearly stated, and both the theoretical analysis and experimental proofs are given to prove the claimed security capabilities. The evaluations of system performance are conducted with simulations, benchmarks and real-world dataset tests. The following sections generally conclude the three subjects and list the potential improvements and extensions to be completed in the future.

6.1 Conclusion of Subjects

A. Memory-secure Database Adaptation In this subject, I first demonstrate that even with the most recent hardware-based security technology such as Intel SGX, a hypervisor can still sniff key database operations running in its guest virtual machine (VM) such as the frequency and type of SQL queries, by monitoring the access pattern of this VM's main and secondary memory. To ensure security against such access pattern monitoring attacks, I then propose ProDB, a minimal adaptation of a conventional DBMS with both hardware enclave and Oblivious RAM protocol. To enhance its performance for practical use, I also design a SQL-aware Path ORAM protocol called SaP ORAM, which optimizes the classic Path ORAM protocol under practical database workload. Through security analysis and extensive experimental results, I prove and show ProDB achieves high security and throughput on commodity cloud hosting servers.

B. Privacy-preserving Fuzzy Keyword Search To enhance memory-level security of multi-keyword fuzzy search, a widely occurred query request, I take the initiative to apply TEE technology to our protocol design which provides hardware-based tamper-proof enclaves. Then I propose the Edit Distance-based Obfuscation Scheme to further protect the query process executed outside TEE against access pattern leakage. With concerns of practicality and performance, I also propose the two-layer fuzzy index structure and Trend-aware Cache. The former addresses the space limitation of TEE memory for searching large datasets, while the latter optimizes the cache utility of TEE with a trend-aware coordinator to effectively reduce the communication overhead.

C. Access Frequency Pattern Hiding for General Query Processors In this work, I study the threats model in which adversaries know both the exact in-memory flow of accessed blocks and the processing boundary of each request. Under these settings, he can precisely observe the access frequency patterns in both aggregated and independent perspectives over queries. Then I propose the ReFlat module as a counter solution through the K-duplication obfuscation mechanism. ReFlat securely runs inside the hardware enclave provided by Intel SGX and requires no modifications on query processors. The K-duplication mechanism is further optimized with two working functions to practically deal with point and range queries. Compared with the state-of-the-art schemes using the similar idea, that is, fake query injection, ReFlat eliminates the security risk of involving intermediate proxy and achieves higher robustness under the proposed threat model. I exhibit comparative experiment results showing that ReFlat exceeds existing schemes providing equal security level in multiple system performance metrics.

6.2 Future Work

First, the current TEE technology used in my studies is Intel SGXv1, while in the next version of Intel SGX, namely SGXv2, the larger and expandable EPC is expected to be enabled. When SGXv2 is maturely supported in CPU and OS, the proposed systems shall be re-designed to make full use of the feature and the overall performance shall be re-evaluated. Secondly, the schemes proposed for privacy-preserving query processing do not consider the data/index updating. Therefore, in the future, I would like to complete

the algorithms to support dynamic data for higher applicability in existing applications. Thirdly, I plan to engage in the privacy issues of other types of query and data services. For example, one of my ongoing research works is about privacy-preserving data oracle of a smart contract. It provides a desensitization function inside the SGX enclave to allow contract owners to truncate sensitive information from retrieved external data before attaching it to the transactions.

References

[1] Daniel J Abadi. Data management in the cloud: Limitations and opportunities. *IEEE Data Eng. Bull.*, 32(1):3–12, 2009.

[2] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Order preserving encryption for numeric data. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 563–574, 2004.

[3] Adil Ahmad, Kyungtae Kim, Muhammad Ihsanulhaq Sarfaraz, and Byoungyoung Lee. Obliviate: A data oblivious file system for intel sgx. In *25th Annual Network and Distributed System Security Symposium, NDSS*, 2018.

[4] Kazi Wasif Ahmed, Md Momin Al Aziz, Md Nazmus Sadat, Dima Alhadidi, and Noman Mohammed. Nearest neighbour search over encrypted data using intel sgx. *Journal of Information Security and Applications*, 54:102579, 2020.

[5] A. Arasu, K. Eguro, R. Kaushik, and R. Ramamurthy. Querying encrypted data. In *Proceedings of 2014 ACM SIGMOD Conference*, pages 1259–1261, 2014.

[6] ARM.com. Arm trustzone.

[7] Nancy Arya, Mukesh Gidwani, and Shailendra Kumar Gupta. Hypervisor security-a major concern. *International Journal of Information and Computation Technology*, 3(6):533–538, 2013.

[8] Gbadebo Ayoade, Amir El-Ghamry, Vishal Karande, Latifur Khan, Mohammed Alrahmawy, and Magdi Zakria Rashad. Secure data processing for iot middleware systems. *The Journal of Supercomputing*, 75(8):4684–4709, 2019.

[9] Sumeet Bajaj and Radu Sion. Trusteddb: A trusted hardware-based database with privacy and data confidentiality. *IEEE Transactions on Knowledge and Data Engineering*, 26(3):752–765, 2014.

[10] Ross A Beaumont. Groups with isomorphic proper subgroups. *Bulletin of the American Mathematical Society*, 51(6):381–387, 1945.

[11] Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[12] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public key encryption with keyword search. In *International conference on the theory and applications of cryptographic techniques*, pages 506–522. Springer, 2004.

[13] Christoph Bösch, Richard Brinkman, Pieter Hartel, and Willem Jonker. Conjunctive wildcard search over encrypted data. In *Workshop on Secure Data Management*, pages 114–127. Springer, 2011.

[14] Ferdinand Brasser, Urs Müller, Alexandra Dmitrienko, Kari Kostiainen, Srdjan Capkun, and Ahmad-Reza Sadeghi. Software grand exposure: Sgx cache attacks are practical. In *11th USENIX Workshop on Offensive Technologies (WOOT 17)*, 2017.

[15] Helena Brekalo, Raoul Strackx, and Frank Piessens. Mitigating password database breaches with intel sgx. In *SysTEX@ Middleware*, pages 1–1, 2016.

[16] Jin Wook Byun, Hyun Suk Rhee, Hyun-A Park, and Dong Hoon Lee. Off-line keyword guessing attacks on recent keyword search schemes over encrypted data. In *Workshop on Secure Data Management*, pages 75–83. Springer, 2006.

[17] Huikang Cao, Ruixuan Li, Wenlong Tian, Zhiyong Xu, and Weijun Xiao. Blockchain-based accountability for multi-party oblivious ram. *Journal of Parallel and Distributed Computing*, 137:224–237, 2020.

[18] Anrin Chakraborti and Radu Sion. Concuroram: High-throughput stateless parallel multi-client oram. *arXiv preprint arXiv:1811.04366*, 2018.

[19] Yuezhi Che and Rujia Wang. Multi-range supported oblivious ram for efficient block data retrieval. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 369–382. IEEE, 2020.

[20] Yaxing Chen, Qinghua Zheng, Zheng Yan, and Dan Liu. Qshield: Protecting outsourced cloud data queries with multi-user access control based on sgx. *IEEE Transactions on Parallel and Distributed Systems*, 32(2):485–499, 2020.

[21] James Cheng, Ada Wai-chee Fu, and Jia Liu. K-isomorphism: privacy preserving network publication against structural attacks. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 459–470, 2010.

[22] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on*, pages 41–50. IEEE, 1995.

[23] M Chuah and W Hu. Privacy-aware bedtree based solution for fuzzy multi-keyword search over encrypted data. In *2011 31st International Conference on Distributed Computing Systems Workshops*, pages 273–281. IEEE, 2011.

[24] Brian F Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 143–154, 2010.

[25] Victor Costan and Srinivas Devadas. Intel sgx explained. *IACR Cryptology ePrint Archive*, 2016:86, 2016.

[26] Victor Costan and Srinivas Devadas. Intel sgx explained. *IACR Cryptology ePrint Archive*, 2016(086):1–118, 2016.

[27] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. *Journal of Computer Security*, 19(5):895–934, 2011.

[28] Srinivas Devadas, Marten van Dijk, Christopher W Fletcher, Ling Ren, Elaine Shi, and Daniel Wichs. Onion oram: A constant bandwidth blowup oblivious ram. In *Theory of Cryptography Conference*, pages 145–174. Springer, 2016.

[29] Shugeng Ding, Yidong Li, Jianhui Zhang, Liang Chen, Zhen Wang, and Qunqun Xu. An efficient and privacy-preserving ranked fuzzy keywords search over encrypted cloud data. In *2016 International Conference on Behavioral, Economic and Socio-cultural Computing (BESC)*, pages 1–6. IEEE, 2016.

[30] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17(3):449–467, 1965.

[31] Saba Eskandarian and Matei Zaharia. Oblidb: Oblivious query processing using hardware enclaves. *arXiv preprint arXiv:1710.00458*, 2017.

[32] Saba Eskandarian and Matei Zaharia. An oblivious general-purpose sql database for the cloud. *CoRR abs/1710.00458*, 2017.

[33] Christopher W Fletcher, Muhammad Naveed, Ling Ren, Elaine Shi, and Emil Stefanov. Bucket oram: Single online roundtrip, constant bandwidth oblivious ram. *IACR Cryptology ePrint Archive*, 2015:1065, 2015.

[34] Zhangjie Fu, Kui Ren, Jiangang Shu, Xingming Sun, and Fengxiao Huang. Enabling personalized search over encrypted outsourced data with efficiency improvement. *IEEE transactions on parallel and distributed systems*, 27(9):2546–2559, 2015.

[35] Zhangjie Fu, Xinle Wu, Chaowen Guan, Xingming Sun, and Kui Ren. Toward efficient multi-keyword fuzzy search over encrypted outsourced data with accuracy improvement. *IEEE Transactions on Information Forensics and Security*, 11(12):2706–2716, 2016.

[36] Zvi Galil. Efficient algorithms for finding maximum matching in graphs. *ACM Computing Surveys (CSUR)*, 18(1):23–38, 1986.

[37] Oded Goldreich. Towards a theory of software protection and simulation by oblivious rams. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 182–194. ACM, 1987.

[38] Michael T Goodrich, Michael Mitzenmacher, Olga Ohrimenko, and Roberto Tamassia. Privacy-preserving group data access via stateless oblivious ram simulation. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 157–167. SIAM, 2012.

[39] Pascal Gremaud, Arnaud Durand, and Jacques Pasquier. Privacy-preserving iot cloud data processing using sgx. In *Proceedings of the 9th International Conference on the Internet of Things*, pages 1–4, 2019.

[40] Alexey Gribov, Dhinakaran Vinayagamurthy, and Sergey Gorbunov. Stealthdb: a scalable encrypted database with full sql query support. *arXiv preprint arXiv:1711.02279*, 2017.

[41] Paul Grubbs, Anurag Khandelwal, Marie-Sarah Lacharité, Lloyd Brown, Lucy Li, Rachit Agarwal, and Thomas Ristenpart. Pancake: Frequency smoothing for encrypted data stores. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*, pages 2451–2468, 2020.

[42] Nils Gruschka and Meiko Jensen. Attack surfaces: A taxonomy for attacks on cloud services. In *2010 IEEE 3rd international conference on cloud computing*, pages 276–279. IEEE, 2010.

[43] Yanzhang He, Rohit Prabhavalkar, Kanishka Rao, Wei Li, Anton Bakhtin, and Ian McGraw. Streaming small-footprint keyword spotting using sequence-to-sequence models. In *2017 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, pages 474–481. IEEE, 2017.

[44] Vagelis Hristidis, Yannis Papakonstantinou, and Luis Gravano. Efficient ir-style keyword search over relational databases. In *Proceedings 2003 VLDB Conference*, pages 850–861. Elsevier, 2003.

[45] Tianlin Huo, Xiaoni Meng, Wenhao Wang, Chunliang Hao, Pei Zhao, Jian Zhai, and Mingshu Li. Bluethunder: A 2-level directional predictor based side-channel attack against sgx. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 321–347, 2020.

[46] IBM. Ibm 4758 pci cryptographic coprocessor general information manual.

[47] Ayad Ibrahim, Hai Jin, Ali A Yassin, and Deqing Zou. Secure rank-ordered search of multi-keyword trapdoor over encrypted cloud data. In *2012 IEEE Asia-Pacific Services Computing Conference*, pages 263–270. IEEE, 2012.

[48] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613, 1998.

[49] Intel. Intelr 64 and ia-32 architectures software developer's manual, sep 2015.

[50] Intel. Software guard extensions programming reference,2014.

[51] Intel Corporation. Intel(r) software guard extensions developer guide.

[52] Intel Corporation. Intel(r) software guard extensions (intel(r) sgx).

[53] Intel Corporation. Kvm sgx, 2017.

[54] Intel.com. Intel(r) architecture instruction set extensions programming reference.

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

[56] Mohammad Saiful Islam, Mehmet Kuzu, and Murat Kantarcioglu. Inference attack against encrypted range queries on outsourced databases. In *Proceedings of the 4th ACM conference on Data and application security and privacy*, pages 235–246, 2014.

[57] Zhang Jinsheng, Zhang Wensheng, and Daji Qiao. A multi-user oblivious ram for outsourced data. *Computer Science Technical Reports*, 262, 2014.

[58] Seny Kamara, Charalampos Papamanthou, and Tom Roeder. Dynamic searchable symmetric encryption. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 965–976, 2012.

[59] Balachandra Reddy Kandukuri, Atanu Rakshit, et al. Cloud security issues. In *Services Computing, 2009. SCC'09. IEEE International Conference on*, pages 517–520. IEEE, 2009.

[60] Nikolaos P Karvelas, Andreas Peter, and Stefan Katzenbeisser. Blurry-oram: a multi-client oblivious storage architecture. *Cryptology ePrint Archive*, 2016.

[61] Deokjin Kim, Daehee Jang, Minjoon Park, Yunjong Jeong, Jonghwan Kim, Seokjin Choi, and Brent Byunghoon Kang. Sgx-lego: Fine-grained sgx controlled-channel attack and its countermeasure. *computers & security*, 82:118–139, 2019.

[62] Vaibhav Kulkarni, Bertil Chapuis, and Benoît Garbinato. Privacy-preserving location-based services by using intel sgx. In *Proceedings of the First International Workshop on Human-centered Sensing, Networking, and Systems*, pages 13–18, 2017.

[63] Marie-Sarah Lacharité, Brice Minaud, and Kenneth G Paterson. Improved reconstruction attacks on encrypted data using range query leakage. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 297–314. IEEE, 2018.

[64] Shangqi Lai, Sikhar Patranabis, Amin Sakzad, Joseph K Liu, Debdeep Mukhopadhyay, Ron Steinfeld, Shi-Feng Sun, Dongxi Liu, and Cong Zuo. Result pattern hiding searchable encryption for conjunctive queries. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 745–762, 2018.

[65] FD Lestari, M Hafiyusholeh, AH Asyhar, WD Utami, and AZ Arifin. Properties of k-isomorphism on k-algebra. In *Journal of Physics: Conference Series*, volume 1211, page 012053. IOP Publishing, 2019.

[66] Jin Li, Qian Wang, Cong Wang, Ning Cao, Kui Ren, and Wenjing Lou. Fuzzy keyword search over encrypted data in cloud computing. In *2010 Proceedings IEEE INFOCOM*, pages 1–5. IEEE, 2010.

[67] Ming Li, Shucheng Yu, Ning Cao, and Wenjing Lou. Authorized private keyword search over encrypted data in cloud computing. In *2011 31st International Conference on Distributed Computing Systems*, pages 383–392. IEEE, 2011.

[68] Weixin Liang, Kai Bu, Ke Li, Jinhong Li, and Arya Tavakoli. Memcloak: Practical access obfuscation for untrusted memory. In *Proceedings of the 34th Annual Computer Security Applications Conference*, pages 187–197, 2018.

[69] Junwei Luo, Xuechao Yang, and Xun Yi. Sgx-based users matching with privacy protection. In *Proceedings of the Australasian Computer Science Week Multiconference*, pages 1–9, 2020.

[70] Yuchuan Luo, Xiaohua Jia, Huayi Duan, Cong Wang, Ming Xu, and Shaojing Fu. pride: private ride request for online ride hailing service with secure hardware enclave. In *Proceedings of the International Symposium on Quality of Service*, pages 1–10, 2019.

[71] Mark Mayzner. English letter frequency counts: Mayzner revisited or etaoin srhldcu.

[72] Charalampos Mavroforakis, Nathan Chenette, Adam O'Neill, George Kollios, and Ran Canetti. Modular order-preserving encryption, revisited. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 763–777, 2015.

[73] Travis Mayberry, Erik-Oliver Blass, and Agnes Hui Chan. Efficient private file retrieval by combining oram and pir. In *NDSS*. Citeseer, 2014.

[74] Pratyush Mishra, Rishabh Poddar, Jerry Chen, Alessandro Chiesa, and Raluca Ada Popa. Oblix: An efficient oblivious search index. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 279–296. IEEE, 2018.

[75] Mohammad H Mofrad, Adam Lee, and Spencer L Gray. Leveraging intel sgx to create a nondisclosure cryptographic library. *arXiv preprint arXiv:1705.04706*, 2017.

[76] Olga Ohrimenko, Michael T Goodrich, Roberto Tamassia, and Eli Upfal. The melbourne shuffle: Improving oblivious storage in the cloud. In *International Colloquium on Automata, Languages, and Programming*, pages 556–567. Springer, 2014.

[77] Rafail Ostrovsky and Victor Shoup. Private information storage. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 294–303. ACM, 1997.

[78] Simon Oya and Florian Kerschbaum. Hiding the access pattern is not enough: Exploiting search pattern leakage in searchable encryption. In *30th USENIX Security Symposium (USENIX Security 21)*, 2021.

[79] Hwee Hwa Pang, Xiaokui Xiao, and Jialie Shen. Obfuscating the topical intention in enterprise text search. In *2012 IEEE 28th International Conference on Data Engineering*, pages 1168–1179. IEEE, 2012.

[80] Diego Perez-Botero, Jakub Szefer, and Ruby B Lee. Characterizing hypervisor vulnerabilities in cloud computing servers. In *Proceedings of the 2013 international workshop on Security in cloud computing*, pages 3–10. ACM, 2013.

[81] Shauna Michelle Policicchio and Attila A Yavuz. Preventing memory access pattern leakage in searchable encryption. *iConference 2015 Proceedings*, 2015.

[82] Christian Priebe, Kapil Vaswani, and Manuel Costa. Enclavedb: A secure database using sgx. In *EnclaveDB: A Secure Database using SGX*, page 0. IEEE.

[83] Radu Prodan and Simon Ostermann. A survey and taxonomy of infrastructure as a service and web hosting cloud providers. In *Grid Computing, 2009 10th IEEE/ACM International Conference on*, pages 17–25. IEEE, 2009.

[84] S Raghavendra, CM Geeta, K Shaila, Rajkumar Buyya, KR Venugopal, SS Iyengar, and LM Patnaik. Msss: most significant single-keyword search over encrypted cloud data. In *Proceedings of the 6th annual intrernational conference on ICT: BigData, Cloud and Security*, 2015.

[85] Ling Ren, Christopher W Fletcher, Albert Kwon, Emil Stefanov, Elaine Shi, Marten van Dijk, and Srinivas Devadas. Ring oram: Closing the gap between small and large client storage oblivious ram. *IACR Cryptology ePrint Archive*, 2014:997, 2014.

[86] Ling Ren, Christopher W Fletcher, Albert Kwon, Emil Stefanov, Elaine Shi, Marten Van Dijk, and Srinivas Devadas. Constants count: Practical improvements to oblivious ram. In *USENIX Security Symposium*, pages 415–430, 2015.

[87] Farzad Sabahi. Cloud computing security threats and responses. In *Communication Software and Networks (ICCSN), 2011 IEEE 3rd International Conference on*, pages 245–249. IEEE, 2011.

[88] Sajin Sasy, Sergey Gorbunov, and Christopher W Fletcher. Zerotrace: Oblivious memory primitives from intel sgx. *IACR Cryptol. ePrint Arch.*, 2017:549, 2017.

[89] Maryam Sepehri and Florian Kerschbaum. Low-cost hiding of the query pattern. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, pages 593–603, 2021.

[90] Zhirong Shen, Jiwu Shu, and Wei Xue. Preferred keyword search over encrypted data in cloud computing. In *2013 IEEE/ACM 21st International Symposium on Quality of Service (IWQoS)*, pages 1–6. IEEE, 2013.

[91] Zhirong Shen, Jiwu Shu, and Wei Xue. Keyword search with access control over encrypted data in cloud computing. In *2014 IEEE 22nd International Symposium of Quality of Service (IWQoS)*, pages 87–92. IEEE, 2014.

[92] Elaine Shi, T-H Hubert Chan, Emil Stefanov, and Mingfei Li. Oblivious ram with o ((logn) 3) worst-case cost. In *International Conference on The Theory and Application of Cryptology and Information Security*, pages 197–214. Springer, 2011.

[93] SINA Weibo. Weibo-index.

[94] Xiaoming Song, Guoliang Li, Jianhua Feng, and Lizhu Zhou. Effective fuzzy keyword search over uncertain data. In *International Conference on Database Systems for Advanced Applications*, pages 66–70. Springer, 2009.

[95] Emil Stefanov, Elaine Shi, and Dawn Song. Towards practical oblivious ram. *arXiv preprint arXiv:1106.3652*, 2011.

[96] Emil Stefanov, Marten Van Dijk, Elaine Shi, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path oram: an extremely simple oblivious ram protocol. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 299–310. ACM, 2013.

[97] Wenhai Sun, Ruide Zhang, Wenjing Lou, and Y Thomas Hou. Rearguard: Secure keyword search using trusted hardware. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pages 801–809. IEEE, 2018.

[98] Olivier Terzo, Pietro Ruiu, Enrico Bucci, and Fatos Xhafa. Data as a service (daas) for sharing and processing of large data collections in the cloud. In *2013 Seventh International Conference on Complex, Intelligent, and Software Intensive Systems*, pages 475–480. IEEE, 2013.

[99] Shruti Tople, Yaoqi Jia, and Prateek Saxena. Pro-oram: Practical read-only oblivious {RAM}. In *22nd International Symposium on Research in Attacks, Intrusions and Defenses ({RAID} 2019)*, pages 197–211, 2019.

[100] tpc.org. Tpc benchmark h.

[101] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F Wenisch, Yuval Yarom, and Raoul Strackx. Foreshadow: Extracting the keys to the intel {SGX} kingdom with transient out-of-order execution. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 991–1008, 2018.

[102] Peter Van Liesdonk, Saeed Sedghi, Jeroen Doumen, Pieter Hartel, and Willem Jonker. Computationally efficient searchable symmetric encryption. In *Workshop on Secure Data Management*, pages 87–100. Springer, 2010.

[103] Bing Wang, Shucheng Yu, Wenjing Lou, and Y Thomas Hou. Privacy-preserving multi-keyword fuzzy search over encrypted data in the cloud. In *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*, pages 2112–2120. IEEE, 2014.

[104] Cong Wang, Ning Cao, Kui Ren, and Wenjing Lou. Enabling secure and efficient ranked keyword search over outsourced cloud data. *IEEE Transactions on parallel and distributed systems*, 23(8):1467–1479, 2012.

[105] Jianfeng Wang, Hua Ma, Qiang Tang, Jin Li, Hui Zhu, Siqi Ma, and Xiaofeng Chen. Efficient verifiable fuzzy keyword search over encrypted data in cloud computing. *Computer science and information systems*, 10(2):667–684, 2013.

[106] Jie Wang, Xiao Yu, and Ming Zhao. Privacy-preserving ranked multi-keyword fuzzy search on cloud encrypted data supporting range query. *Arabian Journal for Science and Engineering*, 40(8):2375–2388, 2015.

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

[108] Yongzhi Wang, Lingtong Liu, Cuicui Su, Jiawen Ma, Lei Wang, Yibo Yang, Yulong Shen, Guangxia Li, Tao Zhang, and Xuewen Dong. Cryptsqlite: Protecting data confidentiality of sqlite with intel sgx. In *Networking and Network Applications (NaNA), 2017 International Conference on*, pages 303–308. IEEE, 2017.

[109] FU Wei, GU Chenyang, and GAO Qiang. Multi-user sharing oram scheme based on attribute encryption. *Journal of Computer Applications*, 40(2):497, 2020.

[110] Marius Wernke, Pavel Skvortsov, Frank Dürr, and Kurt Rothermel. A classification of location privacy attacks and approaches. *Personal and Ubiquitous Computing*, 18(1):163–175, January 2014.

[111] Jian Xu, Yuanjing Zhang, Kuiyuan Fu, and Su Peng. Sgx-based secure indexing system. *IEEE Access*, 7:77923–77931, 2019.

[112] Ka Yang, Jinsheng Zhang, Wensheng Zhang, and Daji Qiao. A light-weight solution to preservation of access pattern privacy in un-trusted clouds. In *European Symposium on Research in Computer Security*, pages 528–547. Springer, 2011.

[113] Zhengwei Yang, Ada Wai-Chee Fu, and Ruifeng Liu. Diversified top-k subgraph querying in a large graph. In *Proceedings of the 2016 International Conference on Management of Data*, pages 1167–1182, 2016.

[114] Hyundo Yoon, Soojung Moon, Youngki Kim, Changhee Hahn, Wonjun Lee, and Junbeom Hur. Speks: Forward private sgx-based public key encryption with keyword search. *Applied Sciences*, 10(21):7842, 2020.

[115] Xiangyao Yu, Syed Kamran Haider, Ling Ren, Christopher Fletcher, Albert Kwon, Marten van Dijk, and Srinivas Devadas. Proram: dynamic prefetcher for oblivious ram. In *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, pages 616–628. IEEE, 2015.

[116] Xian Zhang, Guangyu Sun, Peichen Xie, Chao Zhang, Yannan Liu, Lingxiao Wei, Qiang Xu, and Chun Jason Xue. Shadow block: accelerating oram accesses with data duplication. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 961–973. IEEE, 2018.

[117] Zhenjie Zhang, Marios Hadjieleftheriou, Beng Chin Ooi, and Divesh Srivastava. Bed-tree: an all-purpose index structure for string similarity search based on edit distance. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 915–926, 2010.

[118] Kui Zheng, Xue-ming Shu, and Hong-yong Yuan. Hot spot information auto-detection method of network public opinion. *Computer Engineering*, 36(3):4–6, 2010.

[119] Minghui Zheng and Huihua Zhou. An efficient attack on a fuzzy keyword search scheme over encrypted data. In *2013 IEEE 10th International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing*, pages 1647–1651. IEEE, 2013.

[120] Zibin Zheng, Jieming Zhu, and Michael R Lyu. Service-generated big data and big data-as-a-service: an overview. In *2013 IEEE international congress on Big Data*, pages 403–410. IEEE, 2013.

[121] Wenchao Zhou, Yifan Cai, Yanqing Peng, Sheng Wang, Ke Ma, and Feifei Li. Veridb: An sgx-based verifiable database. In *Proceedings of the 2021 International Conference on Management of Data*, pages 2182–2194, 2021.

[122] Tieying Zhu, Shanshan Wang, Xiangtao Li, Zhiguo Zhou, and Riming Zhang. Structural attack to anonymous graph of social networks. *Mathematical Problems in Engineering*, 2013, 2013.