Information Systems 96 (2021) 101681

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/is

ProDB: A memory-secure database using hardware enclave and practical oblivious RAM

Ziyang Han [a], Haibo Hu [a,b,*]

[a] Dept. of Electronic and Information Engineering, The Hong Kong Polytechnic University, Hong Kong
[b] Hong Kong Polytechnic University Shenzhen Research Institute, China

ARTICLE INFO

Article history:
Received 22 July 2020
Received in revised form 29 October 2020
Accepted 8 November 2020
Available online 10 November 2020
Recommended by Alysson Neves Bessani

Keywords:
Access pattern
Oblivious RAM
Secure database
Path ORAM
SaP ORAM
Hardware-based security

ABSTRACT

One key challenge for data owners to host their databases in the cloud is data privacy. In this paper, we 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, we 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, we 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, we prove and show ProDB achieves high security and throughput on commodity cloud hosting servers.

1. Introduction

With the prosperity of cloud computing, more and more data owners and service providers host their database management systems (DBMS) in virtual machines to drive web and mobile applications [1,2]. However, emerging cybersecurity attacks that target on cloud infrastructures become a major threat [3,4] and jeopardize the trust between data owners and cloud service providers. Conventional attack methods include injecting malware into the VM image and hijacking the hypervisor [5,6] to gain unauthorized access to data.

Recently, inference attacks [7–9] that exploit the access pattern in disk I/O and memory page activities of VM have emerged. For example, an attacker can find "hot" records by monitoring the access frequency of memory addresses, or he can learn the sketch of a B-tree index by mapping sequences of page accesses to traversal paths in this tree. The access pattern statistics on which the inference attacks rely, such as memory address access frequency, access mode (read/write/append), and disk I/O throughput over a period of time, are resistant to traditional database encryption [9–12]. Furthermore, such attacks do not require full access to the disk and memory space, they are easy to launch and difficult to detect.

To demonstrate the feasibility and consequences of such attacks, in a preliminary experiment, we deploy VMware vSphere, a leading hypervisor used by many cloud services, on a bare-metal machine. We then install MySQL 5.6 database on one VM and run interleavingly, among other workloads, 5 OLAP queries of various types (SQL statements #1, 6, 15, 19, 20 of TPC-H [13][1]), each by 50 runs. We then use the integrated analytical tool VMware vSphere ESXi vCenter to monitor the number of disk reads, small-range disk seeks, medium-range disk seeks, and the small-to-medium seek ratio on this VM.

Table 1 shows the average values over 50 runs. To further evaluate the accuracy of such inference attacks, we use the 4 table columns as features and the first 40 runs as training data to build a naive Bayes classifier, and then use the rest 10 runs as testing data to reidentify their query IDs. The accuracy of this attack is around 84%, which is much higher than 20% by random guess.

To prevent such inference attacks, two cryptographic tools, namely Private Information Retrieval (PIR [14]) and Oblivious RAM (ORAM [15,16]) have been proposed. The latter has gained increasing popularity as it is based on continual memory shuffle instead of computationally hard problem as in PIR [14]. 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

* Corresponding author.
E-mail address: haibo.hu@polyu.edu.hk (H. Hu).

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

https://doi.org/10.1016/j.is.2020.101681

Table 1
Disk access patterns of TPC-H queries.

Query	Read requests	Small	Medium	S/M ratio
1	2813	53 832	2371	22.89
6	2812	45 579	10 688	4.31
15	5638	91 622	21 016	4.40
19	2918	56 046	2283	24.55
20	15 460	235 021	18 627	12.67

the data owner, but also 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 [17,18] and ARM TrustZone [19] becomes a mandatory component in modern CPUs. It is an isolated environment where an application can run its code and lock its sensitive data so that even the hypervisor and operating system cannot access. 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 paper we design *ProDB*, a practical oblivious database that is immune to inference attacks on disk and memory access. ProDB is a minimal adaptation of a conventional DBMS that partially runs on the Intel SGX Enclave [17] and partially runs on the untrusted CPU and memory, bridged by an ORAM protocol. SGX is a TEE that is available in all Intel CPUs since the Skylake microarchitecture. The main challenges 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. In particular, we propose SaP ORAM, a SQL-aware Path ORAM protocol [20] that is optimized for database query processing. 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 [21,22] on the history query-table graph.

To summarize, our main contributions of this paper are as follows.

- We 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.
- We propose SaP ORAM as the ORAM protocol that significantly enhances end-to-end communication 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 [13].
- We analyze the security of SaP ORAM on access patterns rigorously and conduct extensive sets of experiments to demonstrate the efficiency of ProDB.

The rest of the paper is organized as follows. Some preliminary of two key concepts, namely, ORAM and hardware-based security are introduced in Section 2. The system model and the design of ProDB are presented in Section 3. The SaP ORAM protocol is shown in Section 4. Security analysis of SaP ORAM and ProDB is given in Section 5, followed by the experimental results and discussion in Section 6. Section 7 reviews related works in the literature, and Section 8 concludes this paper.

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

2. Preliminaries

In this section, we introduce some fundamental concepts of ORAM protocol and hardware-based security that are related to our work.

2.1. Hardware-based trust computing

A trusted execution environment (TEE) is an isolated environment not accessible by the operating system where an application can run its code and lock its sensitive data [23]. A TEE in Intel SGX technology is called an 'Enclave', which is executed in processor reserved memory (PRM). 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). A common use case of TEE is remote attestation to prove that the launched program code is not altered and is run by a trusted processor [24]. The attestation leverages Intel Enhanced Privacy ID (EPID) for providing signature and can be verified only by Intel Attestation Service. Despite of the advantages, this technology as well as other TEE schemes show limitations in some respects. Above all, common disk I/O and syscalls cannot apply to Enclaves directly as they are not trusted [23].

2.2. Oblivious RAM

Oblivious RAM [15,20,25–29] 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.

As our proposed SaP ORAM is based on Path ORAM [20], we briefly introduce the latter in the following. Some related notations are summarized in Table 2.

As shown in Fig. 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

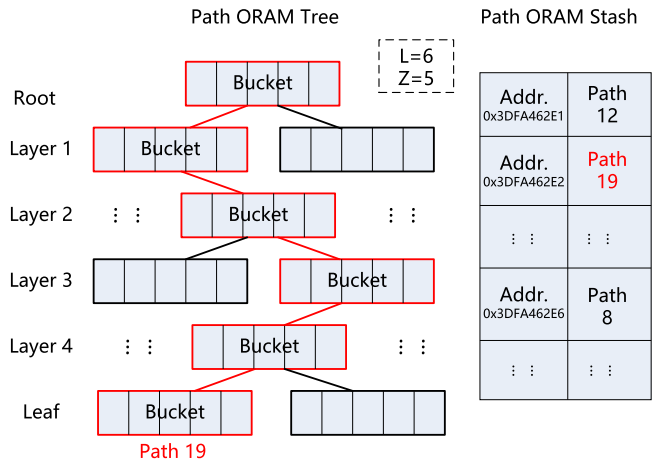


Fig. 1. ORAM path and stash.

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 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 path intersects with the retrieved path at each levels of the tree (see Fig. 1). During this process, if there are more than Z candidate blocks for a node, the surplus ones have to remain in the stash. Otherwise if there are not enough blocks in the stash, the path will be padded with dummy blocks. As the last step, the client writes the path back to server.

3. ProDB architecture

3.1. Overview

3.1.1. Security model

The threats of ProDB come from the hypervisor who has the direct access to the memory space. If the host hypervisor is compromised, data blocks are immediately exposed to adversaries. Even if the database is encrypted, the hypervisor can still analyze the main memory access pattern left by various steps in a SQL query processing, including query execution plan, index and tabular data access. Such memory analysis can infer sensitive information, such as the type and frequency of a query, and the access frequency and order of various data blocks. To defend from both data exposure and memory analysis threats, we define a memory-secure DBMS as follows, which forms the security foundation of ProDB.

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. 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 cannot be eliminated in the whole site of memory space. Some less sensitive information leakage, such as whether the two queries access the same database instance, are laid outside our scope of study. In this work, we set the security boundary of pattern leakage under the level of table data access and the semantic of SQL queries.

3.1.2. System model

As shown in Fig. 2, we assume a database owner (e.g., a business) hosts its database on a virtual machine (VM) in the cloud for web or other applications. To satisfy the second condition in memory-secure DBMS, we adopt an ORAM scheme to conceal the memory access pattern ξ left by DBMS operations. To satisfy the first condition, we further assume a TEE (e.g., Intel SGX [17]) is 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 storage.

3.1.3. Performance metric

As the ORAM cost is the predominant cost (w.r.t. CPU, I/O, and time) in ProDB, we focus on its performance. We use ω to 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.

3.2. A two-tier design

As shown in Fig. 3,[2] ProDB is composed of 2 tiers, namely the ProDB Core and ProDB Shield, which run inside and outside a TEE, respectively.

3.2.1. 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 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 4, is a historical

[2] To clarify, we 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.

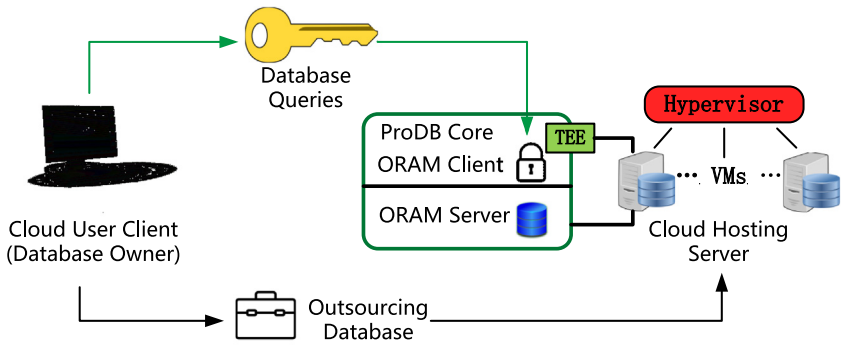


Fig. 2. System model.

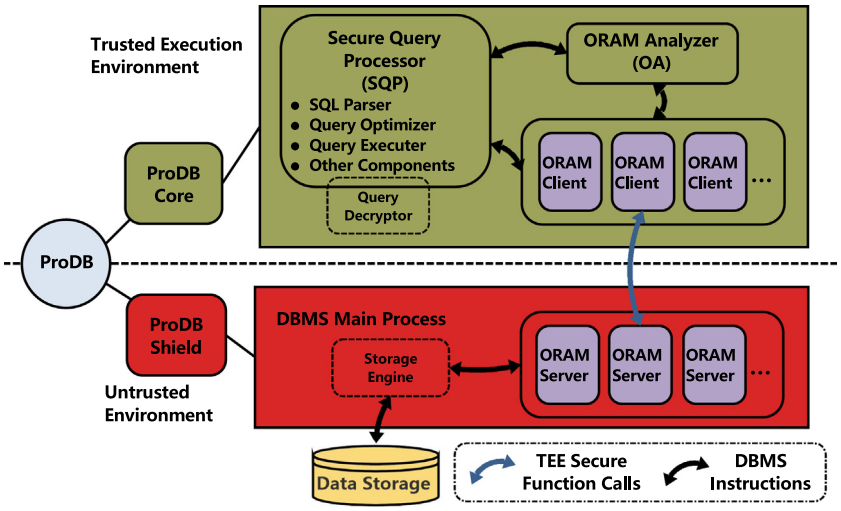


Fig. 3. ProDB two-tier design: Core and shield.

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. The ORAM Analyzer uses these meta-data to find the optimal allocation of tables to ORAM instances (more details in Section 4.2). Alongside the changes on meta-data, ORAM Analyzer updates realtime allocation plan on-demand.

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.

3.2.2. ProDB shield

As shown in Fig. 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 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 4.1.

3.3. Overall ProDB work-flow

ProDB Core and ProDB Shield collaborate with each other to provide memory-secure query operations. Fig. 4 shows the main steps of query processing, among which inflow and outflow instructions of TEE are transmitted through secure channel by issuing secure function calls of TEE (e.g., ECALLs/OCALLs of Intel SGX).

Algorithm 1 shows a detailed work-flow 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 $T = H_{K_h}(E_{K_u}(S)) \parallel E_{K_u}(S)$. 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 4.2) to guide the initialization of ORAM instances (line 8). We will give the construction steps of the optimal table pairing plan in Algorithm 2. Subsequently, each SQL string s is parsed as s^* and optimized to generate execution plan qp (lines 10–11). Meanwhile, the historical query records are updated. If the associated pages of qp do not exist in ORAM instances, SQP notifies storage engine to load them from disk

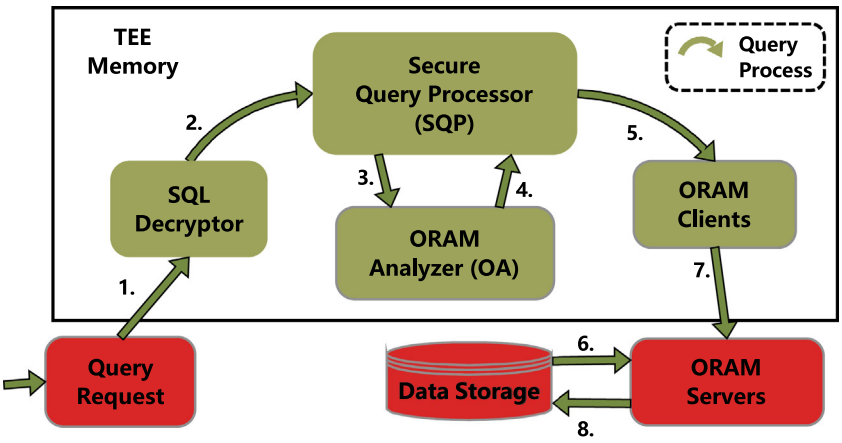


Fig. 4. ProDB query process.

(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 ORAM-to-Disk Mechanism (see Section 4.1) to perform data persistence job and flush related dirty pages to 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**

3.4. 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 characteristics 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.

4. SaP ORAM

In this section, we present the SaP ORAM protocol. It addresses two efficiency problems of Path ORAM when it is applied to ProDB, namely, in persistence and query-based multiple ORAM accesses. In what follows, we discuss them separately.

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 Fig. 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. We add a new attribute 'tag' to the position map of the Path ORAM client. 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 Fig. 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 storage engine about these blocks and infer the query. To address this, we propose a probabilistic dirty-block-generation procedure to produce an expanded and obfuscated update list for persistence. As shown in Fig. 5(c), the procedure randomly adds blocks which 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 maintains independent with distribution of real updates over pages. Obviously, 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

$$prob_b = 2^{-(H+H')}. \quad (1)$$

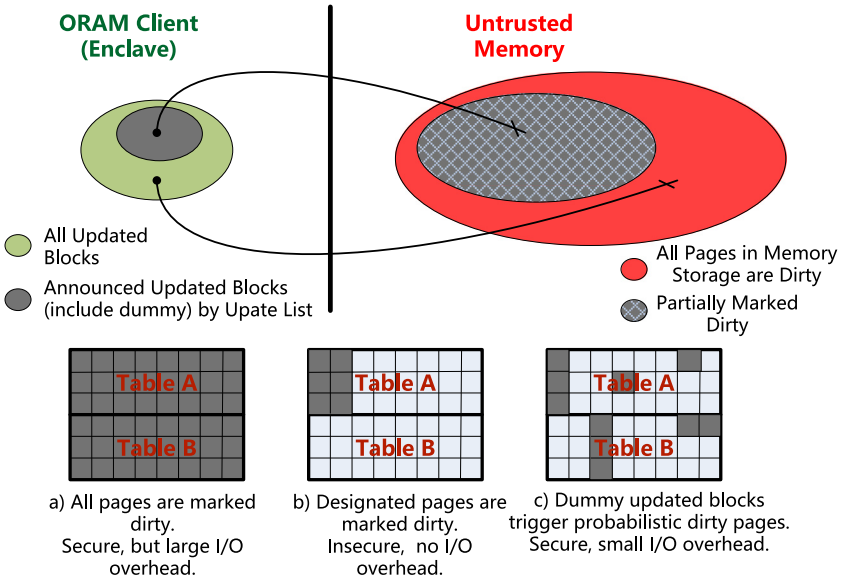


Fig. 5. Probabilistic lazy persistence.

I/O in the storage engine typically use pages rather than blocks. To convert the above to pages, we 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, we assume 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 derive the probability of identifying the real dirty pages as

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

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

Example 1. In a practical setting of $P = 16$ kb $= 4B$, $Z = 4$, $L = 7$, we aim to run data persistence under restrained overhead generating $\leq 1x$, $2x$ number of dirty pages, i.e. $g_D \leq 2g_R$, $g_D \leq 3g_R$. Under such constrains, we let $H = 30$, 40 to derive corresponding $\max H' = 66$, 135 and $\max H' = 102$, 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 $prob_p \leq 2^{-48.4}$, $\leq 2^{-59.9}$ for 1x additional cost and $prob_p \leq 2^{-75.4}$, $\leq 2^{-91.9}$ 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.

4.2. SQL-aware path sharing

Path ORAM and other enhanced tree-based ORAMs [30] 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, we take an orthogonal perspective by **reducing the number of ORAM rounds in the intra- and inter-query scale**.

In this subsection, we propose **SQL-aware path sharing** whose key idea is to fetch more than one block from the tree path during one ORAM round. 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 3.2.1). Based on this history, we merge data blocks from λ tables into a single ORAM server and maintain combined position map and stash of the tables for this ORAM instance. Ideally, if the data blocks in the entire tree path are needed by the query, the amortized ORAM cost can be reduced by up to $(\lambda - 1)\omega$.

For ease of presentation, in what follows we only study $\lambda = 2$, i.e., each ORAM instance holds up to 2 tables A and B, each of which account 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:

$$C = Z(M \log \frac{s}{Z} + N \log \frac{t}{Z})\omega \tag{3}$$

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 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:

$$C' = ZK(\log \frac{s+t}{Z})\omega \tag{4}$$

We then 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 [20]. 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 is always expected that we can find a shared path in ORAM tree holding target blocks of both two tables, so that we can access them in one pass. Now we 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, we show below the relation between $|M^* - N^*|$ and $\Upsilon = |M - N|$. 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, we 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 ith 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 pairing scheme does not take $M + N$ into account. We therefore propose to use a combined metric, namely, the evaluation factor of two tables, to determine if two tables are suitable for pairing.

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 3.2.1. 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 solve this problem, we make an analogy to the maximum weighted matching problem in graph. If we use vertex set V to denote tables and edge set E to denote joint access relations 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 [21] and [22], still require quadratic complexity on $|V|$. In what follows, we propose an efficient heuristic-based approximate algorithm to select a suboptimal table pairing plan (STPP).

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*
1: *sttp* $\leftarrow \emptyset$
2: Initialize vertices set with table set $U : S_V \leftarrow U$
3: Sort α_{ij} in descending order
4: **for each:** α_{ij} from top **do**
5: **if** i and $j \in S_V$ **then**
6: **if** $\alpha_{ij} \neq 0$ **then**
7: Put pair $i \leftrightarrow j$ into *sttp*
8: **else**
9: Put mono-table pair i and j into *sttp*
10: **end if**
11: Remove i, j from S_V
12: **else**
13: **if** i or $j \in S_V$ **then**
14: Put mono-table pair i or j into *sttp*
15: **else**
16: **Continue**.
17: **end if**
18: Remove i or j from S_V
19: **end if**
20: **end for**
21: **Return** *sttp*

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 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).

We further evaluate how the difference on the amount of database tables and the randomness of meta-data impact on computational cost of STPP. We also study the relation between STPP cost and the overall cost in processing a query workload. (Both can be found in Section 6.4.)

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 we make the second condition in security model hold for ProDB. In this section, we show that optimizations in SaP ORAM do not compromise the security of ORAM. We first present a proof on the security of path sharing in SaP ORAM. Then we perform a security analysis on using multiple ORAM instances for a single database.

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. We adopt the same security analysis as in Path ORAM [20] 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 $\xi = (p_m, p_{m-1}, \ldots, p_1)$, 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, we show the that SaP ORAM has 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 of 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 $Pr(S_i)_{SaP} \leq Pr(S_i)_{Path}$.

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 location (e.g., local disks, iSCSI, and NFS), the single-ORAM-instance design cannot capture the difference in their I/O characteristics. Second, the access frequency of various tables are drastically different and single-ORAM-instance design cannot optimize the I/O performance for "hot" tables. As such, we adopt a multi-ORAM-instance design where each ORAM instance accommodates a set of tables. 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, an 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 a single path access to an ORAM server, their original access sequences to these tables can no longer be inferred. To illustrate this, we use the extreme example above. Let $Seq := \ldots, A_1(A, path_i, addr_1, op), A_2(B, path_j, addr_2, op), \ldots$ denote the original retrieval pattern on the two ORAM instances of a net-loop join query on tables A and B. The pattern clearly shows a distinct 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 Fig. 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.

6. Experimental results

We implement a prototype ProDB (i.e., SaP ORAM on MySQL) by refactoring those source codes with block access instructions (such as "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 this section, we evaluate the performance of both ProDB and SaP ORAM through experiments. We first demonstrate the effectiveness of SaP ORAM to hide memory access pattern. We then conduct a comparative study on the efficiency of SaP ORAM and Path ORAM. Finally, we show the overall performance of ProDB under various real-life TPC-H query workloads.

6.1. Effectiveness of SaP ORAM for memory access pattern hiding

We demonstrate the result of memory access pattern hiding against "curious" VM hypervisor and OS administrator for a set of different type TPC-H (1 GB workload) SQL queries (Q4,Q17,Q21, RF1,RF2[3]). Maintaining the same settings as we applied in the preliminary test shown in Section 1, we deploy vSphere ESXi version 6.5 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, 16 GB memory, and MySQL 6.5 database.[4] To eliminate the impact of disk I/O on the query performance, we enforce MySQL to cache all TPC-H query workload in the memory by setting "innodb_buffer_pool_size" to 1 GB. We 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). We then 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[5] on 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 Fig. 7. We observe that this inference attack is very effective (65%+) without SaP ORAM. On the other hand,

[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 [31] 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.

[5] As the attributes of testing results of the same query are very approaching in the stable simulation environment. To avoid overfitting problem, we limit the trial runs in training data. Therefore, based on the features of some classic classifiers [32,33], we carefully select the naive-Bayes classifier for attacking.

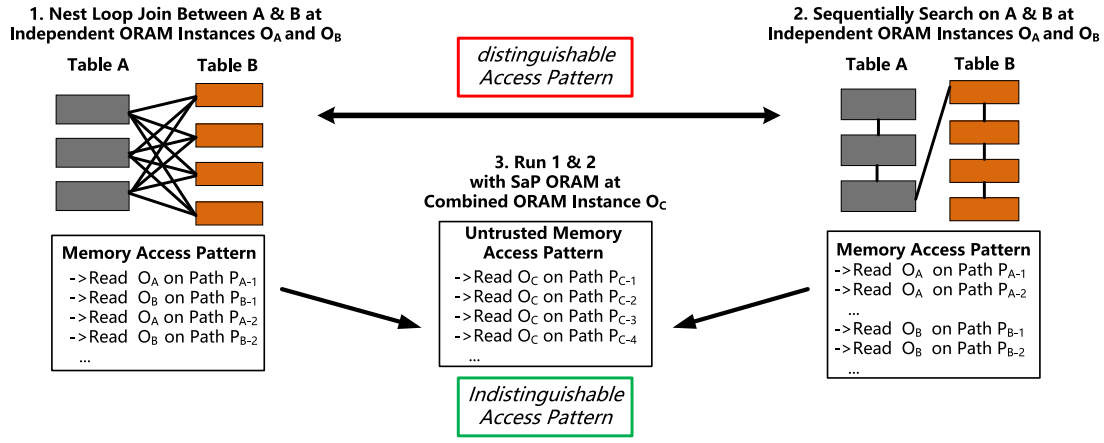


Fig. 6. Access pattern of inter-table queries.

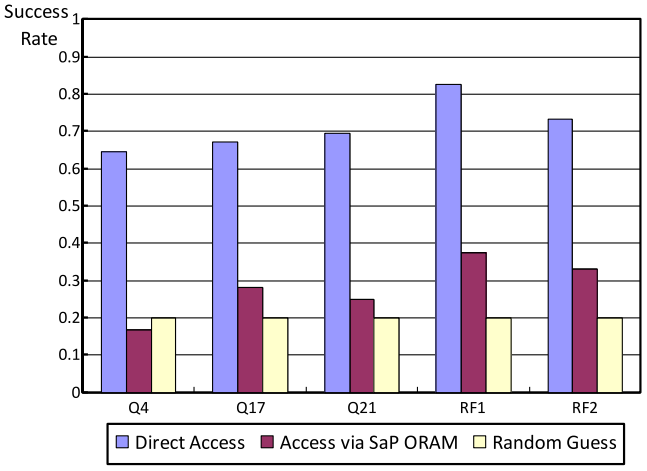


Fig. 7. Success rates of inference attack — Direct access vs. via SaP ORAM.

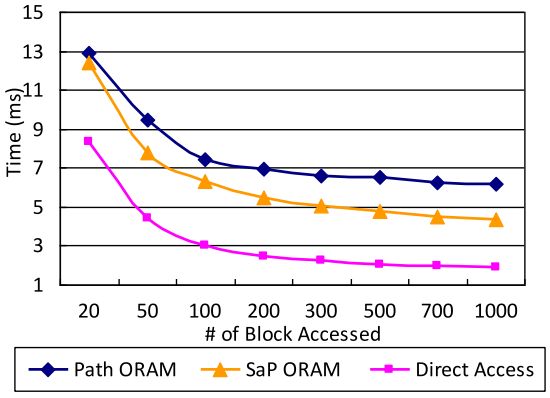


Fig. 8. Amortized elapsed time — SaP ORAM vs. Path ORAM vs. Direct access.

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.

6.2. SaP ORAM performance

In this subsection, we evaluate the SaP ORAM performance against Path ORAM in the presence of Intel SGX. The experiments

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	1295	< 1	1007
*RF*1	68	34.52	159
*RF*2	101	30.66	159

Table 4
Memory access patterns for TPC-H queries – via SaP ORAM.

Query	Mem. reads (K)	Mem. writes (K)	Peak usage (MB)
Q4	2780	2451	893
Q17	10,017	11,634	1012
Q21	15,363	14,268	1022
*RF*1	1057	824	897
*RF*2	2201	2172	997

are conducted on an Intel i7 6700HQ CPU with 16 GB RAM running Windows 10. We sequentially access a 4 kb data array using executable Intel SGX Enclave code segments in three versions of 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 $B = 4$ kb.

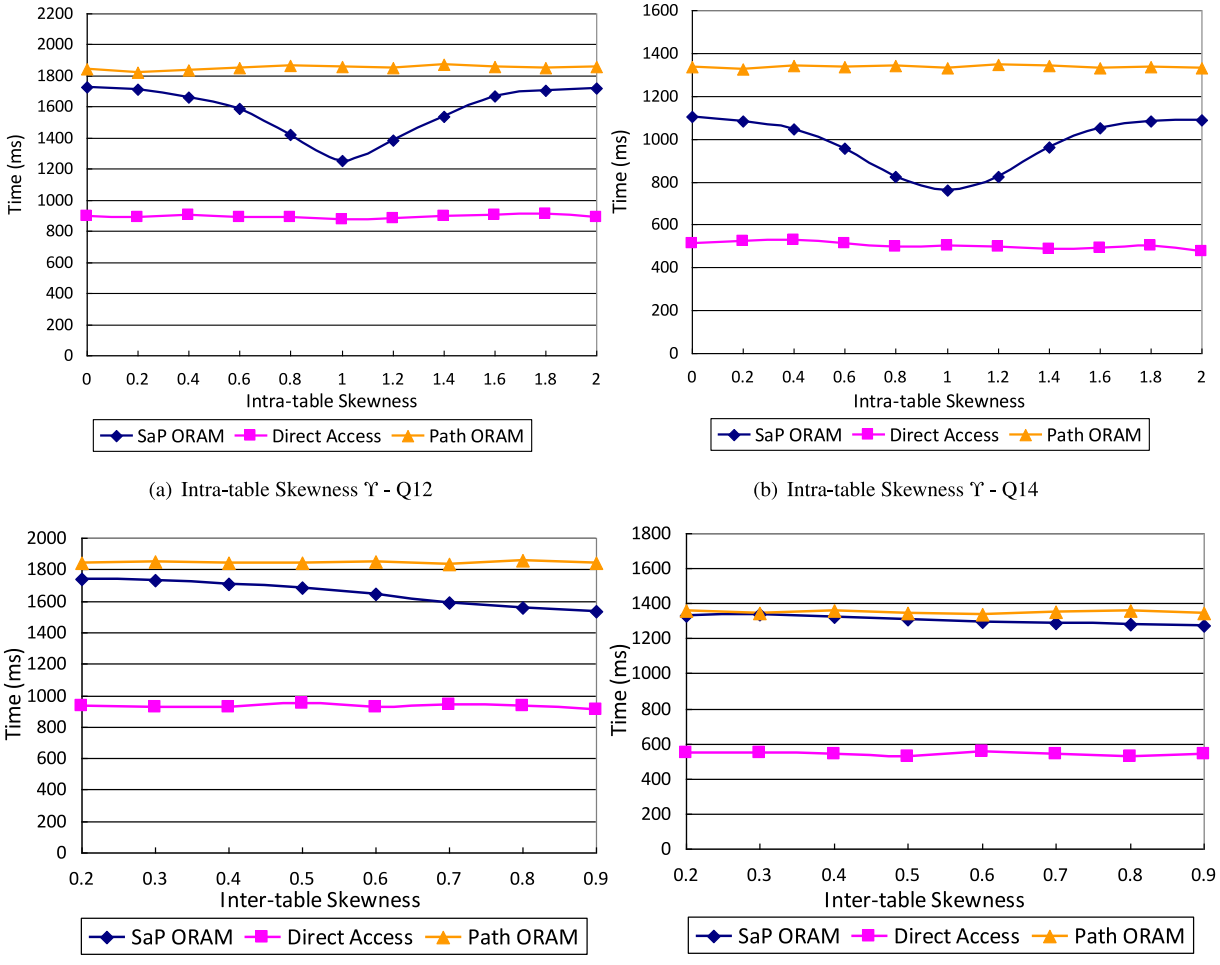
We execute the program using each DLL file for up to 1000 blocks of array data and measure the amortized elapsed time of invoking the enclave to access a block (i.e., from the moment when the program enters the enclave to the moment when the return buffer from the enclave is received). The elapsed time under each DLL file is plotted in Fig. 8. We 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 no ORAM, SaP ORAM only introduces about 100% overhead to the elapsed time when a large number of blocks need to be accessed.

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.

6.3.1. Path sharing

In this part, we evaluate the overall performance of ProDB under TPC-H 1G dataset in terms of amortized query elapsed time. Our experiments focus on two TPC-H queries, namely, Q12



(a) Intra-table Skewness ϒ - Q12

(b) Intra-table Skewness ϒ - Q14

(c) Inter-table Skewness Φ - Q12

(d) Inter-table Skewness Φ - Q14

Fig. 9. Impact of parameters on path sharing performance.

and Q14. Both are two-table join queries.[6] For Q12, we designate tables "orders" and "lineitem" as a pair and merge their blocks in one ORAM tree; for Q14, we pair tables "lineitem" and "part" and merge their blocks in one ORAM tree. To focus on the memory performance rather than disk I/O, we pre-load all data in these tables into memory. All rows in tables are fit in blocks (we set $B = 4$ kb) as full as possible. We arrange the blocks in ORAM tree structure for executing SaP ORAM accesses and in linear structure (Array) for direct non-ORAM accesses.

We study 2 key path sharing parameters that may impact on the performance of ProDB. The first is the intra-table skewness Υ. As defined in Theorem 2, Υ directly impacts on the efficiency gain ψ. In the experiment, we normalize Υ to $[0, 2]$ and $\Upsilon = 1$ means there is no distribution bias between the paired tables. The second parameter is the inter-table skewness Φ which evaluates the degree of 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 $\Phi = \frac{M+N}{X}$.

[6] 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.

In the first experiment, we vary Υ by manipulating the scale of blocks in both tables. Fig. 9(a) and (b) plot the amortized elapsed time of Q12 and Q14, respectively. We 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, we can conclude that ProDB is robust with respect to Υ.

In the second experiment, we vary concentration ratio Φ by adding random block 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. Fig. 9(c) and (d) plot the amortized elapsed time for Q12 and Q14, respectively. We observe that in both figures the elapsed time of SaP ORAM decreases linearly as Φ grows. Note that in Fig. 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".

6.3.2. Probabilistic lazy persistence

In this part, we study how the number of dummy updated blocks added affects the number of generated dirty pages for persistence. We run the experiment on prototype ProDB, as introduced in Section 6. In prior, we configure Innodb parameters as "UNIV_PAGE_SIZE = 16 kb", "UNIV_PAGE_SIZE_SHIFT = 14" and set ORAM block size $B = 4$ kb. In the experiment, we

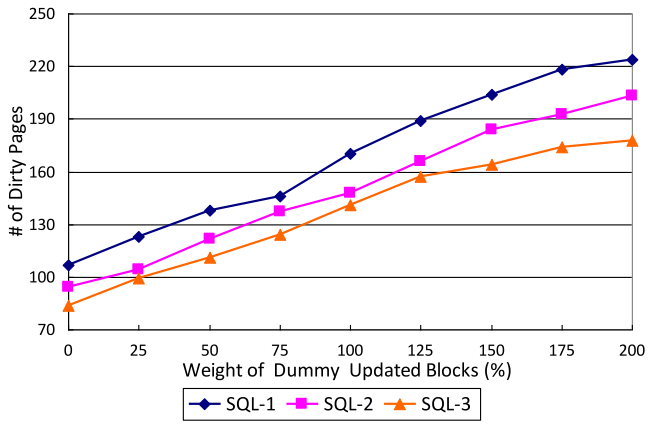


Fig. 10. # of dirty pages changing with # of dummy updates.

perform 3 SQL statements to update different requested rows[7] 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, we observe the number of modified (dirty) pages in Innodb buffer pool. As shown in Fig. 10, the growth of dirty pages of same query fluctuates around linearity along 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.

6.4. Suboptimal table pairing plan

In this subsection, we evaluate the suboptimal table pairing plan (STPP). In the first experiment, we vary the number of tables from 1000 to 10 000 to evaluate its CPU running time while randomizing the meta-data S_i and $J_{i,j}$. Fig. 11(a) plots the results. As each result is averaged by 200 trials, we also plot the standard deviation in the secondary y-axis. We observe that 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, we use real TPC-H query workload to evaluate the time cost of STPP in the whole query processing. Fig. 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). We observe that STTP consumes a fixed amount of CPU running time, and therefore the higher the total query elapsed time, the lower the percentage of STTP cost.

7. Related work

Optimizing ORAM for large-scale data. A rich body of literature has attempted to address the deficiencies of ORAM. TP ORAM [26] and Path ORAM [20] achieve $O(\log N)$ client cost measured by amortized number of blocks accessed per client operation. Goodrich et al. [34] achieves $O(\log^2 N)$ access bandwidth overhead, and Circuit ORAM [29] further reaches $\omega(\log N)$ bandwidth blowup. More recently, many ORAM schemes trade server computation cost for communication cost, such as Path PIR [35], Ring ORAM [36,37] and Onion ORAM [27]. Among them, we remark Bucket ORAM [28] with additive homomorphic encryption for providing single roundtrip (i.e., one single client–server interaction) with $O(1)$ bandwidth blowup. Furthermore, we notice that some efforts have been made to optimize Path ORAM by improving the efficiency of single path retrieval. As a representative work, PrORAM [38] operates directly at the block level by locating shared blocks on same path and retrieving them as "super blocks" in one pass. From the perspective of SQL query processing, Sap ORAM achieves the similar goal without intervening the randomness of block position on ORAM tree, thus furthest preserves the obfuscation of Path ORAM. On the other hand, although based on Path ORAM, the SaP ORAM proposed in this paper can also be adapted to other ORAM schemes for database workloads.

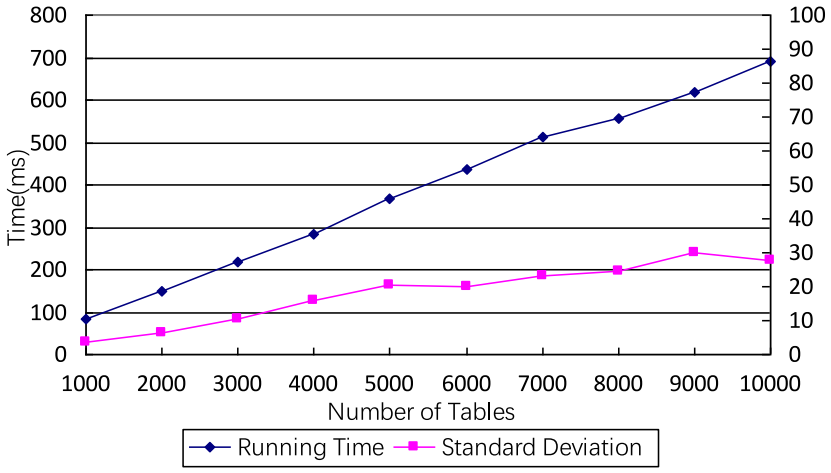
Hardware Enclave for database applications. With the increasing availability of trusted hardware, some recent works have leveraged it for database applications. Bajaj et al. proposed TrustedDB [39], which uses IBM 4758 PCI [40] to implement tamper-proof query processing. CryptSQLite [41] encapsulates the SQLite engine in an Intel SGX enclave to achieve confidentiality with modest performance drop. More recent work, ObliDB [42], further enhances the performance of point query to 7 – 22x faster than existing encryption-based oblivious database. StealthDB [43] and EnclaveDB [44] identify access pattern attacks in untrusted memory/storage and propose cryptographic solutions using secure hardware. They differ from our ProDB in security boundary, optimization of access pattern approaches, as well as high coupling adaptations with hardware enclave, ORAM and disk storage. We also survey into other works [45,46] that adopt hardware enclaves for mitigating related database issues or implementing database systems with specific usages.

Combining use of ORAM and Hardware Enclave. In parallel with this paper, other protocols [47–50] that combine the two cryptographic primitives in the design begin to emerge. ZeroTrace [47] 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 our work. Both Oblix [48] and ObliDB [49] 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 [50] improves the system throughput by using multi-threading Melbourne Shuffle [51] with SGX enclaves while our work fully utilizes the features of SQL queries.

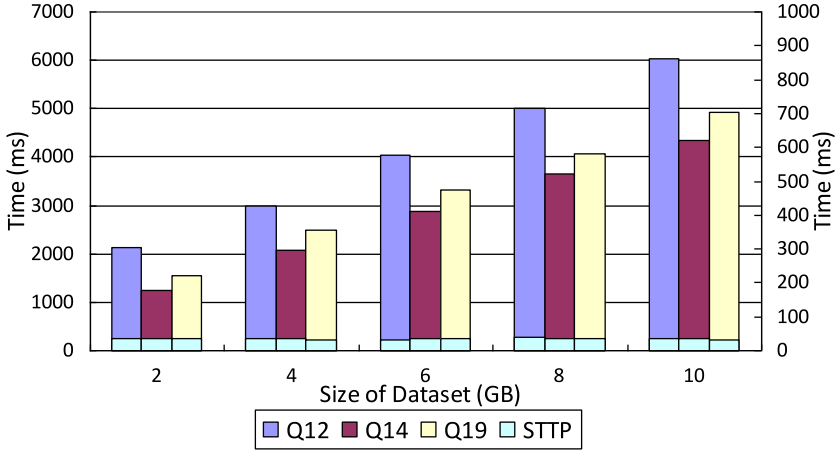
8. Conclusion

In this paper, we 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, we 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 future work, so far ProDB and SaP ORAM achieve high efficiency but cannot perfectly support two key DBMS features, namely the transaction management and concurrency processing. We plan to address them and pave the way for ProDB to become a full-fledged memory-secure DBMS.

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



(a) STPP Running Time Changing with # of Tables



(b) STPP Running Time vs. Query Elapse Time

Fig. 11. STPP running time.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work was supported by National Natural Science Foundation of China (Grant No: U1636205, 61572413), the Research Grants Council, Hong Kong SAR, China (Grant No: 15238116, 15222118, 15218919, and C1008-16G).

References

[1] D.J. Abadi, Data management in the cloud: Limitations and opportunities, IEEE Data Eng. Bull. 32 (1) (2009) 3–12.
[2] R. Prodan, S. 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, IEEE, 2009, pp. 17–25.
[3] B.R. Kandukuri, A. Rakshit, et al., Cloud security issues, in: Services Computing, 2009. SCC'09. IEEE International Conference on, IEEE, 2009, pp. 517–520.
[4] F. Sabahi, Cloud computing security threats and responses, in: Communication Software and Networks (ICCSN), 2011 IEEE 3rd International Conference on, IEEE, 2011, pp. 245–249.
[5] D. Perez-Botero, J. Szefer, R.B. Lee, Characterizing hypervisor vulnerabilities in cloud computing servers, in: Proceedings of the 2013 International Workshop on Security in Cloud Computing, ACM, 2013, pp. 3–10.
[6] N. Arya, M. Gidwani, S.K. Gupta, Hypervisor security-a major concern, Int. J. Inf. Comput. Technol. 3 (6) (2013) 533–538.
[7] A. Arasu, K. Eguro, R. Kaushik, R. Ramamurthy., Querying encrypted data, in: Proceedings of 2014 ACM SIGMOD Conference, 2014, pp. 1259–1261.
[8] M. Wernke, P. Skvortsov, F. Dürr, K. Rothermel, A classification of location privacy attacks and approaches, Pers. Ubiquitous Comput. 18 (1) (2014) 163–175.
[9] M.S. Islam, M. Kuzu, M. Kantarcioglu, Inference attack against encrypted range queries on outsourced databases, in: Proceedings of the 4th ACM Conference on Data and Application Security and Privacy, 2014, pp. 235–246.
[10] R. Ostrovsky, V. Shoup, Private information storage, in: Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing, ACM, 1997, pp. 294–303.
[11] C. Wang, N. Cao, K. Ren, W. Lou, Enabling secure and efficient ranked keyword search over outsourced cloud data, IEEE Trans. Parallel Distrib. Syst. 23 (8) (2012) 1467–1479.
[12] M.S. Islam, M. Kuzu, M. Kantarcioglu, Access pattern disclosure on searchable encryption: ramification, attack and mitigation, in: Ndss, Vol. 20, Citeseer, 2012, p. 12.
[13] tpc.org, Tpc benchmark h, http://www.tpc.org/tpch/.
[14] B. Chor, O. Goldreich, E. Kushilevitz, M. Sudan, Private information retrieval, in: Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on, IEEE, 1995, pp. 41–50.
[15] O. Goldreich, Towards a theory of software protection and simulation by oblivious rams, in: Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing, ACM, 1987, pp. 182–194.
[16] R. Ostrovsky, Efficient computation on oblivious rams, in: Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing, ACM, 1990, pp. 514–523.
[17] Intel, Intel(r) software guard extensions (intel(r) sgx), http://software.intel.com/en-us/sgx.

[18] Intel.com, Intel(r) architecture instruction set extensions programming reference, http://software.intel.com/sites/default/files/managed/48/88/329298-002.pdf.
[19] ARM.com, ARM trustzone, https://www.arm.com/products/security-on-arm/trustzone.
[20] E. Stefanov, M. Van Dijk, E. Shi, C. Fletcher, L. Ren, X. Yu, S. Devadas, Path oram: an extremely simple oblivious ram protocol, in: Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security, ACM, 2013, pp. 299–310.
[21] J. Edmonds, Paths, trees, and flowers, Canad. J. Math. 17 (3) (1965) 449–467.
[22] Z. Galil, Efficient algorithms for finding maximum matching in graphs, ACM Comput. Surv. 18 (1) (1986) 23–38.
[23] V. Costan, S. Devadas, Intel sgx explained, IACR Cryptol. ePrint Arch. 2016 (2016) 86.
[24] J. Aumasson, L. Merino, Sgx secure enclaves in practice–security and crypto review, in: Black Hat, 2016.
[25] E. Shi, T.-H.H. Chan, E. Stefanov, M. Li, Oblivious ram with o ((logn) 3) worst-case cost, in: International Conference on the Theory and Application of Cryptology and Information Security, Springer, 2011, pp. 197–214.
[26] E. Stefanov, E. Shi, D. Song, Towards practical oblivious ram, 2011, arXiv preprint arXiv:1106.3652.
[27] S. Devadas, M. van Dijk, C.W. Fletcher, L. Ren, E. Shi, D. Wichs, Onion oram: A constant bandwidth blowup oblivious ram, in: Theory of Cryptography Conference, Springer, 2016, pp. 145–174.
[28] C.W. Fletcher, M. Naveed, L. Ren, E. Shi, E. Stefanov, Bucket oram: Single online roundtrip, constant bandwidth oblivious ram, IACR Cryptol. ePrint Arch. 2015 (2015) 1065.
[29] X. Wang, H. Chan, E. Shi, Circuit oram: On tightness of the goldreich-ostrovsky lower bound, in: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, ACM, 2015, pp. 850–861.
[30] T. Moataz, T. Mayberry, E.-O. Blass, Constant communication oram with small blocksize, in: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, ACM, 2015, pp. 862–873.
[31] Intel Corporation, Kvm sgx, 2017, https://github.com/intel/kvm-sgx.
[32] M. Stern, J. Beck, B.P. Woolf, Naive Bayes Classifiers for User Modeling, Center for Knowledge Communication, Computer Science Department, University of Massachusetts, 1999.
[33] S.A. Pattekari, A. Parveen, Prediction system for heart disease using Naïve bayes, Int. J. Adv. Comput. Math. Sci. 3 (3) (2012) 290–294.
[34] M.T. Goodrich, M. Mitzenmacher, O. Ohrimenko, R. Tamassia, Privacy-preserving group data access via stateless oblivious ram simulation, in: Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, 2012, pp. 157–167.
[35] T. Mayberry, E.-O. Blass, A.H. Chan, Efficient private file retrieval by combining oram and pir, in: NDSS, Citeseer, 2014.
[36] L. Ren, C.W. Fletcher, A. Kwon, E. Stefanov, E. Shi, M. van Dijk, S. Devadas, Ring oram: Closing the gap between small and large client storage oblivious ram., IACR Cryptol. ePrint Arch. 2014 (2014) 997.
[37] L. Ren, C.W. Fletcher, A. Kwon, E. Stefanov, E. Shi, M. Van Dijk, S. Devadas, Constants count: Practical improvements to oblivious ram., in: USENIX Security Symposium, 2015, pp. 415–430.
[38] X. Yu, S.K. Haider, L. Ren, C. Fletcher, A. Kwon, M. van Dijk, S. Devadas, Proram: dynamic prefetcher for oblivious ram, in: 2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA), IEEE, 2015, pp. 616–628.
[39] S. Bajaj, R. Sion, Trusteddb: A trusted hardware-based database with privacy and data confidentiality, IEEE Trans. Knowl. Data Eng. 26 (3) (2014) 752–765.
[40] IBM, Ibm 4758 pci cryptographic coprocessor general information manual, ftp://www6.software.ibm.com/software/cryptocards/4758gi.pdf.
[41] Y. Wang, L. Liu, C. Su, J. Ma, L. Wang, Y. Yang, Y. Shen, G. Li, T. Zhang, X. Dong, Cryptsqlite: Protecting data confidentiality of sqlite with intel sgx, in: Networking and Network Applications (NaNA), 2017 International Conference on, IEEE, 2017, pp. 303–308.
[42] S. Eskandarian, M. Zaharia, An oblivious general-purpose SQL database for the cloud, 2017, CoRR abs/1710.00458.
[43] A. Gribov, D. Vinayagamurthy, S. Gorbunov, Stealthdb: a scalable encrypted database with full sql query support, 2017, arXiv preprint arXiv:1711.02279.
[44] C. Priebe, K. Vaswani, M. Costa, EnclaveDB: A secure database using sgx, in: EnclaveDB: A Secure Database using SGX, IEEE, p. 0.
[45] A. Ahmad, K. Kim, M.I. Sarfaraz, B. Lee, Obliviate: A data oblivious file system for intel sgx, in: 25th Annual Network and Distributed System Security Symposium, NDSS, 2018.
[46] H. Brekalo, R. Strackx, F. Piessens, Mitigating password database breaches with intel sgx, in: SysTEX@ Middleware, 2016, p. 1.
[47] S. Sasy, S. Gorbunov, C.W. Fletcher, Zerotrace: Oblivious memory primitives from intel sgx., IACR Cryptol. ePrint Arch. 2017 (2017) 549.
[48] P. Mishra, R. Poddar, J. Chen, A. Chiesa, R.A. Popa, Oblix: An efficient oblivious search index, in: 2018 IEEE Symposium on Security and Privacy (SP), IEEE, 2018, pp. 279–296.
[49] S. Eskandarian, M. Zaharia, Oblidb: Oblivious query processing using hardware enclaves, 2017, arXiv preprint arXiv:1710.00458.
[50] S. Tople, Y. Jia, P. Saxena, Pro-oram: Practical read-only oblivious {RAM}, in: 22nd International Symposium on Research in Attacks, Intrusions and Defenses ({RAID} 2019), 2019, pp. 197–211.
[51] O. Ohrimenko, M.T. Goodrich, R. Tamassia, E. Upfal, The melbourne shuffle: Improving oblivious storage in the cloud, in: International Colloquium on Automata, Languages, and Programming, Springer, 2014, pp. 556–567.

Ziyang Han is a Ph.D. student in the Department of Electronic and Information Engineering, Hong Kong Polytechnic University. His current research work include privacy-aware computing, information hiding, hardware-based security and security issues on databases. He has engaged in security component development of many widely used applications.

Dr. Haibo Hu is an associate professor and the deputy program leader (INS) in the Department of Electronic and Information Engineering, Hong Kong Polytechnic University. His research interests include cybersecurity, data privacy, internet of things, and machine learning. He has published over 80 research papers in refereed journals, international conferences, and book chapters. As principal investigator, he has received over 10 million HK dollars of external research grants from Hong Kong and mainland China. He has served in the organizing committee of many international conferences, such as ACM GIS 2020, MDM 2019, DASFAA 2011, DaMEN 2011, 2013 and CloudDB 2011, and in the program committee of dozens of international conferences and symposiums. He is the recipient of a number of titles and awards, including IEEE MDM 2019 Best Paper Award, WAIM Distinguished Young Lecturer, VLDB Distinguished Reviewer, ACM-HK Best Ph.D. Paper, Microsoft Imagine Cup, and GS1 Internet of Things Award.