Beyond Access Pattern: Efficient Volume-Hiding Multi-Range Queries Over Outsourced Data Services

Haoyang Wang, *Graduate Student Member, IEEE*, Kai Fan, *Member, IEEE*, Chong Yu, *Member, IEEE*, Kuan Zhang, *Member, IEEE*, Fenghua Li, *Senior Member, IEEE*, and Haojin Zhu, *Fellow, IEEE*

***Abstract*—Multi-range query (MRQ) is a typical multi-attribute data query widely used in various practical applications. It is capable of searching all data objects contained in a query request. Many privacy-preserving MRQ schemes have been proposed to realize MRQ on encrypted data. However, existing MRQ schemes only consider the security threat caused by access pattern leakage, not the harm of volume pattern leakage. Moreover, most existing schemes cannot achieve efficient queries and updates while preserving the access pattern. In this paper, we propose an efficient MRQ scheme for hiding volume and access patterns. We first design a joint data index using Order-Revealing Encryption (ORE) and Pseudo-random functions (PRFs) to realize volume-hiding range queries. Then, we combine the private set intersection (PSI) and hardware Software Guard Extensions (SGX) to compute each attribute's intersection of query results. In addition, we preserve access patterns during queries by designing a batch refresh algorithm and an update protocol. Finally, rigorous security analysis and extensive experiments demonstrate the security and performance of our scheme in real-world scenarios.**

Received 29 May 2024; revised 14 October 2024; accepted 5 February 2025. Date of current version 4 March 2025. This work was supported in part by the National Natural Science Foundation of China under Grant 62372356 and Grant 92067103, in part by the Key Laboratory of Computing Power Network and Information Security, Ministry of Education under Grant 2023ZD018, in part by the Key Research and Development Program of Shaanxi under Grant 2021ZDLGY06-02, in part by the Natural Science Foundation of Shaanxi Province under Grant 2019ZDLGY1202, in part by Shaanxi Innovation Team Project under Grant 2018TD-007, in part by Xi'an Science and Technology Innovation Plan under Grant 20189168CX9JC10, and in part by the National 111 Program of China under Grant B16037. The associate editor coordinating the review of this article and approving it for publication was Dr. Kaiping Xue. *(Corresponding author: Kai Fan.)*

Haoyang Wang is with the State Key Laboratory of Integrated Services Networks, School of Cyber Engineering, Xidian University, Xi'an 710071, China (e-mail: why19970701@163.com).

Kai Fan is with the State Key Laboratory of Integrated Services Networks, School of Cyber Engineering, Xidian University, Xi'an 710071, China, and also with the Guangzhou Institute of Technology, Xidian University, Guangzhou, Guangdong 510700, China (e-mail: kfan@mail.xidian.edu.cn).

Chong Yu is with the Department of Computer Sciences, University of Cincinnati, Cincinnati, OH 45221 USA (e-mail: yuc5@ucmail.uc.edu).

Kuan Zhang is with the Department of Electrical and Computer Engineering, University of Nebraska–Lincoln, Lincoln, NE 68588 USA (e-mail: kuan.zhang@unl.edu).

Fenghua Li is with the Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China, and also with the State Key Laboratory of Integrated Services Networks, School of Cyber Engineering, Xidian University, Xi'an 710071, China (e-mail: lifenghua@iie.ac.cn).

Haojin Zhu is with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240, China (e-mail: zhu-hj@cs.sjtu.edu.cn).

Digital Object Identifier 10.1109/TIFS.2025.3540576

***Index Terms*—Privacy-preserving multi-range query, volume pattern, access pattern, hardware SGX.**

I. Introduction

NOWADAYS, encrypted MRQs services greatly facilitate people's lives. Millions of users are involved in various applications such as social networks, smart transportation, and e-healthcare, which rely on encrypted MRQs. Existing encrypted MRQ schemes allow partial information leakage, such as access and search patterns, to balance efficiency and security during the setup and query phases. However, the adversary can utilize the leaked patterns to launch Leakage-abused Attacks (LAAs), exploiting vulnerabilities [1], [2], [3], [4], [5], [6], [7]. For instance, LAAs against encrypted range queries usually utilize access pattern to rebuild the datasets [1], [2], [3], [4], while a few advanced LAAs also use search pattern as supplementary information [5], [6], [7]. Therefore, existing studies [8], [9], [10] prioritize preserving the access pattern in encrypted MRQs to mitigate the risk of information leakage. Furthermore, the correlation between the query token (used to perform the search) and the response volume of retrieved data is revealed by different queries (i.e., volume pattern), such that the adversary can infer the value relations of index entries using the distribution of the response volume. In addition, studies [1], [3], [11] identified LAAs exploiting volume-pattern leakage, which can lead to the unauthorized access of sensitive information within encrypted datasets and query responses.

To cope with the new LAAs, recent works [12], [13], [14], [15], [16], [17] proposed volume-hiding index designs to enhance security and privacy in keyword-based queries. However, these schemes face limitations in achieving volume-hiding over encrypted range queries due to two key features: (I) Unlike keyword-based queries, the result size of range queries cannot be pre-determined. Without prior knowledge of the result size, clients must download all datasets to hide volume pattern leakage, leading to significant bandwidth overhead; (II) The continuous nature of range query results (i.e., access pattern) creates multiple unique intersections that adversaries can exploit to infer plaintext distribution in the dataset. For instance, the result set for endpoint values appears in all other range query results. By analyzing these intersections, an

attacker can infer data distribution even if the volume pattern is protected. To address these challenges, HybrIDX [18] combines cryptographic primitives with trusted hardware to enable encrypted range queries with volume-hiding. It utilizes the enclave to split a range query into independent sub-queries, protecting confidentiality during searches from an untrusted server. Similarly, [19] divides the data range into sub-ranges and designs an inverted index based on order-weighted and bitmap indexes to achieve volume-hiding encrypted range queries.

However, despite the effectiveness of both schemes in single-attribute volume-hiding range queries, challenges persist when extending them to encrypted range queries over multi-attribute datasets: (I) HybrIDX utilizes enclave to store indexes and perform searches, but the limited memory of the enclave (128MB) restricts HybrIDX's application on large-scale multi-attribute datasets; (II) In [19], clients are burdened with storing the binary index BT and performing partial searches locally, imposing huge overheads on clients. In addition, the inverted index in [19] cannot be directly extended to multi-attribute for large-scale range queries. To avoid enclave memory limitations, we store the joint index on the storage server, while the enclave is responsible only for generating query tokens for the multi-maps.

In this paper, we propose VHIDX, a volume-hiding range queries scheme for encrypted multi-attribute datasets. VHIDX effectively addresses key challenges while ensuring efficient privacy-preserving MRQs. To balance security and practicality, it designs a joint index framework, consisting of range-based indexes and multi-maps to hide the result volumes for each attribute. VHIDX then proposes a secure and batch search protocol to retrieve matched results. It first retrieves attribute values from the range-based index, and then uses the enclave to decompose the query into independent sub-queries for each value. The corresponding results are fetched from the multi-maps, which consistently have smaller volumes. Finally, VHIDX applies a private set intersection protocol to compute the MRQ results and return them to the clients.

To construct fixed-volume result sets for sub-queries with different attribute values, we draw inspiration from the multi-map structure used in volume-hiding encrypted queries [15]. For each attribute, the result set of matching values is divided into ciphertext blocks of equal volume. However, since the multi-map structure does not directly support range queries, we design an encrypted range-based index using a binary tree. This index enables the mapping of index entries to the multi-maps. Finally, we integrate the joint indexes with our secure batch query protocol, achieving volume-hiding range queries over encrypted multi-attribute datasets.

In addition to volume pattern protection, we mitigate access pattern leakage during queries. To this end, we cache the query results into the enclave at each query and periodically refresh the cached data using our proposed secure batch refresh algorithm. This approach reduces access pattern leakage and lowers communication overhead between clients and the server. We summarize the main contributions below:

1) We design VHIDX as the first scheme that can perform volume-hiding range queries on encrypted multi-attribute datasets. For each attribute, the joint indexes in VHIDX utilize the encrypted tree structure to index the attribute values, and store the data records in the multi-maps. Based on the aid of the enclave, the joint indexes achieve secure and efficient retrieval.
2) We propose a secure batch query protocol and a batch refresh algorithm. Combined with the joint indexes, VHIDX implements volume-hiding for query results and access pattern protection.
3) Based on the analysis and characterization of leakage information during setup and queries, we formally define the leakage function of VHIDX and rigorously prove its security.
4) We implement a prototype of VHIDX and evaluate its performance under a real-world dataset. The experimental results demonstrate that VHIDX has a significant efficiency advantage over the state-of-the-art in existing encrypted MRQ schemes.

The remainder of this paper is organized as follows: Section II studies existing work related to our scheme. Section III presents the preliminaries involved in our design. Section IV formally defines the security model and describes the threat model of our scheme as well as the design goals. Section V presents the concrete construction of our scheme. Sections VI and VII conduct security analysis and performance evaluation, respectively. Finally, we summarize the work of this paper in Section VIII.

II. Related Work

A. Volume-Hiding Scheme

Recent attacks [1], [3], [11] against encrypted queries highlight the importance of volume-hiding, a novel concept in LAAs following access and search patterns. Volume-hiding ensures that the size of query results is hidden. Although a naive way to achieve volume-hiding involves padding query results to ensure identical volumes [12], [13], [14], this incurs extra computational and communication overhead. In their work [15], [16], Kamara et al. introduced two volume-hiding encrypted keyword query schemes using multi-maps. They mapped each keyword query result into multiple fixed-length ciphertext blocks, so that the (fixed) maximum volume of the input dataset is consistent with the query results in the ciphertext blocks. Petal et al. [17] utilized Cuckoo filters and differential privacy to reduce the cost of padding while achieving volume-hiding.

However, these keyword-based query schemes do not apply to range-based queries, as discussed in Section I. Addressing this limitation, Ren et al. [18] introduced a novel hybrid data index with a hardware enclave, enabling volume-hiding in range-based queries. By deploying a binary tree in the enclave and outsourcing encrypted multi-maps to the server, they convert range queries into multiple sub-queries using the tree index and generate encrypted labels. After obtaining the labels, the server retrieves in encrypted multi-maps and returns fixed-length ciphertext blocks for clients. Although the trusted computing environment of the enclave facilitates the scheme [18], its memory space poses a primary limitation. Similarly,

the work [19] divides the entire data range into multiple independent partitions and stores partition information on the client using a local search tree. Additionally, they combined the order-weighted inverted index and bitmap structure to implement range queries with volume-hiding. However, the client incurs extra overhead by performing exact retrieves locally after obtaining matched partitions.

B. Encrypted Multi-Attribute Range Queries

In terms of cryptographic primitives, while order-preserving encryption (OPE) facilitates encrypted range queries [20], [21] due to its protection of plaintext order, it is vulnerable to chosen plaintext attacks [22], compromising plaintext order privacy. Subsequently, efforts integrating public key cryptography, such as Hidden Vector Encryption (HVE) [23], [24], [25], aimed to enhance data privacy in MRQs. Boneh and Waters [23] achieved data privacy protection in MRQs with Hidden Vector Encryption (HVE). Moreover, Wang et al. [24] improved the query efficiency by combining R-tree and HVE. Shi et al. [25] decomposed multi-attribute ranges into single-attribute ones, and protected data privacy with bilinear pairings. However, the above schemes are not applicable in real scenarios due to the high overhead of public key cryptography.

On the other hand, certain data structure methods, such as bucket division [26], [27], focus on maintaining the order of data within buckets to prevent order information leakage. However, these methods often burden resource-constrained clients and may generate false positive records in query results, thus requiring additional computational overhead for filtering.

Additionally, tree indexes have emerged as a mainstream data structure in encrypted MRQs [10], [28], [29], [30], [31], [32], [33], [34]. Lu et al. [28] combined B+ tree with symmetric key encryption to handle single-attribute range queries and extended it to MRQs. Other schemes [29], [30] utilized Asymmetric Scalar Product Encryption (ASPE) for data privacy protection and R-trees for multi-dimensional retrieval. However, these schemes cannot implement semantic security in the random oracle model. To improve search efficiency, Zheng et al. proposed PRQ and PMRQ based on R-tree. Both schemes utilized encoding techniques to encode attributes and ASPE to preserve data and query privacy. Yang et al. [10] enhanced the scheme security based on [30] by secure multi-party computation protocol and protected the access pattern. Similarly, Tu et al. [33] designed intersection predicate encryption (IPE) and subset predicate encryption (SPE) operations using homomorphic encryption, Hadamard product, and ASPE, based on R-tree to achieve efficient retrieval while preserving access pattern. However, [10], [33] are both designed on the two-server model, thus the extra communication overhead limits its practical application. Wang et al. [34] avoided the huge cost incurred by two servers using hardware SGX-assisted servers. They encoded the multi-attribute data by Hierarchical Hyper-rectangle Encoding (HHRE) and Indistinguishable Bloom Filter (IBF), then stored them in a binary tree. Moreover, SGX is used to share the server query cost, thus enabling quick response to MRQs.

TABLE I
FUNCTIONALITY COMPARISON WITH PRIOR ARTS

Scheme	Pattern hiding	Dynamic update	Encryption method	Cloud architecture
Maple [24]	✗	✗	HVE	Single
PRQ [31]	✗	✗	ASPE	Single
PMRQ [32]	✗	✗	ASPE	Single
TRQED+ [10]	AP	✓	Enhanced ASPE	Two
PMRK [33]	AP	✗	ASPE	Two
SGX-Skyline [34]	✗	✓	SKE	Single with SGX
VHIDX	AP,VP [1]	✓	ORE,SKE	Single with SGX

[1] AP and VP denote the access pattern and volume pattern, respectively.

TABLE II
NOTATION DESCRIPTIONS

Main Notations	Descriptions
v_i^j	The attribute value belongs to the attribute V^j
$\mathsf{DB}(v_i^j)$	All matched IDs corresponding to the value v_i^j
I_{rng}^j	The range-based index of the j-th attribute V^j
$N[v_i^j]$	The node of the value v_i^j in I_{rng}^j
I_{mm}^j	The multi-map of the j-th attribute V^j
Q	A query request for a w attributes dataset DB
T	The corresponding query token of Q
$[\cdot]_o^L, [\cdot]_o^R$	The left and right encryption of the ORE algorithm
$[\cdot]_A$	The symmetric encryption of the AES algorithm
G, F	Two pseudo-random functions (PRFs)
$\mathsf{C_{SGX}}$	The cache of the enclave
k_0, k_1, k_2	Secret keys for AES, $G(\cdot)$ and ORE, respectively
(sk, pk)	Key-pair for Pailliar encryption

Unfortunately, all tree-based encrypted MRQ schemes suffer from the "curse of dimensionality". i.e., the query efficiency decreases significantly as the data dimensionality increases.

III. PRELIMINARIES

In this section, we introduce the cryptographic primitives utilized in our scheme VHIDX, which contains Paillier cryptosystem, Private Set Intersection (PSI), and Order-Revealing Encryption (ORE). TABLE II describes the main notations in the following chapters.

A. Paillier Cryptosystem

The Paillier cryptosystem [35], as a probabilistic asymmetric encryption algorithm with additive homomorphism, i.e., given only the public key and the ciphertexts c_1 and c_2 of the plaintexts m_1 and m_2, it is able to compute the ciphertext of $(m_1 + m_2)$. The construction details of Paillier algorithm are depicted as follows:

- **Homomorphic addition of plaintexts**. The product of the given two ciphertexts will decrypt to the sum of their plaintexts, where r_1 and r_2 are two random numbers.

$$D(E(m_1, r_1) \cdot E(m_2, r_2) \bmod n^2) = (m_1 + m_2) \bmod n \quad (1)$$

- **Homomorphic multiplication of plaintexts**. A ciphertext of m_1 raised to a constant c will decrypt to the product of the plaintext and the constant.

$$D(E(m_1, r_1)^k \bmod n^2) = km_1 \bmod n \tag{2}$$

The Paillier cryptosystem implements semantic security against chosen plaintext attacks (IND-CPA).

B. Private Set Intersection

Private Set Intersection (PSI) enables two or more parties to compute the intersection of their respective datasets privately. The output of the PSI does not reveal any information other than the intersection itself. Based on the Paillier cryptosystem, Freedman et al. [36] designed a secure and efficient PSI protocol called FNP protocol.

1) Given a set $\mathcal{A}$, Alice represents $\mathcal{A}$ as a polynomial $f(x) = \prod_{a_i \in A}(x - a_i)$. Apparently, the set of all roots of $f(x) = 0$ is $\mathcal{A}$.
2) The coefficients of the polynomial $f(x) = 0$ are encrypted with Paillier algorithm and the encrypted polynomial $E(f(x))$ is sent to Bob.
3) Based on his own dataset $\mathcal{B}$, Bob can compute the intersection results $\{r_j \in R | r_j = E(f(x)) +_h b_j\}$, where $+_h$ denotes the Paillier homomorphic addition. Then $\mathcal{R}$ is returned to Alice.
4) Alice decrypts $\mathcal{R}$ as $\mathcal{R}'$, and acquires $\mathcal{A} \cap \mathcal{B}$ by computing $\mathcal{A} \cap \mathcal{R}'$.

C. Order-Revealing Encryption

Order-revealing encryption (ORE) is an essential cryptographic primitive in searchable encryption, which allows for private range queries over encrypted data. Based on "left/right" framework, Lewi et al. [37] proposed an efficient and secure ORE scheme compared to prior work [38], [39]. The ORE scheme [37] is a tuple of four algorithms $\prod = (\mathsf{ORE.Setup}, \mathsf{ORE.Enc_L}, \mathsf{ORE.Enc_R}, \mathsf{ORE.Cmp})$ defined over a encrypted message space $[N]$, which is described as follows:

- $\mathsf{ORE.Setup}(1^\lambda)$. Input a security parameter λ, this algorithm outputs the secret key $\mathsf{sk} = (k, \pi)$, where k is a PRF key $k \xleftarrow{\$} \{0,1\}^\lambda$, $\pi : [N] \to [N]$ denotes a uniformly random permutation.
- $\mathsf{ORE.Enc_L}(\mathsf{sk}, x)$. Input a plaintext x and the secret key sk, this algorithm outputs the tuple $\mathsf{ct_L} = (F(k, \pi(x)), \pi(x))$.
- $\mathsf{ORE.Enc_R}(\mathsf{sk}, y)$. Input a plaintext x and the secret key sk. For each $i \in [N]$, this algorithm generates a random nonce $\gamma \xleftarrow{\$} \{0,1\}^\lambda$ and computes the value v_i

$$v_i = \mathsf{CMP}(\pi^{-1}(i), y) + H(F(k, i), r) \bmod 3. \tag{3}$$

  It outputs the ciphertext $\mathsf{ct_R} = (r, v_1, \ldots, v_N)$.
- $\mathsf{ORE.Cmp}(\mathsf{ct_L}, \mathsf{ct_R})$. Input two ciphertexts $(\mathsf{ct_L}, \mathsf{ct_R})$ and outputs the result $(v_h - H(F(k, \pi(x)), r)) \bmod 3$.

Note that the set of right ciphertexts obtained individually is semantically secure, i.e., obtaining only the set of right ciphertexts does not reveal any information about the underlying plaintexts (including their order information).

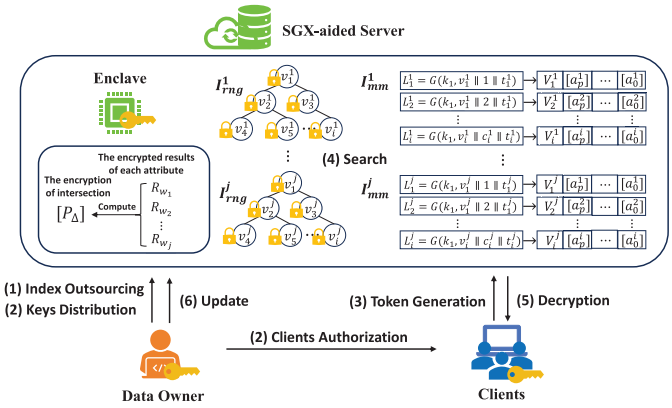


Fig. 1. System model of VHIDX.

D. Software Guard Extensions

Intel SGX is a set of security-related instruction codes built into modern Intel CPUs that enables the creation of isolated, secure memory regions called enclave. These enclaves allow applications to execute sensitive code and handle private data in a trusted execution environment, even if the rest of the system is compromised. The main feature of SGX is its ability to protect the integrity and confidentiality of code and data, shielding them from attacks originating from higher-privileged software such as the operating system or hypervisor. Enclaves are designed to minimize the trusted computing base by isolating only the most critical parts of an application, enabling secure computation on untrusted platforms. SGX is widely used in scenarios such as cloud computing, digital rights management, and secure data analytics, where protecting sensitive information from unauthorized access is crucial.

IV. Problem Formulation

In this section, we introduce the system model, threat model, problem definition, security model, and design goals.

A. System Model & Threat Model

In this paper, we investigate scenarios involving multi-attribute data outsourcing in cloud computing applications such as smart transportation, e-healthcare, and social media. These scenarios involve data owners outsourcing data and clients initiating query requests. We specifically focus on index queries, which involve the private retrieval of multi-attribute data identifiers matching specific query requirements. The system model of VHIDX comprises three main entities: an SGX-aided server, data owners, and clients, as illustrated in Fig. 1.

- **SGX-aided server**. The SGX-aided server consists of the untrusted storage server and the enclave. (I) *Untrusted storage server*. Based on powerful computational and storage capabilities, the untrusted storage server provides search and storage services to clients and the DO, respectively. (II) *Enclave*. As a fully trusted component with limited computational and storage capabilities, the enclave can aid the untrusted storage server in responding to query requests.

- **Data owner**. The DO builds the joint indexes for its multi-attribute datasets, and uploads them to the SGX-aided server.
- **Clients**. The authorized clients initiates query requests by generating query tokens, and decrypts encrypted intersection results after obtaining query results.

The DO, which may be an organization or institution requiring data outsourcing, each data records in the dataset DB are described by multiple attributes and with a unique identifier. Considering data privacy and searchability, the DO builds the joint indexes for the DB and uploads them to the untrusted storage server (Step(1)). The DO also authorizes search access to clients, and shares secret keys with clients and the enclave, respectively. (Step(2)). When authorized clients aim to perform MRQ, they generate a query token and upload it to the server. (Step(3)). Then, the untrusted storage server performs the queries over the joint indexes and returns the encrypted intersection results with the aid of the enclave. (Step(4)). After receiving the query results, clients utilize the secret key to decrypt the encrypted results and obtain the matching data IDs. (Step(5)). Finally, the DO can update the outsourced datasets by uploading the update tokens to the server. (Step(6)). The architecture of the SGX-aided server not only reduces clients' overhead, but also facilitates the volume-hiding and access pattern protection.

Threat Model. A powerful adversary is considered in VHIDX, which can control the entire software stack on the server side, in addition to the code in the enclave. This adversary adheres to predefined protocols but seeks to infer useful information from the background knowledge of the available data and dataset distribution. Particularly, the adversary can monitor the search protocols, and access query tokens, matched index entries, and query responses. Due to SGX protection, the contents of the preserved memory pages and CPU registers in the enclave cannot be accessed directly by the adversary. The DO and clients are considered fully trusted, and they strictly adhere to the protocols for building indexes and generating query tokens. The DO and clients securely store the secret keys. Denial-of-service attacks and side channel attacks [40], [41], [42] are out of the focus of this paper, although orthogonal studies [43], [44], [45] have proposed solutions.

B. Problem Definition

Let $\mathsf{DB} = \{d_1, \ldots, d_n\}$ be a multi-attribute dataset collected by the DO, where each record $id_i (1 \le i \le n)$ is described by w attribute values $\{v^1, \ldots, v^w\}$, that is, $id_i = (id_i(v^1), \ldots, id_i(v^w))$. Client can initiate a MRQ $Q = \{q^j = [x^j, y^j]\}(1 \le j \le w)$ with a privacy-preserving way to search the identifiers that each attribute value v_i^j fall within the corresponding query range q^j, that is, $\mathsf{DB}(Q) = \{id_i | id_i(v^j) \in q^j\}(1 \le j \le w)$. Correspondingly, the formal definitions of volume and access patterns are described as follows.

Definition 1 (Volume Pattern): The volume pattern is a sequence over l MRQs $\mathcal{Q} = \{Q_i\}_{i=1}^{l}$ that reveals the number of results for each query and is defined as $\mathsf{VP}(Q_i) = \{|\mathsf{DB}(Q_i)|\}$.

Definition 2 (Access Pattern): The access pattern is a sequence over l MRQs $\mathcal{Q} = \{Q_i\}_{i=1}^{l}$ that reveals the results for each query and is defined as $\mathsf{AP}(Q_i) = \{\mathsf{DB}(Q_i)\}$.

In this paper, we intend to study the problem of privacy-preserving MRQ with protection for volume and access patterns. According to the above definitions, we define our VHIDX below.

Definition 3 (VHIDX): Given a multi-attribute dataset $\mathsf{DB} = \{d_1, \ldots, d_n\}$ with $id_i = (id_i(v^1), \ldots, id_i(v^w))$ and a MRQ set $\mathcal{Q} = \{Q_i\}_{i=1}^{l}$, $\mathsf{DB}(Q_i) = \{id_i | id_i(v^j) \in q^j\}(1 \le j \le w)$ can be correctly searched for each $Q_i = \{q_i^j = [x_i^j, y_i^j]\}(1 \le i \le l, 1 \le j \le w)$ without leaking $\mathsf{VP}(Q_i) = \{|\mathsf{DB}(Q_i)|\}$ and $\mathsf{AP}(Q_i) = \{\mathsf{DB}(Q_i)\}$.

C. Security Model

We introduce the security model of VHIDX includes five PPT algorithms $\sum = (\mathsf{Setup}, \mathsf{TGen}, \mathsf{Search}, \mathsf{Dec}, \mathsf{Update})$. It guarantees that no extra useful information is leaked to the adversary $\mathcal{A}$ other than what is revealed by the leakage function $\mathcal{L} = \{\mathcal{L}_{\mathsf{Setp}}, \mathcal{L}_{\mathsf{Srch}}, \mathcal{L}_{\mathsf{Updt}}\}$,[1] where $\mathcal{L}_{\mathsf{Setp}}$, $\mathcal{L}_{\mathsf{Srch}}$, and $\mathcal{L}_{\mathsf{Updt}}$ refer to the algorithms Setp, Srch, and Updt, respectively.

We first define the security model of VHIDX by following the simulation-based security definition in [46], [47], and [48]. Specifically, we aim to implement the adaptive security, where the PPT adversary $\mathcal{A}$ can adaptively choose each subsequent query with the leakages of previous queries. The formal adaptive security of VHIDX is defined as follows:

Definition 4: Let $\sum = (\mathsf{Setup}, \mathsf{TGen}, \mathsf{Search}, \mathsf{Dec}, \mathsf{Update})$ be the secure MRQ scheme over encrypted dataset, λ be the security parameter, $\mathcal{L} = \{\mathcal{L}_{\mathsf{Setp}}, \mathcal{L}_{\mathsf{Srch}}, \mathcal{L}_{\mathsf{Updt}}\}$ be the leakages. Based on a PPT adversary $\mathcal{A}$ and a PPT simulator $\mathcal{S}$. The following probabilistic experiments $\mathbf{Real}_{\sum,\mathcal{A}}(1^\lambda)$ and $\mathbf{Ideal}_{\sum,\mathcal{A}}(1^\lambda)$ are defined as follows.

- *$\mathbf{Real}_{\sum,\mathcal{A}}(1^\lambda)$: $\mathcal{A}$ chooses a dataset DB and asks the DO to output the ciphertexts and indexes based on Setup and Update algorithms. Then, $\mathcal{A}$ initiates a polynomial number of queries and asks the DO for the query tokens and encrypted results based on the Search algorithm. Finally, $\mathcal{A}$ outputs a bit b.*
- *$\mathbf{Ideal}_{\sum,\mathcal{A},\mathcal{S}}(1^\lambda)$: $\mathcal{A}$ chooses a dataset DB, and $\mathcal{S}$ simulates ciphertexts and indexes for $\mathcal{A}$ with $\mathcal{L}_{Setp}$. $\mathcal{S}$ performs the update operations with $\mathcal{L}_{Updt}$. Then, $\mathcal{A}$ adaptively initiates a polynomial number of queries. $\mathcal{S}$ simulates ciphertexts and query tokens with $\mathcal{L}_{Srch}$ in each query, which are performed on the simulated index. Finally, $\mathcal{A}$ outputs a bit b.*

$\sum$ is adaptively secure for all PPT adversaries $\mathcal{S}$, there exists a simulator $\mathcal{S}$ such that:

$$\Pr\left[\mathbf{Real}_{\sum,\mathcal{A}}(1^\lambda)\right] - \Pr\left[\mathbf{Ideal}_{\sum,\mathcal{A}}(1^\lambda)\right] \le negl(\lambda),$$

where $negl(\lambda)$ is a negligible function in λ.

The volume-hiding of the returned results guarantees that the adversary $\mathcal{A}$ cannot obtain any useful information from the volume pattern. To define the volume-hiding of VHIDX, we introduce a game between any PPT adversary $\mathcal{A}$ and the DO, the formal game of volume-hiding is defined as follows:

[1]Since the two algorithms TGen and Dec are performed by clients locally, it will not leak any useful information to adversary $\mathcal{A}$.

Definition 5: Let $S_0 = (Q_0, |R(Q_0)|)$ *and* $S_1 = (Q_1, |R(Q_1)|)$ *be two signatures from adversary* $\mathcal{A}$*, where* $|R(Q)|$ *is the length of result volume corresponding to* Q*. Given* m *total records, fixed volume* R*, and a bit* $\mu \in \{0, 1\}$*.*

$\mathbf{Game}^{\mu}_{\mathcal{A},\mathcal{L}}(m, R)$*:* $\mathcal{A}$ *generates two signatures* S_0*,* S_1 *and asks the DO to output the index with the signature* S_μ*. The DO returns* $\mathcal{L}_{Step}$ *to the adversary* $\mathcal{A}$*.* $\mathcal{A}$ *initiates a polynomial number of queries, and the DO generates* $\mathcal{L}_{Srch}$ *for each query. Finally,* $\mathcal{A}$ *outputs a bit* b*.*

Let $\Pr^{\mu}_{\mathcal{A},\mathcal{L}}(m, |R|)$ *be the probability that* $\mathcal{A}$ *outputs* $\mu = 1$ *when playing game* $\mathbf{Game}^{\mu}_{\mathcal{A},\mathcal{L}}(m, R)$*. For all adversaries* $\mathcal{A}$*, the leakage function is volume-hiding if and only if for all queries* $|R(Q)| \in \{1, m\}$*:*

$$\Pr^{0}_{\mathcal{A},\mathcal{L}}(\mathrm{m}, |\mathrm{R}|) = \Pr^{1}_{\mathcal{A},\mathcal{L}}(\mathrm{m}, |\mathrm{R}|).$$

D. Design Goals

We aim to propose a privacy-preserving and volume-hiding MRQ scheme, which enables SGX-aided server to support retrieval services for resource-limited clients over encrypted multi-attribute datasets. The design goals are described as follows:

- **Efficiency**. Clients in our scheme should have low computational and communication overhead, while the server should provide fast and accurate query services.
- **Adaptive security**. Facing the server that may initiate adaptive attacks with the obtained information, our scheme should preserve the confidentiality of outsourced data and query tokens.
- **Pattern protection**. Our scheme should avoid the access pattern and results volume leaking any useful information to the server during queries.

V. The Proposed Scheme

In this section, we first introduce the scheme rational of VHIDX, and then specify its concrete construction.

A. Scheme Rational

To implement volume-hiding MRQs over encrypted multi-attribute datasets, the primary challenge is building a secure and effective data index. Cryptographic primitives for volume-hiding keyword queries, particularly those that hide query result lengths in encrypted multi-maps, have been proposed [15], [16], [17]. These techniques map keyword-matched data records to fixed-size ciphertext blocks and retrieve them with a uniform volume. However, as discussed in Section I, existing volume-hiding techniques are not directly applicable to MRQs over multi-attribute datasets, as they do not address the varying volumes of range queries.

To address this issue, we propose a novel joint index framework consisting of two types of indexes: one for attribute values and another for corresponding data records. The framework operates within a hardware enclave for secure query execution. First, a range-based binary tree index is built on attribute values, with nodes encrypted using ORE primitive to protect data privacy. Second, we propose a secure multi-map structure based on [16]. This framework allows clients to perform batched range queries with fixed-size ciphertext blocks, hiding the query result volumes from the server. Note that the query results are ciphertext blocks of uniform size, preventing the server from inferring the exact volume of the query responses. After retrieving range query results for each attribute via the joint index, we utilize the FNP protocol for private set intersection in the enclave to compute the final result set and return the encrypted output to the clients.

Although the framework ensures volume-hiding, it does not protect access pattern, meaning the server can infer which locations were queried. The existing solution [49] relies on the interactions between DO and the server to refresh access pattern, which is inefficient. We re-encrypt index entries within the enclave during batch queries, eliminating DO-server interactions to improve this. We also design a batch refresh mechanism that caches query results and updates the joint indexes with re-encrypted entries when the cache is full or data updates. This ensures the server cannot link queries to specific results.

The overflow of VHIDX is shown in Fig. 2. In the **Setup** phase, the DO generates PRFs with corresponding keys, and constructs the joint index framework, which is then uploaded to the storage server. In the **TGen** phase, clients generate query tokens based on the query requests and send them to the storage server for search. During the **Search** phase, with the assistance of the Enclave, the server first searches the tree-based index to obtain ciphertext block labels, and then retrieves the matching ciphertext blocks from the multi-maps using the labels. Then, the server computes the query intersection for each attribute and the final results based on the FNP protocol. In the **Decrypt** phase, clients decrypt the results to obtain the matching IDs. During the **Update** phase, the DO can generate update tokens with update requests and send them to the server, which will perform the update operation.

B. Concrete Construction of VHIDX

The concrete construction of VHIDX is described as follows.

1) Setup Phase: According to the scheme rational, Algorithm 1 describes the specific Setup procedure for building joint indexes, performed at the DO side. For each attribute value v_i^j in attribute V^j, the DO first divides the corresponding ID set $\mathsf{DB}(v_i^j)$ into $(\beta + 1)$ blocks, with each block containing a fixed number of p data IDs. Note that the last block in $(\beta + 1)$ is padded to p with random IDs from $\mathsf{DB}(v_i^j)$. Using the FNP protocol, the DO constructs a polynomial $F_{v_i^j}(x) = \prod_{id_i \in DB(v_i^j)} (x - id_i)$ for set $\mathsf{DB}(v_i^j)$. Subsequently, the DO encrypts the coefficients of the polynomial with Pailliar encryption, where each encrypted polynomial $[F_{v_i^j}(x)]_P$ serves as the ciphertexts of the block $\{id_1, \ldots, id_p\}$, donated as $C = [F_{v_i^j}(x)]_P$. Additionally, the DO constructs a label L using a PRF $G(k_1, v_i^j \parallel c_i^j \parallel t_i^j)$ to index the corresponding ciphertexts block C, where c_i^j represents a self-incremental counter, and t represents an index state used to track the accessed times of value v_i^j. Once labels and ciphertext blocks for all attribute values v_i^j in attribute V^j are generated, the DO constructs an encrypted mutli-map I_{mm}^j of attribute V^j containing all tuples

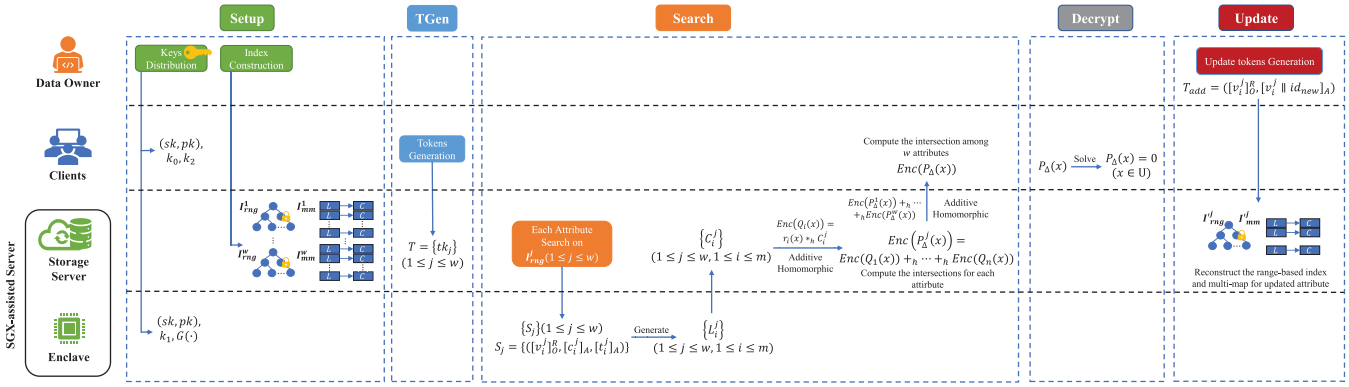


Fig. 2. System overflow of VHIDX.

$\{L, C\}$, capable of mapping $\mathsf{DB}(v_i^j)$ to individual L-C pairs in a volume-hiding way.

To obtain L-C pairs of the value v_i^j in range queries, we also design an encrypted range-based index I_{rng}^j for attribute V^j, as demonstrated from Line 21 to 33 in Algorithm 1. In detail, the DO first constructs each tree node $N[v_i^j]$ with the attribute value v_i^j, the number of L-C pairs c_i^j, and the current index state t_i^j. Then, the DO encrypts the v_i^j with ORE algorithm to obtain the ciphertexts $[v_i^j]_O^L$, $[v_i^j]_O^R$, i.e., $[v_i^j]_O^L = ORE.Enc_L(k_2, v_i^j)$, $[v_i^j]_O^R = ORE.Enc_R(k_2, v_i^j)$. Additionally, the DO encrypts the c_i^j and t_i^j with AES encryption to obtain the ciphertexts $[c_i^j]_A$, $[t_i^j]_A$. With the range-based index I_{rng}^j, the attribute value and its associated information falling within the query range can be searched and sent to the enclave, which is used to generate the labels for I_{mm}^j. Finally, the joint indexes $\{I_{rng}^j, I_{mm}^j\}$ of each attribute V^j are uploaded to the untrusted storage server.

2) TGen Phase: The token generation algorithm, as demonstrated in Algorithm 2, is used for constructing query tokens. Upon determining the query range for each attribute, clients utilize the ORE algorithm to generate the corresponding query token $tk^j (1 \le j \le w)$. Subsequently, clients send the multi-attribute query token $T = \{tk^j\}(1 \le j \le w)$ to the untrusted storage server.

3) Search Phase: Algorithm 3 demonstrates the concrete Search protocol over the secure joint indexes, executed by the server-side. For each attribute V^j, the untrusted storage server compares the ciphertext of each value $[v_i^j]_O^R$ in I_{rng}^j with the query token $([x^j]_O^L, [y^j]_O^L)$ using the $\mathsf{ORE.Cmp}(\cdot)$ algorithm, and if v_i^j satisfies

$$[x^j]_O^L \le [v_i^j]_O^R \le [y^j]_O^L, \tag{4}$$

indicating that it falls within the query range, the encrypted tuple $([v_i^j]_O^R, [c_i^j]_A, [t_i^j]_A)$ in the node $N[v_i^j]$ is added to the set S^j.

After receiving the encrypted tuples, the enclave decrypts them and checks whether each values is cached in $\mathsf{C_{SGX}}$. Specifically, if the attribute value v_i^j is cached in the $\mathsf{C_{SGX}}$, indicating it has been previously queried, the enclave directly acquires the corresponding L-C pairs from $\mathsf{C_{SGX}}$. Otherwise, if the enclave computes the query label $L = G(k_1, v_i^j \| c \| t_i^j)$ and sends it to the untrusted storage server. Subsequently, the server acquires the ciphertext block C of L from I_{mm} and returns it to the enclave.

For each value v_i^j in attribute V^j, the enclave randomly chooses an appropriate degree polynomial $r_i(x)$ and computes $Enc(Q_i(x)) = r_i(x) *_h C$ using the additive homomorphic property. Subsequently, it employs the same homomorphic property to compute the intersection for each attribute V^j as shown in Eq.(5).

$$Enc(P_\Delta^j(x)) = Enc(Q_1(x)) +_h \cdots +_h Enc(Q_n(x)) \tag{5}$$

Finally, the intersection results across all attributes are computed as denoted by Eq.(6)

$$Enc(P_\Delta(x)) = Enc(P_\Delta^1(x)) +_h \cdots +_h Enc(P_\Delta^w(x)), \tag{6}$$

which will be returned to clients.

To mitigate access pattern leakage during queries, VHIDX utilizes the $\mathsf{C_{SGX}}$ cache to store node tuples and L-C pairs of queried attribute values v_i^j, performing the Rebuild algorithm when the cache is full or a new data record is inserted. As shown in Algorithm 4, the enclave updates its index state t_i^j for the accessed attribute values in each dimension, and then rebuilds the L-C pairs v_i^j as well as the encrypted node tuple $([v_i^j]_O^R, [c_i^j]_A, [t_i^j]_A)$ for value v_i^j. The rebuilt L-C pairs and encrypted tuple are then inserted into $I^j{}_{mm}$ and $I^j{}_{rng}$, respectively.

Remark 1: Based on the secure batch refresh algorithm, VHIDX achieves two goals: (I) it guarantees that multiple matched results are cached in parallel, which prevents the adversary from inferring the correlation between the label L and the ciphertext block C; (II) even if no refresh operation occurs due to an under-filled cache or no updates, the adversary can only infer the correlation between L-C with a probability less than $\sum_{i=1}^{w} c_i^j \times (\left|v_i^j\right| + |C|)/|\mathsf{C_{SGX}}|$. *In summary, with the enclave cache* $\mathsf{C_{SGX}}$, *VHIDX can achieve access pattern obfuscation by performing the secure batch refresh algorithm.*

4) Dec Phase: As shown in Algorithm 5, the Dec is performed by clients. In detail, clients first decrypt the encrypted results $Enc(P_\Delta(x))$ and obtain the plaintexts of coefficients $\{a_p, \ldots, a_0\}$ for polynomial $P_\Delta(x)$. Then, clients re-constructs the $P_\Delta(x)$

$$P_\Delta(x) = a_p x^p + a_{p-1} x^{p-1} + \cdots + a_0, \tag{7}$$

Algorithm 1 Setup: Build Encrypted Joint Indexes

input : primary keys $\{sk, k_0, k_1, k_2\}$, secure PRF $G(\cdot)$, fixed block size p and IDs $\mathsf{DB}(v_i^j)$ for value v_i^j
output: encrypted joint indexes $\{I_{rng}^j, I_{mm}^j\}$

DO.Build Multi-maps(v_i^j, $\mathsf{DB}(v_i^j)$)**:**
for *each attribute* V^j **do**
  for *each value* v_i^j **do**
    $\beta \leftarrow \left\lfloor \frac{|DB(v_i^j)|}{p} \right\rfloor$, *Pad* $(\beta+1)$*-th block with random IDs in* $\mathsf{DB}(v_i^j)$ *if needed*;
    Set $\{c_i^j \parallel t_i^j\} = 0$;
    for *each block* **do**
      $L = G(k_1, v_i^j \parallel c_i^j \parallel t_i^j)$;
      $F_{v_i^j}(x) = \prod_{id_i \in \mathsf{DB}(v_i^j)} (x - id_i)$;
      $C = \mathsf{Paillier.Enc}(sk, F_{v_i^j}(x))$, $c_i^j ++$;
      Insert $\{L, C\}$ *into* I_{mm}^j;
    Encrypt (v_i^j, c_i^j, t_i^j);
    Put states $\{[v_i^j]_O^R, [c_i^j]_A, [t_i^j]_A\}$;
  Generate the multi-map I_{mm}^j *for* j*-th attribute*;
Upload $\{I_{mm}^j\}(1 \le j \le w)$ *to the server*;
DO.Build Range-based Indexes(v_i^j, $N[v_i^j]$)**:**
for *each attribute* V^j **do**
  for *each value* v_i^j **do**
    if $N[v_i^j] == NULL$ **then**
      $N[v_i^j] = ([v_i^j]_O^R, [c_i^j]_A, [t_i^j]_A)$;
      Insert new node $N[v_i^j]$ *into* I_{rng}^j;
    else if $[v_i^j]_O^L < [v]_O^R$ **then**
      Run ***Build Range-based Indexes***(v_i^j, N_r);
    else if $[v_i^j]_O^L > [v]_O^R$ **then**
      Run ***Build Range-based Indexes***(v_i^j, N_l);
  Generate the range-based index I_{rng}^j *for* j*-th attribute*;
Upload $\{I_{rng}^j\}(1 \le j \le w)$ *to the server*;

Algorithm 2 TGen: Token Generation Algorithm

input : ORE secret key k_2 and query range $\{q^j\}(1 \le j \le w)$ for each attribute
output: query token T

for *each attribute* V^j **do**
  $q^j = [x^j, y^j]$;
  $tk^j = (\mathsf{ORE.Enc_L}(k_2, x^j), \mathsf{ORE.Enc_L}(k_2, y^j))$;
Upload $T = \{tk^j\}(1 \le j \le w)$ *to enclave*;

and solve its all roots. The query results R consists of each unique ID of data record in DB.

5) Update Phase: As described in Algorithm. 6, clients first generate an update token based on the new data record ID id_{new} and its corresponding attribute value v_i^j, then send it to the enclave. During the update phase, two cases are considered: (I) If the cache $\mathsf{C_{SGX}}$ stores the node tuple (v_i^j, c_i^j, t_i^j) of

Algorithm 3 Search: Secure Batch Query Protocol

input : primary keys $\{sk, k_0, k_1, k_2\}$, secure PRF $G(\cdot)$, enclave cache $\mathsf{C_{SGX}}$, joint indexes $\{I_{rng}, I_{mm}\}$ and query token T
output: encrypted results $Enc(P_\Delta(x))$

Server.Search(T, I_{rng})**:**
for *each attribute* V^j **do**
  Locate range-matched $\{N[v_1^j], \ldots, N[v_n^j]\}$ *from* I_{rng}^j;
  Put each tuple $([v_i^j]_O^R, [c_i^j]_A, [t_i^j]_A)(1 \le i \le n)$ *from* I_{rng}^j *into set* $S_j(1 \le j \le w)$;
return j sets to enclave;
Enclave.Compute($\{q^j\}(1 \le j \le w)$)**:**
for *each attribute* V^j **do**
  $v_i^j = \mathsf{ORE.Dec}(k_2, [v_i^j]_O^R)$;
  if $v_i^j \in \mathsf{C_{SGX}}$ **then**
    Acquire each corresponding ciphertext blocks C *from* $\mathsf{C_{SGX}}$;
  else
    $\{c_i^j, t_i^j\} = \mathsf{AES.Dec}(k_0, [c_i^j]_A, [t_i^j]_A), m = 0$;
    while $m \le c_i^j$ **do**
      $L = G(k_1, v_i^j \parallel m \parallel t_i^j), m++$;
      Cache $\{v_i^j, L\}$ *inside* $\mathsf{C_{SGX}}$;
    Acquire $\{L, C\}$ *with* ***Server.Acquire***(L, I_{mm});
    for *each ciphertext block* C *for value* v_i^j **do**
      Generate a random polynomial $r_i(x)$ *with appropriate degree from* $\mathbb{R}^p[x]$;
      $Enc(Q_i(x)) = r_i(x) *_h C$;
    $Enc(P_\Delta^j(x)) = Enc(Q_1(x)) +_h \cdots +_h Enc(Q_n(x))$;
  $Enc(P_\Delta(x)) = Enc(P_\Delta^1(x)) +_h \cdots +_h Enc(P_\Delta^w(x))$;
return *encrypted intersection result* $Enc(P_\Delta(x))$ *to clients*;
Server.Acquire(L, I_{mm})**:**
for *each attribute* V^j **do**
  for *each label* L **do**
    $\{C\} \leftarrow I_{mm}^j[L]$, *Cache* $\{L, C\}$ *inside* $\mathsf{C_{SGX}}$;
    Delete $\{L, C\}$ *from* I_{mm}^j;
  while $\mathsf{C_{SGX}}$ *is full or inserting new data records* **do**
    Run ***Enclave.Rebuild***$(\mathsf{C_{SGX}})$;

v_i^j and its L-C pairs, the enclave updates the self-incremental counter c_i^j, and re-generates the last ciphertext block (i.e., the $(\beta+1)$-th ciphertext block) using the Pailliar algorithm.[2] (II) If the cache $\mathsf{C_{SGX}}$ does not store the corresponding information of v_i^j, the enclave acquires the encrypted tuple $([v_i^j]_O^R, [c_i^j]_A, [t_i^j]_A)$ from I_{rng}. It then generates the query labels $L_m(1 \le m \le c_i^j)$ with (v_i^j, c_i^j, t_i^j) and acquires L-C pairs from the multi-maps I_{mm}. The enclave updates the c_i^j and the last ciphertext block of v_i^j.

Finally, the enclave re-constructed the node tuple $I_{rng}(v_i^j) = ([v_i^j]_O^R, [c_i^j]_A, [t_i^j]_A)$ for I_{rng} and L-C pairs

[2]Due to the probabilistic property of Pailliar encryption, the re-generated ciphertext block is different from the previous one.

Algorithm 4 Rebuild: Secure Batch Refresh Algorithm

input : primary keys $\{sk, k_0, k_1, k_2\}$, secure PRF $G(\cdot)$, enclave cache $\mathsf{C_{SGX}}$ and joint indexes $\{I_{rng}, I_{mm}\}$
output: rebuilt joint indexes $\{I'_{rng}, I'_{mm}\}$

Enclave.Rebuild($\mathsf{C_{SGX}}$):
for *each attribute* V^j **do**
  for *each cached node tuple* (v_i^j, c_i^j, t_i^j) *and L-C pair* $\{L_m, C_m\}(1 \le m \le c_i^j)$ **do**
    $m = 0$;
    $t_i^j = t_i^j + 1$;
    while $m \le c_i^j$ **do**
      $L'_m = G(k_1, v_i^j \parallel m \parallel t_i^j)$;
      $F_{v_i^j}(x) = \mathsf{Paillier.Dec}(pk, C_m)$;
      $C'_m = \mathsf{Paillier.Enc}(sk, F_{v_i^j}(x))$;
      $m++$;
    Encrypt (v_i^j, c_i^j, t_i^j);
  Fetch $\{L'_m, C'_m\}(1 \le m \le c_i^j)$ *and* $([v_i^j]_O^R, [c_i^j]_A, [t_i^j]_A)$;
Insert $\{L'_m, C'_m\}(1 \le m \le c_i^j)$ *into* $I^j{}_{mm}$;
Insert $([v_i^j]_O^R, [c_i^j]_A, [t_i^j]_A)$ *into* $I^j{}_{rng}$;

Algorithm 5 Dec: Results Decryption Algorithm

input : public key of Paillier pk, encrypted results $Enc(P_\Delta(x))$
output: query results R

```
/*Obtain the plaintexts of
coefficients for polynomial P_Δ(x)*/
```
$\{a_p, \ldots, a_0\} = \mathsf{Paillier.Dec}(pk, Enc(P_\Delta(x)))$;
Re-construct the intersection polynomial with $\{a_p, \ldots, a_0\}$;
Solve all roots of the $P_\Delta(x)$ *and construct result set* R;

$I_{mm}(v_i^j) = \{\{L_m, C_m\}(1 \le m \le c_i^j)\}$ for I_{mm}, then inserts $\{I_{rng}(v_i^j), I_{mm}(v_i^j)\}$ into the joint indexes.

Remark 2: The work [45] highlights the potential for side-channel attacks targeting SGX, where the adversary can exploit accessed enclave pages and branches to infer cached data. Several existing studies [50], [51], [52] have proposed defenses against specific attacks. In VHIDX, both tuples of query values v_i^j and L-C pairs are stored on a single enclave page. Since memory accesses to the same page cannot be distinguished, VHIDX effectively mitigates the risk of side-channel attacks. Consequently, our scheme provides robust protection against side-channel leakage during queries.

VI. Security Analysis

In this section, based on the security definition of VHIDX, we demonstrate the security guarantees of our VHIDX by providing a formal security analysis. Briefly, we construct encrypted joint indexes for multi-attribute datasets to achieve secure and efficient MRQs with the aid of the enclave. Moreover, our proposed secure batch query protocol combined with multi-map indexes can hide the volume of matched results during queries.

Algorithm 6 Update: Secure Data Insertion Protocol

input : primary keys $\{sk, k_0, k_1, k_2\}$, secure PRF $G(\cdot)$, joint indexes $\{I_{rng}, I_{mm}\}$ and new record $\{id_{new}\}$
output: re-constructed $\{I'_{rng}, I'_{mm}\}$

Client.AddToken(v_i^j, id_{new}, k_0, k_2):
$[v_i^j \parallel id_{new}]_A = \mathsf{AES.Enc}(k_0, v_i^j \parallel id_{new})$;
$[v_i^j]_O^L = \mathsf{ORE.Enc_L}(k_2, v_i^j)$;
$T_{add} = ([v_i^j]_O^R, [v_i^j \parallel id_{new}]_A)$;
Enclave.Add(T_{add}, I_{rng}, I_{mm}):
$(v_i^j \parallel id_{new}) = \mathsf{AES.Dec}(k_0, [v_i^j \parallel id_{new}]_A)$;
if $\mathsf{C_{SGX}}$ *stores node tuple* (v_i^j, c_i^j, t_i^j) *and its pairs* $\{L_m, V_m\}(1 \le m \le c_i^j)$ **then**
  $c_i^j = c_i^j + 1$;
  $F_{v_i^j}^{(\beta+1)}(x) = \prod_{id_i \in DB(v_i^j)} (x - id_i)(x - id_{new})$;
  $[F_{v_i^j}^{(\beta+1)}(x)]_P = Pailliar.Enc(sk, F_{v_i^j}^{(\beta+1)}(x))$;
else
  Send $[v_i^j]_O^L$ *to the* I_{rng} *and acquire the encrypted tuple* $([v_i^j]_O^R, [c_i^j]_A, [t_i^j]_A)$;
  Decrypt $([v_i^j]_O^R, [c_i^j]_A, [t_i^j]_A)$;
  for $1 \le m \le c_i^j$ **do**
    $L_m = G(k_1, v_i^j \parallel c_i^j \parallel t_i^j)$;
    Acquire $\{L_m, V_m\}$ *with* ***Server.Acquire***(L_m, I_{mm});
  $c_i^j = c_i^j + 1$;
  $F_{v_i^j}^{(\beta+1)}(x) = \prod_{id_i \in DB(v_i^j)} (x - id_i)(x - id_{new})$;
  $[F_{v_i^j}^{(\beta+1)}(x)]_P = Pailliar.Enc(sk, F_{v_i^j}^{(\beta+1)}(x))$;
Encrypt (v_i^j, c_i^j, t_i^j);
$I_{rng}(v_i^j) = ([v_i^j]_O^R, [c_i^j]_A, [t_i^j]_A)$;
$I_{mm}(v_i^j) = \{\{L_m, C_m\}(1 \le m \le c_i^j)\}$;
Insert $\{I_{rng}(v_i^j), I_{mm}(v_i^j)\}$ *into the joint indexes* $\{I_{rng}, I_{mm}\}$;

We first present definitions of the different patterns leakage. Informally, (I) the search pattern SP refers to the correlation between different queries; (II) the access pattern AP refers to the index entries that match the query; (III) the update pattern UP includes the updated operations and entries. With the notations in the VHIDX, we provide formal definitions of the different patterns:

$$\mathsf{SP} = \{L | (L, T) \in \mathsf{C_{SGX}^{in}}, (L, T) \in \mathsf{C_{SGX}^{out}}\}$$
$$\mathsf{AP} = \{L | (L, T) \in Q, (op, \{L, C\}) \in Q\}$$
$$\mathsf{UP} = \{L | (op, \{L, C\}) \in Q\},$$

note that $\mathsf{C_{SGX}^{in}}$ represents the tuples and *L-C* pairs are cached in the enclave, and $\mathsf{C_{SGX}^{out}}$ represents the *L-C* pairs are acquired from indexes I_{mm}.

Following the design and security definitions of VHIDX, we formally define the leakage functions for setup, search and update phases:

$$\mathcal{L}_{\mathsf{Setp}} = (N_{rng}, \{|v|, |c|, |t|\}, N_{mm}, \{|L|, |C|\})$$
$$\mathcal{L}_{\mathsf{Srch}} = (\mathsf{SP}, \mathsf{UP})$$
$$\mathcal{L}_{\mathsf{Updt}} = (\mathsf{UP})$$

where $|\cdot|$, N_{rng} and N_{mm} are the ciphertext length, the number of encrypted tuples and *L-C* pairs, respectively.

Access Pattern Protection: As designed by VHIDX, the enclave re-constructs the cached *L-C* pairs when the cache $\mathsf{C}_{\mathsf{SGX}}$ is full or updated, ensuring that the untrusted storage server cannot infer the correlation between the initiated queries and the accessed ciphertext blocks, which achieves the access pattern protection.

Update Pattern Protection: In the VHIDX update, the enclave not only updates the index state t_i^j of the attribute value v_i^j in the range-based indexes I_{rng}, but also re-generates the *L-C* pairs in the multi-maps I_{mm} with the new index state $(t_i^j)'$. Due to the difference of index states after the update, the adversary cannot capture the correlation between the newly inserted index entries and the queries initiated in the past.

Lemma 1: $\sum$ is an adaptive $\mathcal{L}$-secure scheme in the random oracle model if G a secure PRF, ORE and Pailliar encryption are semantic-secure primitives.

Proof 1: Based on Definition 4, we prove that there exists a simulator $\mathcal{S}$ for all PPT adversaries $\mathcal{A}$, the output of $\mathbf{Real}_{\sum,\mathcal{A}}(1^\lambda)$ and $\mathbf{Ideal}_{\sum,\mathcal{A},\mathcal{S}}(1^\lambda)$ are computationally indistinguishable.

In detail, for $\mathcal{L}_{\mathsf{Setp}}$, the simulator $\mathcal{S}$ is able to generate the simulated encrypted indexes for each attribute, which is indistinguishable from the real encrypted indexes. The $\mathcal{S}$ generates a binary tree with n nodes and a multi-map with m entries, where each node includes $(|v|, |c|, |t|)$-bit random strings as a node tuple and each entry includes $(|L|, |C|)$-bit random strings as a label-ciphertext pair.

As for $\mathcal{L}_{\mathsf{Srch}}$, the $\mathcal{S}$ can simulate a MRQ and its query results. Specifically, the $\mathcal{S}$ first generates a $|T_{add}|$-bit random string as the simulated token for the simulated indexes. Then, the $\mathcal{S}$ performs a random string to acquire randomly chosen tuples and entries in the binary tree and multi-map, which indicates the same simulated label to correspond to the real labels observed from the $\mathcal{L}_{\mathsf{Srch}}$. The simulated query results are the same number of simulated records acquired from multi-map of each attribute. The $\mathcal{S}$ generates the search pattern SP *by checking whether the results are cached in $\mathsf{C}_{\mathsf{SGX}}^{\mathsf{in}}$, and then the $\mathcal{S}$ generates identical random strings for simulation, thus ensuring consistency among adaptive queries.*

According to the $\mathcal{L}_{\mathsf{Updt}}$, the $\mathcal{S}$ can generate the simulated results when inserting new entries into indexes. Correspondingly, the $\mathcal{S}$ also simulates the cached results with refreshed strings and stores them at the simulated indexes. Based on the semantic security of ORE and Pailliar as well as the pseudo-randomness of PRF, the adversary $\mathcal{A}$ cannot distinguish the simulated interactions and the real ones. Therefore, we say that the outputs of $\mathbf{Real}_{\sum,\mathcal{A}}(1^\lambda)$ and $\mathbf{Ideal}_{\sum,\mathcal{A},\mathcal{S}}(1^\lambda)$ are computationally indistinguishable. The proof is complete. ■

Lemma 2: Leakage functions $\mathcal{L}_{Step}$, $\mathcal{L}_{Srch}$ of our VHIDX are volume-hiding.

Proof 2: The volume-hiding property prevents the adversary $\mathcal{A}$ from acquiring the exact number of records corresponding to any query value and only leaks the fixed volume for a query. As defined in Definition 5, given any two signatures S_0, S_1, the leakage $\mathcal{L}_{\mathsf{Step}}$ of a multi-map in VHIDX *includes m entries, and $\mathcal{L}_{\mathsf{Srch}}$ only includes the fixed volume for any queries.*

TABLE III
PARAMETER DESCRIPTIONS

Main Parameters	Descriptions
l	The encryption parameter of IPE
$\bar{l}$	The encoding parameter of IBF
d	The dimensions of multi-attribute record
N	The size of the dataset
C_{BP}	The computational cost of a bilinear pairing operation

The leakage functions $\mathcal{L}_{\mathsf{Step}}$ and $\mathcal{L}_{\mathsf{Srch}}$ for both signatures are the same. Therefore, the adversary $\mathcal{A}$ should not be able to distinguish any two signatures, i.e.,

$$\mathrm{Pr}^0_{\mathcal{A},\mathcal{L}}(\mathrm{m}, |\mathrm{R}|) = \mathrm{Pr}^1_{\mathcal{A},\mathcal{L}}(\mathrm{m}, |\mathrm{R}|).$$

The proof is complete. ■

VII. PERFORMANCE EVALUATION

Experimental Environment: To evaluate the performance of VHIDX, we implement a prototype in Java and C++, which deployed to the SGX-enabled server with an Intel(R) Core(TM) i5-10300H 2.50GHz CPU and 16RAM, running on Ubuntu (v18.04) [53]. Since the SGX hardware constraints, the physical memory of the SGX is limited to 128MB. In addition, we generate a Redis (v7.2.4) cluster to maintain the encrypted multi-maps. VHIDX utilizes Apache Thrift (v.0.18.1) to perform remote procedure calls (RPC) between the server and clients. For cryptographic primitives, we utilize Intel SGX SDK and OpenSSL (v1.1.0) [54] to achieve 128-bit AES encryption and HAMC-SHA256 PRFs.

Experimental Dataset: We apply a real-world dataset called *Expedia Hotel* [55] in our experiments, while the dataset is pre-processed with functions from the *Clusion* (v0.2.0) framework [56]. The dataset contains over 100K data records, each described by 30 dimensions, and the values on each dimension are appropriately scaled for query compatibility. Also, we generate artificial text as "*Hotel Accommodation Reviews (HAR)*" and randomly assign it to each data record.

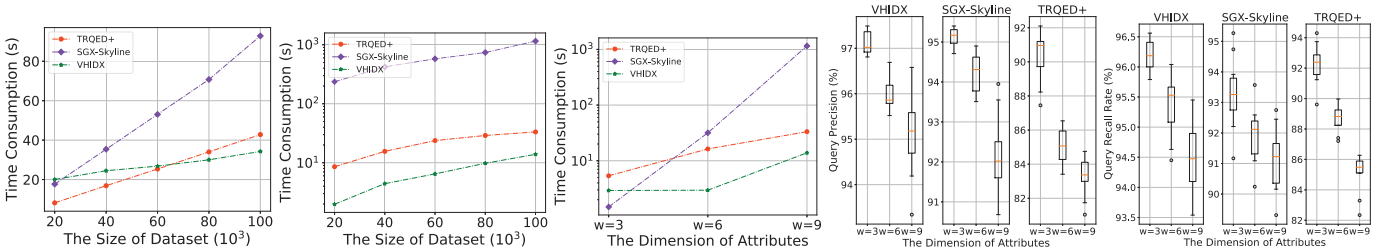
A. Comparison With Prior Arts

To assess the performance of VHIDX relative to other state-of-the-art schemes, we first theoretically analyze the computational cost of each scheme on index generation, token generation, and search, as summarized in TABLE III, IV. Following this, we analyze the overhead associated with index construction and querying among the work and VHIDX under the same experimental settings.

1) Index Construction: The time cost of index construction is shown in Fig. 3(a), where TRQED+, SGX-Skyline, and VHIDX construct their data indexes for $w = 9$ and $N = \{20K, 40K, 60K, 80K, 100K\}$, respectively. As we observed, the time cost of all three schemes grows linearly with dataset sizes. Specifically, SGX-Skyline utilizes HRE to encode w-dimensional data points and organize them into a binary tree, where each leaf node stores an IBF built from its HRE code, while the non-leaf nodes store an IBF built from the union of the child node's HRE codes. In contrast,

TABLE IV
THEOERTICAL COMPARISON WITH PRIOR ARTS

Scheme	Computational complexity of IndexGen	Computational complexity of TokenGen	Computational complexity of search	Computational complexity of Update
Maple [24]	$C_{BP} \times O(d \times N)$	$C_{BP} \times O(d)$	$C_{BP} \times O(d \times logN)$	-
PRQ [31]	$O(d \times l^2 \times N)$	$O(d \times l^3)$	$O(d \times l^2 \times logN)$	-
PMRQ [32]	$O(d \times l^2 \times N)$	$O(d \times l^2)$	$O(d \times l^2 \times logN)$	-
TRQED+ [10]	$O(d \times N)$	$O(d)$	$O(d \times \log N)$	$O(d \times \log N)$
PMRK [33]	$O(d \times l^3 \times N)$	$O(d \times l^2)$	$O(d \times l^2 \times logN)$	-
SGX-Skyline [34]	$O(d \times l \times N)$	$O(d \times l)$	$O(d \times l \times \log N)$	$O(d \times l \times \log N)$
VHIDX	$O(d \times N)$	$O(d)$	$O(d \times N)$	$O(d \times N)$



(a) (b) (c) (d) (e)

Fig. 3. (a) Comparison of Setup. (b) Comparison of Search with diverse datasets. (c) Comparison of Search with diverse attributes. (d) Comparison of query precision. (e) Comparison of query recall.

TRQED+ utilizes ASPE to construct encrypted vector sets for w-dimensional data points and ranges, storing them in leaf and non-leaf nodes of the R-tree. Our VHIDX only uses binary trees to index the attribute values of each dimension, thus the construction cost grows slowly with the size of datasets. Note that the cost of VHIDX becomes the smallest among the three schemes after $N \geq 60K$, indicating its suitability for large-scale datasets.

2) Search: The search time costs of all three schemes with $N = \{20K, 40K, 60K, 80K, 100K\}$ ($w = 9$) and $w = \{3, 6, 9\}$ ($N = 100K$) are shown in Fig. 3(b) and 3(c), respectively. For dataset sizes, all three schemes demonstrate sub-linear growth. Specifically, SGX-Skyline locates the region closest to the query Q based on a binary tree, and then utilizes SGX to filter the n nearest neighbors to Q as search results. However, the cost of the set membership tests based on IBFs during the search is high, and the candidate set to be filtered by SGX increases with dataset growth. On the other hand, TRQED+ utilizes the Secure Scalar Product (SSP) protocol to implement MRQs based on a two-server model. Although queries are more efficiently performed using ASPE, frequent communication between the two servers imposes extra costs, and the two-server model requires higher security assumptions. In contrast, VHIDX performs binary tree search on the server side, followed by label generation and matching on multi-maps with SGX assistance. The server computes the intersection of each dimension and returns it to the client. Since label generation and multi-map matching involve concatenation and PRF computation, SGX incurs lower costs.

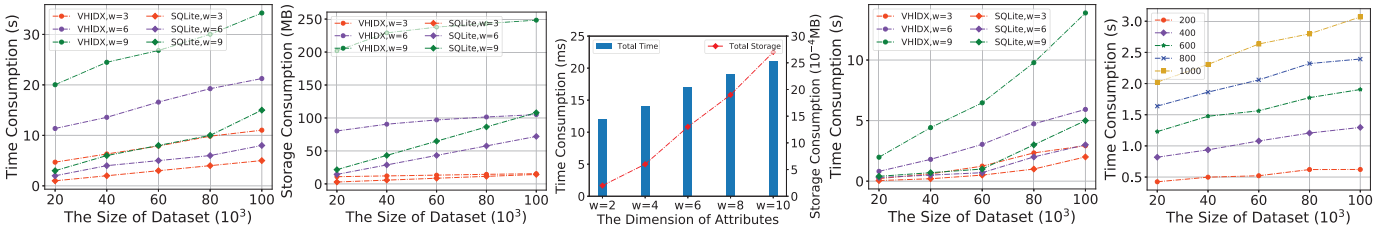
For the attribute dimension, the search cost of SGX-Skyline is lower than that of TRQED+ and VHIDX at $w = 3$, however, SGX-Skyline significantly surpasses the two compared schemes when $w \geq 6$. Essentially, for SGX-Skyline, changes in the search dimension affect the size of the IBFs in the binary tree. In contrast, for TRQED+, the search dimension determines the dimensions of the data and query vectors, thus the attribute dimension impacts the search cost of SGX-Skyline more. Whereas, as VHIDX performs range queries in parallel on each dimension, it is minimally affected by dimension growth.

3) Precision and Recall: In general, precision refers to the proportion of true results in the whole query results, and recall refers to the proportion of true query results in the whole true results. The precision and recall are defined as follows

$$Precision = \frac{TP}{TP + FP}, Recall = \frac{TP}{TP + FN},$$

where TP and FP are the number of true and false results, and FN is the number of true results that have been returned.

The query precision and recall rates vary as the attribute dimension w ranges from 3 to 9, with a fixed dataset size of $N = 100K$. The precision and recall rates for all three schemes decrease with the growth of attribute dimensions. In detail, since the index of TRQED+ is built on R-tree, the "curse of dimensionality" on such multi-dimensional tree structures will significantly impact query efficiency, i.e., query precision and recall will decrease dramatically with the growth of dimensionality (generally $w \geq 8$). In SGX-Skyline, using IBF-based set membership tests introduces false positives, which increase with attribute dimensionality, further decreasing query precision and recall. In VHIDX, when the server computes the intersection of dimensional queries on homomorphic ciphertexts, it generates ciphertext noise and eventually affects the query results. In Fig. 3(d) and 3(e), the experimental results demonstrate that the query precision and recall of VHIDX in



(a) (b) (c) (d) (e)

Fig. 4. (a) Time cost of Setup. (b) Storage cost of Setup. (c) Performance of TGen. (d) Performance of Search. (e) Performance of Update.

diverse dimensions have a distinct superiority compared to other state-of-the-art schemes.

B. Self-Performance Evaluation

To further evaluate the practicality of VHIDX, we provide its self-performance evaluation in four aspects. In particular, we verify the feasibility of VHIDX in a practical database system by comparing it with SQLite (v3.43.1).

1) Index Construction: The main factors affecting index construction are the dimensions of attributes and the size of the dataset. Fig. 4(a), 4(b) measure the time and storage overhead of index construction in VHIDX's joint index compared to the index of SQLite. In VHIDX, we first construct range-based indexes for each attribute and then generate multi-maps based on attribute value labels. In contrast, SQLite utilizes only B+ tree for indexing. Specifically, when $w = 9$ and $N = 100K$, the time and storage of VHIDX are 15s and 108MB, respectively. Compared to constructing plaintext indexes in SQLite, the time and storage overhead of building the joint index in VHIDX are only 2 and 2.5 times that of SQLite, respectively. The evaluation results prove that the overhead of index construction in VHIDX is suitable for practical database systems.

2) Token Generation: In VHIDX, clients only need to encrypt the search range for each dimension using the ORE algorithm to generate query tokens. Therefore, the number of attribute dimensions is the sole factor affecting the token generation overhead. As shown in Fig. 4(c), both the time and storage overheads for token generation increase linearly with the number of attribute dimensions. In detail, when $w = 10$, the token generation time is 21ms, and the storage overhead is 0.0027MB, thus it does not impose any extra burden on clients.

3) Search: To evaluate the practicality of the search protocol, we further measure the MRQ latency on the joint index. The total latency includes the time costs of attribute binary tree search, label generation, multi-maps matching and intersection computation. Fig. 4(d) also compares the search performance between the joint index and SQLite's plaintext index. In VHIDX, the server first searches the range-based index and generates labels via the enclave. Then, the server utilizes the labels to retrieve matched ciphertext blocks from the multi-maps, and finally computes the intersection of the attributes to output the MRQ results. In contrast, SQLite directly performs the search on a plaintext B+ tree. In detail, when $w = 9$ and $N = 100K$, the search time for VHIDX is 13.9s, only 2.8 times that of SQLite. Therefore, in scenarios where security is a high priority, the search efficiency of VHIDX can meet the performance requirements of practical databases.

TABLE V
UPDATE TIME OVERHEAD OF INDEX IN SQLITE

The dimension of updated records is set to $w = 9$					
Dataset	20K	40K	60K	80K	100K
$n = 200$	2.2s	2.4s	2.6s	2.8s	2.9s
$n = 400$	4.3s	4.7s	5.0s	5.3s	5.5s
$n = 600$	6.5s	7.0s	7.5s	7.9s	8.2s
$n = 800$	8.6s	9.3s	9.9s	10.4s	10.8s
$n = 1000$	10.8s	11.7s	12.4s	13.0s	13.5s

4) Update: Similar to practical database systems, VHIDX implements data record insertion. Fig. 4(e) presents the update time under different dataset sizes and numbers of inserted records. For comparison, TABLE V also provides the time overhead of SQLite when inserting new records. The total latency of updates consists of binary tree search, state updates, and multi-maps reconstruction. Specifically, when $w = 9$, $N = 100K$, and inserted records $n = 1000$, the update time is only 3.07s, which is lower than SQLite under the same conditions. When inserting data in VHIDX, since the range-based index only stores attribute values, the update probability is low, and the time overhead is minimal. Additionally, Redis, which stores multi-maps, only needs to update the ciphertext block corresponding to the inserted records. In contrast, SQLite needs to update the entire B+ tree. Therefore, VHIDX has an advantage over SQLite in data insertion, making it more suitable for practical database systems.

VIII. CONCLUSION

In this paper, we propose an efficient and dynamic privacy-preserving MRQ scheme for volume-hiding, called VHIDX, with basic cryptographic primitives. The main challenge is performing efficient MRQs while protecting volume and access patterns over dynamic multi-attribute datasets. To address this challenge, we have designed a novel data structure called joint index using ORE, PRFs, and AES. Based on this index, we combine PSI and hardware SGX to design a secure batch query protocol. In addition, we introduced a secure batch refresh algorithm and an update protocol to hide access pattern. Finally, we prove the security of VHIDX under precisely pre-defined leakages. Comprehensive experimental results on

real datasets show that the query efficiency of our VHIDX is on average 10 times faster than the baseline TRQED+ and 100 times faster than the baseline SGX-Skyline. For future work, we will focus on how to simultaneously protect access, volume, and search patterns to further mitigate information leakage.

REFERENCES

[1] G. Kellaris, G. Kollios, K. Nissim, and A. O'Neill, "Generic attacks on secure outsourced databases," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2016, pp. 1329–1340.

[2] P. Grubbs, M. Lacharité, B. Minaud, and K. G. Paterson, "Learning to reconstruct: Statistical learning theory and encrypted database attacks," in *Proc. IEEE Symp. Security Privacy (SP)*, May 2019, pp. 1067–1083.

[3] M. Lacharité, B. Minaud, and K. G. Paterson, "Improved reconstruction attacks on encrypted data using range query leakage," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2018, pp. 297–314.

[4] J. L. Dautrich and C. V. Ravishankar, "Compromising privacy in precise query protocols," in *Proc. 16th Int. Conf. Extending Database Technol.*, Mar. 2013, pp. 155–166.

[5] E. M. Kornaropoulos, C. Papamanthou, and R. Tamassia, "Response-hiding encrypted ranges: Revisiting security via parametrized leakage-abuse attacks," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2021, pp. 1502–1519.

[6] E. M. Kornaropoulos, C. Papamanthou, and R. Tamassia, "The state of the uniform: Attacks on encrypted databases beyond the uniform query distribution," in *Proc. IEEE Symp. Secur. Privacy (SP)*, Apr. 2020, pp. 1223–1240.

[7] E. A. Markatou and R. Tamassia, "Full database reconstruction with access and search pattern leakage," in *Proc. 22nd Int. Conf. Inf. Secur.*, New York, NY, USA. Cham, Switzerland: Springer, Jan. 2019, pp. 25–43.

[8] H.-I. Kim, H.-J. Kim, and J.-W. Chang, "A secure kNN query processing algorithm using homomorphic encryption on outsourced database," *Data Knowl. Eng.*, vol. 123, Sep. 2019, Art. no. 101602.

[9] N. Cui, X. Yang, B. Wang, J. Geng, and J. Li, "Secure range query over encrypted data in outsourced environments," *World Wide Web*, vol. 23, no. 1, pp. 491–517, 2020.

[10] W. Yang, Y. Geng, L. Li, X. Xie, and L. Huang, "Achieving secure and dynamic range queries over encrypted cloud data," *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 1, pp. 107–121, Jan. 2022.

[11] Z. Gui, O. Johnson, and B. Warinschi, "Encrypted databases: New volume attacks against range queries," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2019, pp. 361–378.

[12] L. Xu, X. Yuan, C. Wang, Q. Wang, and C. Xu, "Hardening database padding for searchable encryption," in *Proc. IEEE Conf. Comput. Commun.*, Apr. 2019, pp. 2503–2511.

[13] L. Blackstone, S. Kamara, and T. Moataz, "Revisiting leakage abuse attacks," in *Proc. Netw. Distrib. Syst. Secur. (NDSS) Symp.*, Feb. 2020, pp. 1–18.

[14] V. Vo, X. Yuan, S.-F. Sun, J. K. Liu, S. Nepal, and C. Wang, "ShieldDB: An encrypted document database with padding countermeasures," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 4, pp. 4236–4252,Apr. 2023.

[15] S. Kamara, T. Moataz, and O. Ohrimenko, "Structured encryption and leakage suppression," in *Proc. Annu. Int. Cryptol. Conf.*, Santa Barbara, CA, USA. Cham, Switzerland: Springer, 2018, pp. 339–370.

[16] S. Kamara and T. Moataz, "Computationally volume-hiding structured encryption," in *Proc. 38th Int. Conf. Theory Appl. Cryptograph. Techn.*, Darmstadt, Germany. Cham, Switzerland: Springer, 2019, pp. 183–213.

[17] S. Patel, G. Persiano, K. Yeo, and M. Yung, "Mitigating leakage in secure cloud-hosted data structures: Volume-hiding for multi-maps via hashing," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2019, pp. 79–93.

[18] K. Ren et al., "HybrIDX: New hybrid index for volume-hiding range queries in data outsourcing services," in *Proc. IEEE 40th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Nov. 2020, pp. 23–33.

[19] F. Liu et al., "Volume-hiding range searchable symmetric encryption for large-scale datasets," *IEEE Trans. Dependable Secure Comput.*, vol. 21, no. 4, pp. 3597–3609, Jul. 2024.

[20] A. Boldyreva, N. Chenette, Y. Lee, and A. O'Neil, "Order-preserving symmetric encryption," in *Proc. Int. Conf. Adv. Cryptol.*, Cologne, Germany. Cham, Switzerland: Springer, 2009, pp. 224–241.

[21] A. Boldyreva, N. Chenette, and A. O'Neill, "Order-preserving encryption revisited: Improved security analysis and alternative solutions," in *Proc. 31st Annu. Cryptol. Conf.*, Santa Barbara, CA, USA. Cham, Switzerland: Springer, 2011, pp. 578–595.

[22] L. Xiao and I. Yen, "A note for the ideal order-preserving encryption object and generalized order-preserving encryption," *Cryptol. ePrint Arch.*, vol. 2012, p. 350, Jan. 2012.

[23] D. Boneh and B. Waters, "Conjunctive, subset, and range queries on encrypted data," in *Proc. Theory Cryptogr. Conf.*, Amsterdam, The Netherlands. Cham, Switzerland: Springer, 2007, pp. 535–554.

[24] B. Wang, Y. Hou, M. Li, H. Wang, and H. Li, "Maple: Scalable multi-dimensional range search over encrypted cloud data with tree-based index," in *Proc. 9th ACM Symp. Inf., Comput. Commun. Secur.*, Jun. 2014, pp. 111–122.

[25] E. Shi, J. Bethencourt, T.-H. H. Chan, D. Song, and A. Perrig, "Multi-dimensional range query over encrypted data," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2007, pp. 350–364.

[26] Y. Lee, "Secure ordered bucketization," *IEEE Trans. Dependable Secure Comput.*, vol. 11, no. 3, pp. 292–303, May 2014.

[27] B. Hore, S. Mehrotra, M. Canim, and M. Kantarcioglu, "Secure multidimensional range queries over outsourced data," *VLDB J.*, vol. 21, no. 3, pp. 333–358, 2012.

[28] Y. Lu, "Privacy-preserving logarithmic-time search on encrypted data in cloud," in *Proc. NDSS*, Jan. 2012, pp. 1–8.

[29] P. Wang and C. V. Ravishankar, "Secure and efficient range queries on outsourced databases using Rp-trees," in *Proc. IEEE 29th Int. Conf. Data Eng. (ICDE)*, Apr. 2013, pp. 314–325.

[30] W. Yang, Y. Xu, Y. Nie, Y. Shen, and L. Huang, "TRQED: Secure and fast tree-based private range queries over encrypted cloud," in *Proc. Int. Conf. Database Syst. Adv. Appl.*, Gold Coast, QLD, Australia. Cham, Swizerland: Springer, 2018, pp. 130–146.

[31] Y. Zheng, R. Lu, Y. Guan, J. Shao, and H. Zhu, "Towards practical and privacy-preserving multi-dimensional range query over cloud," *IEEE Trans. Dependable Secure Comput.*, vol. 19, no. 5, pp. 3478–3493, Sep./Oct. 2021.

[32] Y. Zheng et al., "PMRQ: Achieving efficient and privacy-preserving multidimensional range query in eHealthcare," *IEEE Internet Things J.*, vol. 9, no. 18, pp. 17468–17479, Sep. 2022.

[33] X. Tu, H. Bao, R. Lu, C. Huang, and H.-N. Dai, "PMRK: Privacy-preserving multidimensional range query with keyword search over spatial data," *IEEE Internet Things J.*, vol. 11, no. 6, pp. 10464–10478, Mar. 2024.

[34] J. Wang, M. Du, and S. S. M. Chow, "Stargazing in the dark: Secure skyline queries with SGX," in *Proc. Int. Conf. Database Syst. Adv. Appl.* Cham, Switzerland: Springer, Jan. 2020, pp. 322–338.

[35] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Proc. Int. Conf. Theory Appl. Cryptograph. Techn.* Cham, Switzerland: Springer, 1999, pp. 223–238.

[36] M. J. Freedman, K. Nissim, and B. Pinkas, "Efficient private matching and set intersection," in *Proc. Int. Conf. Theory Appl. Cryptograph. Techn.* Cham, Switzerland: Springer, 2004, pp. 1–19.

[37] K. Lewi and D. J. Wu, "Order-revealing encryption: New constructions, applications, and lower bounds," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2016, pp. 1167–1178.

[38] D. Boneh, K. Lewi, M. Raykova, A. Sahai, M. Zhandry, and J. Zimmerman, "Semantically secure order-revealing encryption: Multi-input functional encryption without obfuscation," in *Proc. Int. Conf. Adv. Cryptol.* Cham, Switzerland: Springer, 2015, pp. 563–594.

[39] N. Chenette, K. Lewi, S. A. Weis, and D. J. Wu, "Practical order-revealing encryption with limited leakage," in *Proc. Int. Conf. Fast Softw. Encryption*, Bochum, Germany. Cham, Switzerland: Springer, 2016, pp. 474–493.

[40] F. Brasser, U. Müller, A. Dmitrienko, K. Kostiainen, S. Capkun, and A.-R. Sadeghi, "Software grand exposure: SGX cache attacks are practical," in *Proc. 11th USENIX Workshop Offensive Technol. (WOOT)*, 2017, pp. 1–12.

[41] J. Götzfried, M. Eckert, S. Schinzel, and T. Müller, "Cache attacks on Intel SGX," in *Proc. 10th Eur. Workshop Syst. Secur.*, Apr. 2017, pp. 1–6.

[42] M. Schwarz, S. Weiser, D. Gruss, C. Maurice, and S. Mangard, "Malware guard extension: Using SGX to conceal cache attacks," in *Proc. Int. Conf. Detection Intrusions Malware, Vulnerability Assessment*, Bonn, Germany. Cham, Switzerland: Springer, 2017, pp. 3–24.

[43] S. Chen, X. Zhang, M. K. Reiter, and Y. Zhang, "Detecting privileged side-channel attacks in shielded execution with Déjàvu," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, Apr. 2017, pp. 7–18.

[44] V. Costan, I. Lebedev, and S. Devadas, "Sanctum: Minimal hardware extensions for strong software isolation," in *Proc. 25th USENIX Secur. Symp. (USENIX Secur.)*, 2016, pp. 857–874.

[45] M.-W. Shih, S. Lee, T. Kim, and M. Peinado, "T-SGX: Eradicating controlled-channel attacks against enclave programs," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2017, pp. 1–11.

[46] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: Improved definitions and efficient constructions," *J. Comput. Secur.*, vol. 19, no. 5, pp. 895–934, 2011.

[47] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic searchable symmetric encryption," in *Proc. ACM Conf. Comput. Commun. Secur. (CCS)*, 2012, pp. 965–976.

[48] D. Cash et al., "Dynamic searchable encryption in very-large databases: Data structures and implementation," in *Proc. Netw. Distrib. Syst. Secur. (NDSS) Symp.*, Feb. 2014, pp. 1–32.

[49] M. Etemad, A. Küpçü, C. Papamanthou, and D. Evans, "Efficient dynamic searchable encryption with forward privacy," in *Proc. Privacy Enhanc. Technol.*, vol. 1, 2018, pp. 5–20.

[50] P. Mishra, R. Poddar, J. Chen, A. Chiesa, and R. A. Popa, "Oblix: An efficient oblivious search index," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2018, pp. 279–296.

[51] W. Zheng, A. Dave, J. G. Beekman, R. A. Popa, J. E. Gonzalez, and I. Stoica, "Opaque: An oblivious and encrypted distributed analytics platform," in *Proc. 14th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2017, pp. 283–298.

[52] S. Sasy, S. Gorbunov, and C. W. Fletcher, "ZeroTrace: Oblivious memory primitives from Intel SGX," in *Proc. Netw. Distrib. Syst. Secur. (NDSS) Symp.*, Feb. 2018, pp. 1–15.

[53] (Jun. 18, 2023). *Intel SGX Linux 2.3.1 Release for Ubuntu 18.04*. [Online]. Available: https://download.01.org/intel-sgx/linux-2.3.1/ubuntu18.04/

[54] (Jun. 18, 2023). *Intel SGX SDK and OpenSSL*. [Online]. Available: https://github.com/intel/linux-sgx

[55] (Jun. 18, 2023). *The Data of Expedia Hotel*. [Online]. Available: https://www.kaggle.com/datasets/vijeetnigam26/expedia-hotel?resource=download

[56] (Jun. 18, 2023). *Clusion From Encrypted Systems Lab Brown University*. [Online]. Available: https://esl.cs.brown.edu/blog/clusion/