Reduce Leakages in Query Processing on Encrypted Multi-dimensional Data with Practicality

Xinle Cao*†
Zhejiang University
xinle@zju.edu.cn

Weiqi Feng*
University of Massachusetts Amherst
weiqifeng@umass.edu

Quanqing Xu
OceanBase, AntGroup
xuquanqing.xqq@oceanbase.com

Chuanhui Yang
OceanBase, AntGroup
rizhao.ych@oceanbase.com

Jian Liu§
Zhejiang University
liujian2411@zju.edu.cn

Kui Ren
Zhejiang University
kuiren@zju.edu.cn

***Abstract*—Encrypted databases (EDBs) have been a prominent focus in the database community with the rise of cloud computing. However, many challenging open problems in EDBs make most existing solutions undeployable in real-world applications. One significant issue is how to avoid dangerous unexpected leakages when querying encrypted multi-dimensional data. For example, given one query SELECT $user\_id$ FROM T WHERE $\mathsf{Age} = 30$ AND $\mathsf{Salary} = 1000$, the minimal leakage should be the records that satisfy both conditions. However, most existing EDBs reveal separate frequency leakages: 1) records that satisfy $\mathsf{Age} = 30$, and 2) records that satisfy $\mathsf{Salary} = 1000$. As numerous works have shown, frequency leakage is very dangerous for EDBs, making it crucial to reduce additional frequency leakage when querying multi-dimensional data. In this paper, we build upon a line of works, including the SOTA work in ICDE 2022, to address the additional leakage. Compared to prior works, our proposed approach elegantly eliminates leakage in a more efficient and flexible manner. The experimental results demonstrate that our work outperforms the SOTA work in a series of common queries over multi-dimensional data, such as equality tests on a single table and equi-joins over two tables where each table consists of multiple columns to be filtered.**

***Index Terms*—Encrypted Database, Multi-dimensional Data, Functional Encryption**

I. Introduction

Encrypted database (EDB) [22, 29, 43, 44, 52] is a promising direction motivated by the popularity and convenience of cloud computing [24]. The data owner (i.e., the client) wishes to store its database in the could to save costs associated with database management software and purchasing equipment. Due to the concern about data security and privacy, the client needs to encrypt the database to avoid the untrusted cloud server learning sensitive information. However, the client still wants to execute SQL queries over the encrypted data efficiently, which is not allowed by the traditional *all-or-nothing* encryption. To this end, numerous novel encryption techniques [2, 5, 10, 25] and encrypted database systems [17, 43, 44, 56] have been proposed to address this practical and challenging scenario.

a) Challenge: Despite the surprising advances in EDBs over the last two decades, many open problems remain unsolved. One of the most important and challenging problem is how to avoid dangerous unexpected leakages when querying multi-dimensional data. To illustrate this issue, consider a database consisting of multiple columns denoted by $\{A_1, A_2, ..., A_m\}$ within a single table and the client issues the following typical query towards the database:

$$\text{SELECT * FROM } T \text{ WHERE } A_1 = a_1 \text{ AND } A_2 = a_2 \\ \text{AND } \cdots \text{ AND } A_m = a_m.$$

Plenty of existing EDBs [35, 36, 43, 44] process such query by separately filtering these columns and then overlapping their results. This is because there have been many well-studied encryption techniques [2, 5, 25] for processing a single column and they can be trivially applied here. Specifically, they process each column to get the result sets $\{S_1, S_2, ..., S_m\}$ where S_i is the record set corresponding to the filter on column A_i. Then they return the final result set S to the client via overlapping these sets:

$$S = \bigcap_{i=1}^{m} S_i.$$

Clearly, while the client may hope leaking only the final result set S to the server as the **necessary leakage** [26, 47], the above process unexpectedly reveal much more sensitive information, i.e., the result of processing each single column. To explain why this additional leakage is dangerous, we give two typical threats based on it as below[1]:

- *Frequency leakage:* The above process reveals the frequency information about a_i in column A_i which has been well-known as dangerous leakage [31, 40] in EDB. For example, suppose column $A_1 = \mathsf{Gender}$ and $a_1 = Female$, then the server directly knows all records of females. As there may be only two distinct values for gender, this column has been nearly decrypted. Instead, the set S may include only several records of females since the filter on other columns,

§Jian Liu is the corresponding author.

*These authors contributed equally to this work.

†Work done during the internship at Ant Research.

[1] The leakage of S_i can be ignored if it includes all records in the database.

the leakage to the gender column should have been much less.

- *Correlation leakage:* The above process can reveal data correlation between columns. For example, if S_1 and S_2 are always nearly identical, then the server can infer the data dependency [55] between A_1 and A_2 with high probability. Recall data correlation has been widely known as valuable information [12] and also another dangerous leakage [16] to EDB, the client definitely wants to control such leakage.

b) Expensive prior works: While some works [26, 43, 47] have tried to process queries over multi-dimensional data with less leakages, they often achieve that in a impractical way. Poddar et al. [43] remove the overlap in the server side to the client side. In their work, each column is *separately* stored with encrypted corresponding index such that the server cannot know the overlap of any two result sets S_i and S_j. However, the client has to retrieve all result sets $\{S_1, S_2, ..., S_m\}$ to do the overlap locally. As the result set S_i possibly is much larger than the final overlapped set S, this protocol is much expensive in production when processing multiple columns. Moreover, it still reveals the size of each result set to the server, i.e., $\{|S_1|, |S_2|, ..., |S_m|\}$. This is also can be regarded as weakened frequency leakages, which is helpful for the server to recovering the database [31].

The more advanced works appear in [26, 47] which achieve a stricter security design goal: **leaking nothing besides the final overlapped set** S. Although these works are initially designed to complete equi-join with meaningful security guarantees, they require the filter on multiple columns should also follow the goal above. To do that, they adopt the well-developed attribute-based encryption [27] (ABE) and inner-product functional encryption (IPFE) [32]. This actually results in expensive costs and impractical disadvantages in their constructions: as the ABE and IPFE schemes used in them are not specialized to the filter on multiple columns, these schemes suffer from some impracticality when directly applying here with some minor modifications. For example, the SOTA work [47] in ICDE 2022 applying IPFE has to operate on the whole m columns even if the query only requires operating a subset of these m columns, i.e., the scheme in [47] has to filter each column, incurring $O(mn)$ computational complexity where n is the number of rows in the database even if the query is as simple as SELECT * FROM T WHERE $A_1 = a_1$. This impracticality results from the design of IPFE, which requires operations must be done over all dimensions (i.e., m here) the client sets in the beginning.

c) Our contributions: In this work, we follow prior works [26, 47] to achieve the strict security design goal on filtering multi-dimensional columns but **in a more efficient and flexible way to enhance impracticality**. In detail, we design a new totally new encryption scheme for filtering over multiple columns to achieve 1) leaking only the final overlapped set S to the server and 2) enabling the calculation to be done only in columns filtered by the query. The flexibility makes our scheme much more efficient than prior works especially in very large databases which can consist of hundreds of columns. Surprisingly, our scheme can be **further extended to achieve more operations** on multi-dimensional data with similar strict security guarantees. Firstly, consistent with [47], it can be extended to perform (faster) equi-join over two tables which are filtered by queries in the same time. In other words, it significantly improves the efficiency of [47]. Secondly, it can be modified to support queries like

$$\text{SELECT * FROM T WHERE } \textstyle\sum_{j=1}^{k} w_j A_{i_j} = p.$$

where $\{i_1, i_2, ..., i_k\} \subset [m]$, $\{w_1, ..., w_k\} \in \mathbb{N}^k$ and constant p is decided by the client. Such queries are important as they cover equality tests between columns and on aggregation of columns. To the knowledge, our scheme is **the first** to achieve such queries with leaking only the final test results. We summarize our contributions as below:

- *Practical multi-dimensional filter.* We propose a new encryption scheme for queries filtering on multi-dimensional data. It not only achieves the same strict security guarantees as the state-of-the-art (SOTA) work [47], but also offers significantly improved efficiency and flexibility.
- *More multi-dimensional operations.* We extend our scheme to support more operations over multi-dimensional data including 1) equi-join over two filtered tables, 2) equality test between columns, and 3) equality test over the aggregation of multiple columns. Notably, our extended scheme achieve all these operations with strict security guarantees.
- *Thorough evaluation.* We implement and evaluate the SOTA work [47] and our schemes. The experimental results show for all operations both our schemes and [47] support, our schemes significantly outperform [47] across three benchmarks. It is expected that with larger databases consisting of more columns, the speedup can be much greater than what is shown in Section VII.

II. Preliminaries

a) Notations: The consecutive set $\{1, 2, .., n\}$ is denoted by $[n]$ or $\mathbb{Z}_n$. The positive integer set $\{1, 2, ...\}$ is represented by $\mathbb{N}^+$. Given two sets A and B, we define $A \backslash B := \{x \in A | x \notin B\}$. All algorithms in this paper are assume to be *efficient*, i.e., they can be run in probabilistic polynomial time (PPT). We use bold lowercase letters to denote vectors (e.g., $\mathbf{x}$, $\mathbf{y}$). $\|\mathbf{x}\|_2$ denotes the Euclidean norm of vector $\mathbf{x}$, i.e., suppose $\mathbf{x} = \{x_1, ..., x_m\}$, then $\|\mathbf{x}\|_2 = \sqrt{x_1^2 + \cdots + x_m^2}$. For a map M that stores key-values pairs, we use $y \leftarrow M[x]$ to denote retrieving the value y corresponding to the key x from the map M. Similarly, $M[x] = y$ represents either saving the new pair (x, y) in the map M or updating the value associated with x to y, if x already exists in M.

A. Bilinear Groups

Here we review some basic notions about *(asymmetric) bilinear groups* [8, 28, 39]. Given two distinct groups with a prime order q denoted by $(\mathbb{G}_1, \mathbb{G}_2)$, suppose $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$ are generators of $\mathbb{G}_1$ and $\mathbb{G}_2$. Then a function $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ is defined to map elements in $(\mathbb{G}_1, \mathbb{G}_2)$ to the target group $\mathbb{G}_T$ whose order is still q. Consistent with the style of Kim et al. [32], we write the group operations in the groups *multiplicatively* and use 1 to denote the multiplicative identity. That means the product of any element in $\mathbb{G}_\gamma$ with $1 \in \mathbb{G}_\gamma$ is equal to itself where $\gamma \in \{1, 2, T\}$. Formally, a tuple $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, e)$ is an asymmetric bilinear group if the properties below hold on it:

- *Computable:* Both the groups operations in the three groups and the map e are efficiently computable.
- *Non-degenerate:* $e(g_1, g_2) \neq 1$.
- *Bilinear:* $\forall x, y \in \mathbb{Z}_q, e(g_1^x, g_2^y) = e(g_1, g_2)^{xy}$.

For ease of presentation, given a vector $\mathbf{v} = (v_1, ..., v_n) \in \mathbb{Z}_q^n$ where $n \in \mathbb{N}^+$ and a group $\mathbb{G}$ with prime order p and generate $g \in \mathbb{G}$, we use $g^{\mathbf{v}}$ to denote $(g^{v_1}, ..., g^{v_n})$. Naturally, for operations on vectors, the scalar operation implies $(g^{\mathbf{v}})^k = g^{(\mathbf{v}k)}$ where $k \in \mathbb{Z}_q$ and vectors additions has $g^{\mathbf{v}} \cdot g^{\mathbf{w}} = g^{\mathbf{v}+\mathbf{w}}$ where $\mathbf{v}, \mathbf{w} \in \mathbb{Z}_q^n$. In particular, the pairing operation describes the inner product of vectors:

$$e(g_1^{\mathbf{v}}, g_2^{\mathbf{w}}) = \Pi_{i \in [n]} e(g_1^{v_i}, g_2^{w_i}) = e(g_1, g_2)^{\langle \mathbf{v}, \mathbf{w} \rangle}.$$

B. The Generic Group Model

Following [32, 47], we prove our constructions in this paper secure under the generic model of bilinear groups [6, 7], an extension of the generic group model [41, 48] (GGM). under this model, all accessed are replaced and controlled by *handles*. The adversary is allowed to access the (stateful) oracle which implements the group operations including the pairing operation when considering the bilinear groups. This oracle keeps a mapping from handles to group elements inside. Instead of specific group elements, the oracle always answers handles to operations issued by the adversary. The handles can be random strings to guarantee the group operations are used only as a black-box. Therefore, if an adversary cannot break the security under GGM, then it cannot break the security when only applying group operations as a black-box. We refer to [1, 19, 32] for more discussion about GGM, in this paper, we are consistent with [47] in ICDE 2022 to propose the security guarantee based on this model. The generic group model denoted by BG provides the stateful oracle $\mathcal{G}$ with the following interfaces:

- $\mathsf{BG.Setup}(1^\lambda)$ takes the security parameter λ as input and outputs a prime q sampled with λ bits. Besides, the oracle $\mathcal{G}$ setups three empty maps (M_1, M_2, M_T) inside as the initial state so that any subsequent Setup will fail.
- $\mathsf{BG.Encode}(\mathbf{x}, \ell)$ take a vector $\mathbf{x} \in \mathbb{Z}_q^*$ and the label $l \in \{1, 2, T\}$ as inputs, outputs the encoding of this vector according to l. The oracle $\mathcal{G}$ produces a fresh and random handle $h_i \leftarrow\!\!\$\ \{0, 1\}^\lambda$ for each x_i. Finally, it outputs $(h_1, ...)$ and sets $M_l[h_i] = x_i$.
- $\mathsf{BG.Op}(h_1, h_2, \ell)$ takes two handles (h_1, h_2) and the label $l \in \{1, 2, T\}$ as inputs. The oracle $\mathcal{G}$ first checks that both h_1 and h_2 are in M_l, otherwise, it outputs $\perp$ to indicate invalid operations. After passing the check, $\mathcal{G}$ returns a fresh handle $h \leftarrow\!\!\$\ \{0, 1\}^\lambda$ and sets $M_l[h] = M_l[h_1] + M_l[x_2]$.
- $\mathsf{BG.Pair}(h_1, h_2)$ takes two handles (h_1, h_2) as inputs. The oracle $\mathcal{G}$ first checks if $M_1[h_1]$ and $M_2[h_2]$ exists. If so, $\mathcal{G}$ returns a fresh handle $h \leftarrow\!\!\$\ \{0, 1\}^\lambda$ and sets $M_T[h] = M_1[h_1] \cdot M_2[h_2]$. Otherwise, $\mathcal{G}$ returns $\perp$ to indicate that this is an invalid operations.
- $\mathsf{BG.ZT}(x, \ell)$ takes a value x and the label $l \in \{1, 2, T\}$ as inputs to check if $x = 0$. The oracle $\mathcal{G}$ checks if $\exists h \in M_\ell$ such that $M_\ell[h] = x$, outputs *true* if so and *False* otherwise.

C. Polynomial Functions

As introduced in [32, 57], $\mathsf{BG.Encode}(\mathbf{x}, l)$ also can be regarded as instantiating n variables $(x_1, .., x_n)$ while other operations except $\mathsf{BG.ZT}$ are specifying new polynomials. For example, $\mathsf{BG.Op}(h_1, h_2, 1)$ where $M_1[h_1] = x_1$ and $M_1[h_2] = y_1$ creates a new polynomial $x_1 + y_1$ from the two variables. The interface $\mathsf{BG.ZT}$ is used to evaluate if the polynomials tested are equal to 0 with specific variable under some distributions. Here we introduce an important Lemma used in this paper for polynomial evaluation.

Lemma 1. *(Schwartz-Zippel [46, 58] adapted in [32]) Fix a prime q and let $f \in \mathbb{Z}_q[x_1, \cdots, x_n]$ be an n-variate polynomial with total degree at most t and which is not identically zero. Then,*

$$Pr[x_1, \cdots, x_n \xleftarrow{R} \mathbb{Z}_q : f(x_1, \cdots, x_n) = 0] \leq \frac{t}{q}. \qquad (1)$$

III. Security Model

A. Problem Overview

a) Basic operations: Consistent with popular works [3, 36, 43, 44], we consider a relational database $\mathcal{DB}$ consisting of one or multiple tables. Suppose a table T_A in $\mathcal{DB}$ includes n_A rows and m_A columns. And each row and column represent a record and an attribute, respectively. The columns in tale T_A are denoted by $\{A_1, A_2, ..., A_{m_A}\}$ and we focus on queries that operate on multiple columns at the same time. Formally, we mainly study filter queries as below:

SELECT * FROM T_A WHERE $A_{i_1} \in s_1$ AND $\cdots$ AND $A_{i_j} \in s_j$.

Where $\{i_1, i_2..., i_j\} \subset [m_A]$. In particular, while $s_k (k \in [j])$ should be allowed to include unlimited values for general queries in reality, here we follow [47] to require $|s_k|$ to be small for efficiency. Even under this limitation, such queries still cover plenty of realistic applications such as filtering age and gender, searching keywords, and picking up employees in several departments.

b) Extended operations: Besides the basic and crucial filter query above, we also support more operations in this paper including equi-join and equality tests 1) between columns and 2) for column aggregation. Here we describe their forms in this work as below:

- **Equi-join** connects two tables denoted by (T_A, T_B) which may consist of multiple columns and these columns are required to be filtered before the join:

SELECT * FROM T_A JOIN T_B ON $A_1 = B_1$ WHERE $(A_{i_1^A}, ..., A_{i_{j_A}^A}) \in s_1^A \times \cdots \times s_{j_A}^A$ AND $(B_{i_1^B}, ..., B_{i_{j_B}^B}) \in s_1^B \times \cdots \times s_{j_B}^B$.

where $\{i_1^A, ..., i_{j_A}^A\} \subset \{2, ..., m_A\}$ and $\{i_1^B, ..., i_{j_B}^B\} \subset \{2, ..., m_B\}$

- **Equality test** is a typical query in databases and its simple form has been well covered by the filtering query above. Here we aim for two more complex equality tests [13, 53]. The first is the test between columns like
  $$\text{SELECT} * \text{FROM } T_A \text{ WHERE } A_{i_1} = A_{i_2}$$
  where $i_1, i_2 \in [m_A]$. This captures plenty of practical applications, e.g., a delivery company may want to know if a package is transferred within the same city or area (SEND_CITY=RECEIVE_CITY). The second is the test for **weighted** column aggregations. Given the weights $\{w_1, ..., w_k\} \in \mathbb{N}^k$, we address the equality test:
  $$\text{SELECT} * \text{FROM } T_A \text{ WHERE } \textstyle\sum_{j=1}^{k} w_j A_{i_j} = p$$
  where $\{i_1, i_2, ..., i_k\} \subset [m_A]$ and constant p are decided by the client. As far as we know, we are the first to achieve both (weighted) column aggregation and equality tests on the sums without the costly fully homomorphic encryption (FHE) while leaking only the test results. The readers may notice this query can cover the equality test between columns via setting $\{w_{i_1}, w_{i_2}\} = \{1, -1\}$ and $p = 0$. We distinguish them to emphasize their differences in the main application scenarios they are designed for.

B. Security Model

a) Adversary: In this paper, we adopt the widely accepted standard security model [12, 37, 43] in EDB. The client uploads its encrypted database $\mathcal{DB}$ to the server and it requires the cloud encrypted database to support all operations mentioned in Section III-A. We consider the adversary to be *honest-but-curious*, so it does not try to either violate the protocol predefined by the client or invade the client to issue queries. But it can access the server during the whole protocol execution, i.e., it can observe anything available in the server during a long period of time [43]. It is assumed to steal sensitive information about the database from all its views inside the server. Such an adversary can be exactly captured by the untrusted cloud server [43, 52].

b) Functional Encryption: In this work, we adopt *functional encryption* (FE) to achieve operations in Section III-A over multi-dimensional data. FE refers to encrypting data but allowing well-defined leakage to the untrusted server for query processing. A kind of famous FE is *order-revealing encryption* [9, 35] which reveals order information to process range queries. Formally, we define a FE encryption scheme $\Pi = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ over a message space $\mathbb{Z}_q^m$ as below:

- $\mathsf{Setup}(1^\lambda) \to (\mathsf{pp}, \mathsf{msk})$: On input a security parameter λ, the setup algorithm outputs the public parameters pp and the master secret key msk.
- $\mathsf{KeyGen}(\mathsf{msk}, \mathbf{y}) \to \mathsf{sk}_\mathbf{y}$: On input the master secret key msk and a vector $\mathbf{y} \in \mathbb{Z}_q^{t_1}$, the key generation algorithm outputs a functional secret key $\mathsf{sk}_\mathbf{y}$.
- $\mathsf{Encrypt}(\mathsf{msk}, \mathbf{x}) \to \mathsf{ct}_\mathbf{x}$: On input the master secret key msk and a vector $\mathbf{x} \in \mathbb{Z}_q^{t_2}$, the encryption algorithm outputs a ciphertext $\mathsf{ct}_\mathbf{x}$.
- $\mathsf{Decrypt}(\mathsf{pp}, \mathsf{ct}, \mathsf{sk}) \to z$: On input the public parameters pp, a functional secret key sk, and a ciphertext ct, the decryption algorithm either outputs a message $z \in \mathbb{Z}_q$ or a special symbol $\perp$.

This work aims to design new novel FE schemes to address all operations in Section III-A with strict security and feasible practicality. So we define the correctness of our new schemes with the necessary leakage corresponding to different operations. Given a FE scheme Π, letting $(\mathsf{pp}, \mathsf{msk}) \leftarrow \Pi.\mathsf{Setup}(1^\lambda, S)$, $\mathsf{sk}_\mathbf{y} \leftarrow \Pi.\mathsf{KeyGen}(\mathsf{msk}, \mathbf{y})$, $\mathsf{ct}_\mathbf{x} \leftarrow \Pi.\mathsf{Encrypt}(\mathsf{msk}, \mathbf{x})$, then we say it is correct for muti-dimensional *filter* if for any two vectors $(x, y) \in \mathbb{Z}_q^{t_2} \times \mathbb{Z}_q^{t_1}$,

$$\Pr[\Pi.\mathsf{Decrypt}(\mathsf{pp}, \mathsf{ct}_\mathbf{x}, \mathsf{sk}_\mathbf{y}) = \mathsf{Filter}(\mathbf{x}, \mathbf{y})] = 1 - \mathsf{negl}(\lambda)$$

where where $\mathsf{negl}(\lambda)$ denotes a function negligible in λ and

$$\mathsf{Filter}(\mathbf{x}, \mathbf{y}) = \begin{cases} 1, & \mathbf{x} \in S_\mathbf{y} \\ 0, & \mathbf{x} \notin S_\mathbf{y} \end{cases}$$

where $S_\mathbf{y} := (s_1, s_2, ..., s_{t_1})$ are the filter sets prepared for vector $\mathbf{x}$ (i.e., the record) and described by the verctor $\mathbf{y}$. For equality tests between columns or on column aggregations, we say Π is correct if for any two vectors $(x, y) \in (\mathbb{Z}_q^m)^2$,

$$\Pr[\Pi.\mathsf{Decrypt}(\mathsf{pp}, \mathsf{sk}_\mathbf{y}, \mathsf{ct}_\mathbf{x}) = \mathsf{Aggre}(\mathbf{x}, \mathbf{y})] = 1 - \mathsf{negl}(\lambda)$$

where

$$\mathsf{Aggre}(\mathbf{x}, \mathbf{y}) = \begin{cases} 1, & \displaystyle\sum_{j=1}^{k} w_j x_{i_j} = p \\ 0, & \displaystyle\sum_{j=1}^{k} w_j x_{i_j} \neq p \end{cases}$$

where $\{i_1, i_2, ..., i_k\} \subset [t_2]$, $\{w_1, ..., w_k\} \in \mathbb{Z}_q^k$, and constant p are decided by the vector $\mathbf{y}$. In FE, the results of these functions are the only leakages allowed to be revealed. Any additional information like the specific filter set s_1 in Filter or p in Aggre should not be known to the adversary.

c) Security Notion: Now we can introduce the security notions for our new FE schemes under the passive adversary. We adopt the simulation-based security like [32, 47]. Intuitively, the simulation-based notion below implies that we do not leak any information about the data besides the necessary query results [11, 42], i.e., the outputs of Filter or Aggre.

Definition 1 (SIM-Security)**.** *Given a FE scheme* $\Pi = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ *over a message space* $\mathbb{Z}_q^m$*, we say* Π *is SIM-secure on function* $\mathsf{F} : \mathbb{Z}_q^m \times \mathbb{Z}_q^m \to \{0, 1\}$ *if for any PPT adversaries* $\mathcal{A}$*, there exists a PPT simulator* $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3)$ *such that the outputs of the following experiments are computationally indistinguishable:*

1) $Real_\mathcal{A}(1^\lambda)$
   - $(\mathsf{pp}, \mathsf{msk}) \leftarrow \Pi.\mathsf{Setup}(1^\lambda)$
   - $b \leftarrow \mathcal{A}^{\mathcal{O}_{\Pi.\mathsf{KeyGen}}(\mathsf{msk}, \cdot), \mathcal{O}_{\Pi.\mathsf{Encrypt}}(\mathsf{msk}, \cdot)}(\lambda)$
   - *output* b
2) $Sim_{\mathcal{A},\mathcal{S}}(1^\lambda)$
   - *initialize* $\mathcal{X} \leftarrow \emptyset$ *and* $\mathcal{Y} \leftarrow \emptyset$

- $(\mathsf{pp}, \mathsf{st}^2) \leftarrow \mathcal{S}_1(1^\lambda)$
- $b \leftarrow \mathcal{A}^{\mathcal{O}'_{\Pi.\mathsf{KeyGen}}(\mathsf{st},\cdot), \mathcal{O}'_{\Pi.\mathsf{Encrypt}}(st,\cdot)}(\lambda)$
- *output b*

where the oracles $\mathcal{O}_{\Pi.\mathsf{KeyGen}}(\mathsf{msk}, \cdot)$ *and* $\mathcal{O}_{\Pi.\mathsf{Encrypt}}(\mathsf{msk}, \cdot)$ *represent the real key generation and encryption algorithms in FE while* $\mathcal{O}'_{\Pi.\mathsf{KeyGen}}(\cdot)$*,* $\mathcal{O}'_{\Pi.\mathsf{Encrypt}}(\cdot)$ *are key generation and encryption interfaces simulated by the simulators. Suppose two vectors* $\mathbf{x}$ *and* $\mathbf{y}$ *are called by* $\mathcal{O}_{\Pi.\mathsf{KeyGen}}(\mathsf{msk}, \cdot)$ *and* $\mathcal{O}_{\Pi.\mathsf{Encrypt}}(\mathsf{msk}, \cdot)$*, respectively. Then the simulators only observe and share the information of* $\mathsf{F}(\mathbf{x}, \mathbf{y})$*. We refer to [11, 32] for more details about the simulators.*

To rule out the equi-join operation, we adopt the identical security notion as [47]. Let λ be the security parameter of the encryption schemes. Let $Q = \{q_1, \ldots, q_\mu\}$ be a sequence of join queries where $\mu = \mathsf{poly}(\lambda)$. Let $\sigma(q_i)$ be the result of the equi-join query q_i, i.e., $\sigma(q_i) = \{(r_1, r'_1), \ldots, (r_\nu, r'_\nu)\}$ is the set of equality pairs between rows $r_{i,j}$ and $r'_{i,j}$ (which can be from the same table). We define the trace $\tau(Q) = \{n, m, \sigma(q_1), \ldots, \sigma(q_\nu)\}$.

Definition 2. *(SIM-Security for Join Encryption) We say a Join Encryption is SIM-secure, if there exists a simulator* $\mathcal{S}(\tau(Q))$ *that given the trace* $\tau(Q)$ *such that the following two experiments are computationally indistinguishable in the security parameter* λ*:*

1) $Real_{\mathcal{A}}(1^\lambda)$
   - $(pp, msk) \leftarrow$ *IPE.Setup*(1^λ)
   - $b \leftarrow \mathcal{A}(VIEW_{DBMS}(Q))$
   - *output b*
2) $Sim_{\mathcal{A},\mathcal{S}}(1^\lambda)$
   - $(pp, st) \leftarrow$ *Setup*$'(1^\lambda)$
   - $b \leftarrow \mathcal{A}(\mathcal{S}(\tau(Q), st))$
   - *output b*

Besides, notice our FE schemes are used to only reveal the results of the required function (Filter and Aggre). They cannot be used even by the client to recover plaintexts. So the client will bind each FE ciphertext with the corresponding semantically secure ciphertexts [30] used to recover underlying plaintexts, which is common in EDBs [20, 43].

IV. Multi-dimensional Filter

In this section, we propose our FE schemes for multi-dimensional filter operations. We first introduce the most significant advances of our work in enhancing practicality. Then we give the specific simple and efficient constructions.

A. Enhancing Practicality

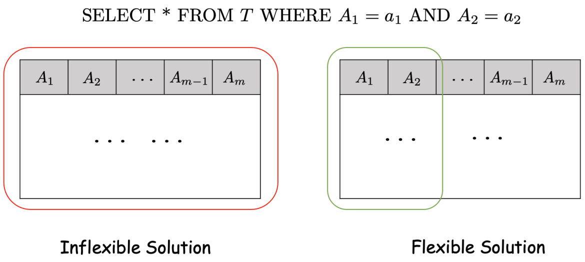


Fig. 1: The illustration of practicality enhancement. The inflexible solution conducts calculations on all the columns (inside the red rectangle), our flexible solution calculates only the filtered columns (inside the green rectangle).

Recall given the database consisting of multiple columns $\{A_1, A_2, ..., A_m\}$, the filter operation selects k of them denoted by $\{A_{i_1}, ..., A_{i_k}\}$ and defines k sets $\{s_1, s_2, ..., s_k\}$ to filter records, i.e., it is required that $x_{i_j} \in s_j$ for any record $\mathbf{x} = (x_1, ..., x_m)$. While it is natural to select the ciphertexts of columns $\{A_{i_1}, ..., A_{i_k}\}$ to determine if they satisfy the filter conditions, the SOTA work [47] cannot support such a **flexible** property. Instead, they require the calculation must be based on the whole ciphertexts of $\{A_1, ..., A_m\}$. That means the calculation of filter is independent of the filter column numbers, resulting in unacceptable performance overhead in practice. In the worst case, the client may filter only 1 column but needs to conduct the filter calculation on the whole m columns. Moreover, the filter set size (i.e., $|s_j|$) is initialized in encryption and they adopt polynomials with assigned roots to conduct the filter (which we will formally introduce in Section IV-C). This requests the ciphertext of a record in a single column to consist of $t + 1$ group elements, incurring a higher calculation complexity and overhead. Suppose each column is set to allow the filter size to be at most t, the computational complexity is $O(m \cdot (t + 1))$ even if the client filters on only a single column. To that end, we enhance the practicality of filter on multi-dimensional data via allowing the flexibility in our new constructions. In detail, we require the calculation is needed only on the filtered k columns. In this way, our constructions are much more efficient when filtering only a subset of the m columns. We illustrate the practicality enhancement in Figure 1.

Besides, another important property for practicality is to *elastically* allow dynamic databases where new columns may be added or old columns should be deleted. The prior work [47] does not enable such a property such that it has to re-encrypt all columns again to pad new columns. This disadvantage results from the limitation of the existing FE scheme [32] that the prior work is based on. Therefore, we design totally new FE schemes to allow the elasticity, further enhancing the practicality. Interestingly, our schemes are specialized to the multi-dimensional operations, making them faster *in the client side* than that in [47] even when filtering all the m columns, which is another contribution of this work.

B. Basic Filter

We first introduce the FE scheme for basic filter operation where given the filter set $\{s_1, ..., s_k\}$, it holds that $|s_1| = |s_2| = \cdots = |s_k| = 1$. Based on this core construction, we will extend it to address more complicated filters in Section IV-C and other operations in Section V.

[2] A simulator state

a) Hash values: In this basic filter, the client wonders if a record (vector) $\mathbf{x} \in \mathbb{Z}_q^m$ has required values on columns $\{A_{i_1}, ..., A_{i_k}\}$ denoted by $\{a_{i_1}, ..., a_{i_k}\}$. To achieve that, we let the client set another vector $\mathbf{y} \in \mathbb{Z}_q^m$ such that $\forall j \in [m]$,

$$y_j = \begin{cases} a_j, & j \in \{i_1, ..., i_k\} \\ 0, & j \notin \{i_1, ..., i_k\} \end{cases}$$

In this way, it holds that $\langle \mathbf{x}, \mathbf{y} \rangle = \langle \mathbf{y}, \mathbf{y} \rangle$ if $\mathbf{x}$ satisfies all equality conditions. This provides a feasible solution for filtering but leaves *false positive (FP) rate*: the inner product equation may still hold although $\mathbf{x}$ does not follow the filter conditions, e.g., $\mathbf{y} = \langle 2, 1 \rangle$ and $\mathbf{x} = \langle 0, 5 \rangle$. To guarantee a negligible FP rate, we apply a hash function $\mathsf{H} : \{0,1\}^\lambda \times \{0,1\}^{\lceil \log q \rceil} \times \{0,1\}^{\lceil \log m \rceil} \to \mathbb{Z}_q$. That means, for any vector $\mathbf{x} = \{x_1, ..., x_m\}$, instead of calculating inner product using $\mathbf{x}$ directly, we use a new vector $\mathbf{x}'$ where

$$\forall i \in [m], x'_i = \mathsf{H}(hsk, x_i, i).$$

where hsk is the secret key for hash. The vector $\mathbf{y}$ is processed similarly to get a new vector $\mathbf{y}'$. Components in $\mathbf{y}'$ are set as zero besides: $\forall j \in \{i_1, ..., i)k\}$, $y'_j = \mathsf{H}(hsk, y_j, j)$. In this way, we can bound the FP rate with the following theorem:

Theorem 1. *Suppose* $\mathbf{z_1}, \mathbf{z_2}$ *are two random vectors in* $\mathbb{Z}_q^t$ ($t \in \mathbb{N}^+$)*, then it holds that* $\Pr[\langle \mathbf{z_1}, \mathbf{z_2} \rangle = \langle \mathbf{z_2}, \mathbf{z_2} \rangle] < 2/q$*.*

Proof. To calculate the probability of two random vectors in $\mathbb{Z}_q^t$ being orthogonal, we use f_j to denote the inner product of two random vectors in $\mathbb{Z}_q^j$. As any vector with length $(j+1)$ can be decomposed to two vectors with length j and 1, respectively. So we have $\forall j \geq 0$,

$$\Pr[f_{j+1} = 0] = \Pr[f_j = 0] \cdot \Pr[f_1 = 0] + \Pr[f_j \neq 0] \cdot \frac{q-1}{q} \cdot \frac{1}{q}$$

where $\Pr[f_1 = 0] = 2/q - 1/q^2 < 2/q$. Thus we have the inequality:

$$\Pr[f_{i+1} = 0] < \Pr[f_j = 0] \cdot \frac{2}{q} + \Pr[f_j \neq 0] \cdot \frac{1}{q} < \frac{2}{q}$$

which concludes our proof. □

Note $\langle \mathbf{x}', \mathbf{y}' \rangle = \langle \mathbf{y}', \mathbf{y}' \rangle$ is equivalent to $\langle \mathbf{x}' - \mathbf{y}', \mathbf{y}' \rangle = 0$ and $\mathbf{y}'$ has at most k non-zero values in positions $\{i_1, ..., i_k\}$. Suppose $\mathbf{x}'$ and $\mathbf{y}'$ are different on t ($0 < t < k$) components in these positions, denote the corresponding positions as $S_{differ} \subset \{i_1, ..., i_k\}$. Then FP happens only when the inner product of the two random vectors in $\mathbb{Z}_q^t$ produced by $\mathbf{x}' - \mathbf{y}'$ and $\mathbf{y}'$ on positions S_{differ} is zero, which happens with a probability smaller than $2/q$. Since the value of q is always exponential in the security parameter λ, the FP rate actually is negligible in λ.

Remark. It is more common and natural [32, 45, 49] to check if $\mathbf{x} = \mathbf{y}$ via calculating if $\|\mathbf{x} - \mathbf{y}\|_2 = 0$, i.e., if $\langle \mathbf{x} - \mathbf{y}, \mathbf{x} - \mathbf{y} \rangle = 0$. This enables no FP rate (perfect correctness) but requires complex and expensive costs because it is required to calculate $\|\mathbf{x}\|^2 + \|\mathbf{y}\|^2 - 2\langle \mathbf{x}, \mathbf{y} \rangle$ as shown in prior works [45, 49]. Therefore, we present applying the hash function for a simpler and more efficient check while preserving the FP rate negligible.

b) Construction: Now we describe our construction in detail. As defined in Section III, there are four algorithms in our new FE scheme, namely $\Pi = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$. In particular, for any record $\mathbf{x}$ in the database, the client encrypts it with $\mathsf{Encrypt}$ to upload ciphertexts to the server. When there is a multi-dimensional query issued, the client converts it to a vector $\mathbf{y}$ and then processes it with KeyGen, which produces a token for filtering required records. Now we introduce them below. The database columns and dimensions of $\mathbf{x}$ and $\mathbf{y}$ in Π are defaulted as $m \in \mathbb{N}^+$.

1) $\Pi.\mathsf{Setup}(1^\lambda, 1^m)$: (Client, upload phase)
This algorithm takes an encoding of the security parameter λ and an encoding of an integer m as inputs and produces an asymmetric bilinear group $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, e)$. It selects two generators $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$. Importantly, it samples a vector $\mathbf{r} \in \mathbb{Z}_q^m$ and secret key hsk for the hash function H. Finally, it defines $pp := (\mathbb{G}_1, \mathbb{G}_2.\mathbb{G}_T, q, e)$ and the master secret key $msk := (pp, g_1, g_2, \mathbf{r}, hsk)$.
2) $\Pi.\mathsf{Encrypt}(msk, \mathbf{x})$: (Client, upload phase)
The encryption algorithm can be completed with two simple steps and thus is more efficient than prior similar works [32, 47]. Firstly, the client converts $\mathbf{x}$ to $\mathbf{x}'$ as introduced earlier. It also samples a number $\alpha \in \mathbb{Z}_q$. Then the client converts $\mathbf{x}'$ to ciphertexts by calculating $g_1^{\alpha(\mathbf{x}'+\mathbf{r})}$, i.e.,
$$(x'_1, ..., x'_m) \to (g_1^{\alpha(x'_1+r_1)}, ..., g_1^{\alpha(x'_m+r_m)}).$$
Finally, the client outputs $\mathsf{ct}_\mathbf{x} := (g_1^{\alpha(\mathbf{x}'+\mathbf{r})}, g_1^\alpha)$.
3) $\Pi.\mathsf{KeyGen}(msk, \mathbf{y})$: (Client, query phase)
This algorithm is as simple and efficient as the encryption one. Firstly, the client coverts $\mathbf{y}$ to $\mathbf{y}'$ using the hash function H. It also samples a number $\beta \in \mathbb{Z}_q$. Then it convert the vector $\mathbf{y}'$ to $\mathsf{sk}_{\mathbf{y}'}$ by calculating $g_2^{\beta\mathbf{y}'}$, i.e.,
$$(y'_1, ..., y'_m) \to (g_2^{\beta y'_1}, ..., g_2^{\beta y'_m}).$$
Additionally, this algorithm calculates
$$u := g_2^{\beta \sum_{i=1}^m r_i y'_i + \beta \sum_{i=1}^m (y'_i)^2}.$$
Finally, the client outputs $\mathsf{sk}_\mathbf{y} := (g_2^{\beta\mathbf{y}'}, u^{-1})$.
4) $\Pi.\mathsf{Decrypt}(\mathsf{ct}_\mathbf{x}, \mathsf{sk}_\mathbf{y})$: (Server, query phase)
The encryption algorithm can be completed with only $(m+1)$ pairing operations while the prior work [47] requires at least $2m$ pairing operations. Specifically, the **server** calculates:
$$\begin{aligned} & e(\mathsf{ct}_\mathbf{x}, \mathsf{sk}_\mathbf{y}) \\ = & e(g_1, g_2)^{\alpha\beta(\sum_{i=1}^{m}(x'_i y'_i + r_i y'_i)) - \alpha\beta(\sum_{i=1}^{m} r_i y'_i) - \alpha\beta \sum_{i=1}^{m} (y'_i)^2} \\ = & e(g_1, g_2)^{\alpha\beta\langle \mathbf{x}', \mathbf{y}' \rangle - \alpha\beta\langle \mathbf{y}', \mathbf{y}' \rangle}. \end{aligned}$$
If the result above is equal to $e(g_1, g_2)^0$, the algorithm outputs 1, implying that $\mathbf{x}$ satisfies the filter condition. Otherwise, the algorithm outputs 0, filtering out this $\mathbf{x}$.

c) Flexibility and elasticity: Now we introduce how to filter records with calculation on only filtered columns where the filter columns are different from the m columns set in advance.

- *Filtering subset.* To filter a column subset $\{A_{i_1}, ..., A_{i_k}\}$, given the ciphertext for $\mathbf{x}$ and key for $\mathbf{y}$ denoted by $(\mathsf{ct}_{\mathbf{x}}, \mathsf{sk}_{\mathbf{y}})$. Now we let the server to extract the component with index $\{i_1, ..., i_k\}$, i.e.,

  $$\mathsf{ct}'_{\mathbf{x}} := (g_1^{\alpha(x'_{i_1}+r_{i_1})}, ..., g_1^{\alpha(x_{i_k}+r_{i_k})}, g^{\alpha}, g^{\alpha}).$$

  The client also extracts similar components to get

  $$\mathsf{sk}'_{\mathbf{y}} := (g_2^{\beta y'_{i_1}}, ..., g_2^{\beta y_{i_k}}, (t')^{-1})$$

  where $t' := g_2^{\beta \sum_{j=1}^{j=k} r_{i_j} y'_{i_j} + \sum_{j=1}^{k} (y'_{i_j})^2}$.
- *Adding a column.* Adding a new column is more natural. The client samples a new random number $r_{m+1} \in \mathbb{Z}_q$ and resets $\mathbf{r} = (r_1, ..., r_m, r_{m+1})$. For any vector $\mathbf{x}$ and its ciphertext $\mathsf{ct}_{\mathbf{x}}$, the client extends $\mathbf{x} = (x_1, ...x_m, x_{m+1})$ and $\mathsf{ct}_{\mathbf{x}} = (g_1^{\alpha(x'_1+r_1)}, ..., g_1^{\alpha(x'_m+r_m)}, g_1^{\alpha(x_{m+1}+r_{m+1})})$ where x_{m+1} is the corresponding value under the new added column and $x'_{m+1} := \mathsf{H}(hsk, x_{m+1}, m+1)$. Then the filter can be done identically to our construction but now $(m+1)$ columns are allowed.

C. Extended Filter

Now we introduce how to apply polynomial with assigned roots in [47] to filter columns when the filter set includes multiple values. Formally, given the column A_i and the filter set $s_i := \{s_i^0, ..., s_i^t\}$ where $t > 1$, how to determine if $x_i \in s_i$. Shafieinejad et al. [47] define the polynomial

$$f(x) = (x - s_i^0)(x - s_i^1) \cdots (x - s_i^t) = \sum_{j=0}^{t} c_j x^j.$$

It is obvious that $f(x_i) = 0$ if $x_i \in s_i$. Differ from [47] which can calculate $f(x)$ using the inner product of $(x^0, x^1, ..., x^t)$ and $(c_0, c_1, ..., c_t)$, we request there exists no zero in the coefficient vector for security. So we decompose:

$$\sum_{j=0}^{t} c_j x^j = \sum_{j=0}^{t} (c_j^1 + c_j^2) x^j$$

where $c_j \equiv c_j^1 + c_j^2 \mod q$ and $c_j^1, c_j^2 \neq 0$. That means we separately calculate the inner product of $(x^0, x^1, ..., x^t)$ and $(c_0^h, c_1^h ..., c_t^h)$ $(h \in \{1, 2\})$ and add them together for $f(x)$. Using this polynomial decompose, we describe our extended FE scheme for extended filter as below. We simplicity the description for ease of presentation, please refer to the basic FE scheme for details.

1) $\Pi.\mathsf{Setup}(1^\lambda, 1^m)$: (Client, upload phase)
   It produces the bilinear group $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, e)$ and two generators $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$. It samples a random number $\delta \in \mathbb{Z}_q$, two vectors $\mathbf{r^1}, \mathbf{r^2} \in \mathbb{Z}_q^{(t+1)\cdot m}$, two vectors $\mathbf{b^1}, \mathbf{b^2} \in (\mathbb{Z}_q/\{0\})^{(t+1)\cdot m}$ and secret key hsk for the hash function H. Finally, it defines $pp := (\mathbb{G}_1, \mathbb{G}_2 . \mathbb{G}_T, q, e)$ and the master secret key $msk := (pp, g_1, g_2, \delta, \mathbf{r^1}, \mathbf{r^2}, \mathbf{b^1}, \mathbf{b^2}, hsk)$.
2) $\Pi.\mathsf{Encrypt}(msk, \mathbf{x})$: (Client, upload phase)
   Firstly, it converts $\mathbf{x}$ to $\mathbf{x}'$ via H. It also samples a number $\alpha \in \mathbb{Z}_q$. Then it converts $\mathbf{x}'$ to two groups of ciphertexts by calculating

   $$\mathsf{ct}_{\mathbf{x}}^1 := (g_1^{\alpha b_1^1[(x'_1)^0 + r_1^1]}, ..., g_1^{\alpha b_{t+1}^1[(x'_1)^t + r_{t+1}^1]}, ..., g_1^{\alpha b_{(t+1)m}^1[(x'_m)^t + r_{(t+1)\cdot m}^1]}),$$

   $$\mathsf{ct}_{\mathbf{x}}^2 := (g_1^{\alpha b_1^2[(x'_1)^0 + r_1^2]}, ..., g_1^{\alpha b_{t+1}^2[(x'_1)^t + r_{t+1}^2]} ..., g_1^{\alpha b_{(t+1)m}^2[(x'_m)^t + r_{(t+1)\cdot m}^2]}).$$

   Finally, it outputs $\mathsf{ct}_{\mathbf{x}} := (\mathsf{ct}_{\mathbf{x}}^1, \mathsf{ct}_{\mathbf{x}}^2, g^{\alpha\delta})$.
3) $\Pi.\mathsf{KeyGen}(msk, \mathbf{y})$: (Client, query phase)
   It first calculates the filter polynomial f_i for column A_i, the final polynomial coefficient vector is denoted by $\mathbf{c} \in \mathbb{Z}_q^{(t+1)\cdot m}$. It samples a random number $\beta \in \mathbb{Z}_1$ and generates two vectors $\mathbf{c}^1, \mathbf{c}^2$ including no zero such that $\mathbf{c} \equiv \mathbf{c^1} + \mathbf{c^2} \mod q$. Then it calculates

   $$\mathsf{sk}_{\mathbf{y}}^1 := (g_2^{\beta (b_1^1)^{-1} c_1^1}, ..., g_2^{\beta (b_2^1)^{-1} c_{t+1}^1}, ..., g_2^{\beta (b_{(t+1)\cdot m}^1)^{-1} c_{(t+1)m}^1}),$$

   $$\mathsf{sk}_{\mathbf{y}}^2 := (g_2^{\beta (b_1^2)^{-1} c_1^2}, ..., g_2^{\beta (b_2^2)^{-1} c_{t+1}^2}, ..., g_2^{\beta (b_{(t+1)\cdot m}^2)^{-1} c_{(t+1)m}^2}),$$

   Additionally, it calculates

   $$u := g_2^{\delta^{-1}\beta \sum_{i=1}^{(t+1)\cdot m} r_i^1 c_i^1 + \delta^{-1}\beta \sum_{i=1}^{(t+1)\cdot m} r_i^2 c_i^2}.$$

   Finally, the client outputs $\mathsf{sk}_{\mathbf{y}} := (\mathsf{sk}_{\mathbf{y}}^1, \mathsf{sk}_{\mathbf{y}}^2, u^{-1})$.
4) $\Pi.\mathsf{Decrypt}(\mathsf{ct}_{\mathbf{x}}, \mathsf{sk}_{\mathbf{y}})$: (Server, query phase)
   It calculates

   $$\begin{aligned} & e(\mathsf{ct}_{\mathbf{x}}, \mathsf{sk}_{\mathbf{y}}) \\ = & e(\mathsf{ct}_{\mathbf{x}}^1, \mathsf{sk}_{\mathbf{y}}^1) \cdot e(\mathsf{ct}_{\mathbf{x}}^2, \mathsf{sk}_{\mathbf{y}}^2) \cdot e(g_1, g_2)^{-\alpha\beta \sum_{i=1}^{(t+1)\cdot m} (r_i^1 c_i^1 + r_i^2 c_i^2)} \\ = & e(g_1, g_2)^{\sum_{i=1}^{m} f_i(x_i)}. \end{aligned}$$

   $\mathbf{x}$ satisfies the filter conditions if the result above is $e(g_1, g_2)^0$. Otherwise, $\mathbf{x}$ does not.

V. Extensive Multi-dimensional Operation

In this section, we extend our schemes to support other typical operations in EDB. The first operation is the equi-join [47] and the second operation is the equality test between columns and on column aggregations [53]. The extensions do not require much modifications to the basic schemes for multi-dimensional filter in Section IV, making them easy for the client to deploy.

A. Equi-join between Tables

Given two tables (T_A, T_B) with m_A and m_B columns separately, the usual equi-join without no filter can be trivially completed with deterministic encryption [2] but leaks all plaintext frequency and incurs dangerous *frequency-analysis* attacks [23, 40]. So we follow [26, 47] to address the more complex equi-join (with filter) with fine granular security, which are concerned about queries like

$$\text{SELECT * FROM } T_A \text{ JOIN } T_B \text{ ON } A_1 = B_1 \text{ WHERE}$$
$$(A_{i_1^A}, ..., A_{i_{j_A}^A}) \in s_1^A \times \cdots \times s_{j_A}^A \text{ AND}$$
$$(B_{i_1^B}, ..., B_{i_{j_B}^B}) \in s_1^B \times \cdots \times s_{j_B}^B$$

where $\{i_1^A, ..., i_{j_A}^A\} \subset \{2, ..., m_A\}$ and $\{i_1^B, ..., i_{j_B}^B\} \subset \{2, ..., m_B\}$. Such queries are expected to leak only *partial* plaintext frequency about records selected by the filter conditions and thus guarantees a more promising security level. However, even the SOTA work [47] cannot process such queries practically. It does not only is limited by the inflexibility of filter operations (cf. Section IV-A), but also cannot allow multiple target columns for join. In detail, the client in [47] adopts a suite of encryption forms to encryption the database and such a suite can *fit only one equation* between columns for join like $A_1 = B_1$. If the client hopes to enable both $A_1 = B_2$ and $A_2 = B_2$ for join in the future, then it has to prepares two suits of encryption for the two equations. That means, the storage and calculation overheads increases linearly to the number of join equations requested. In this paper, we first achieve the same functionality as [47] in equi-join, then we further improve our construction to support more join equations with better practicality.

a) Construction with fixed equation: Now we briefly describe the specific construction for equi-join requiring $A_1 = B_1$ based on the basic FE scheme in IV-B. The similar modifications can be naturally extended to other join equation and the FE scheme IV-C to filter multiple values in one column. We refer to the full version for detailed description.

1) $\Pi.\mathsf{Setup}(1^\lambda, 1^m)$: (Client, upload phase)
   It produces the bilinear group $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, e)$ and two generators $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$. It samples a random vector $\mathbf{r} \in \mathbb{Z}_q^m$ and secret key msk for the hash function H. Finally, it defines $pp := (\mathbb{G}_1, \mathbb{G}_2.\mathbb{G}_T, q, e)$ and the master secret key $msk := (pp, g_1, g_2, \mathbf{r}, hsk)$.
2) $\Pi.\mathsf{Encrypt}(msk, \mathbf{x})$: (Client, upload phase)
   Firstly, it converts $\mathbf{x}$ to $\mathbf{x}'$ via H. It also samples a number $\alpha \in \mathbb{Z}_q$. Then it converts $\mathbf{x}'$ to ciphertext:
   $$(x'_1, ..., x'_m) \to (g_1^{x'_1 + \alpha r_1}, g_1^{\alpha(x'_2 + r_2)}, ..., g_1^{\alpha(x'_m + r_m)}).$$
   Finally, it outputs the above ciphertexts with a padding g_1^α as $\mathsf{ct}_\mathbf{x}$.
3) $\Pi.\mathsf{KeyGen}(msk, \mathbf{y})$: (Client, query phase)
   It first converts the filter vector $\mathbf{y}$ to $\mathbf{y}'$. Note (y_1, y'_1) are None as column A_1 is for join instead of filter. Then it samples two numbers $\beta, \gamma \in \mathbb{Z}_q$ and calculates:
   $$(y'_1, ..., y'_m) \to (g_2^\gamma, g_2^{\beta y'_2}, ..., g_2^{\beta y'_m}).$$
   Finally, it outputs the above the above results with a padding u^{-1} as $\mathsf{sk}_\mathbf{y}$ where
   $$u := g_2^{\beta \sum_{i=2}^m r_i y'_i + \beta \sum_{i=2}^m (y'_i)^2 + \gamma r_1}.$$
4) $\Pi.\mathsf{Decrypt}(\mathsf{ct}_\mathbf{x}, \mathsf{sk}_\mathbf{y})$: (Server, query phase)
   It calculates
   $$\begin{aligned} & e(\mathsf{ct}_\mathbf{x}, \mathsf{sk}_\mathbf{y}) \\ = & e(g_1, g_2)^{\alpha\beta \sum\limits_{i=2}^{m} [x'_i y'_i + r_i y'_i - r_i y'_i - (y'_i)^2] + \gamma x'_1} \end{aligned}$$
   If $\mathbf{x}$ satisfies the filter conditions on columns $(A_2, ..., A_m)$, then the above process outputs $e(g_1, g_2)^{\gamma x'_1}$, i.e., deterministic encryption.

We further explain the equi-join: the client encrypts records in two tables with the same Setup and $\mathsf{Encrypt}$. Given an equi-join on $A_1 = B_1$ with specified filter on other columns, the client generates the vector $\mathbf{y}$ corresponding this query and processes it with keyGen to upload $\mathsf{sk}_\mathbf{y}$ where the randomness includes the number $\gamma \in \mathbb{Z}_q$. Then give two records $\mathbf{x^1} \in T_A$ and $\mathbf{x^2} \in T_B$ satisfying the filter condition, the server gets

$$e(g_1, g_2)^{\gamma (x_1^1)'} = e(\mathsf{ct}_{\mathbf{x^1}}, \mathsf{sk}_\mathbf{y}),$$
$$e(g_1, g_2)^{\gamma (x_1^2)'} = e(\mathsf{ct}_{\mathbf{x^2}}, \mathsf{sk}_\mathbf{y}).$$

And the server knows the two records should be concatenated if $e(g_1, g_2)^{\gamma (x_1^1)'} = e(g_1, g_2)^{\gamma (x_1^2)'}$ because it implies $x_1^1 = x_1^2$ and the FP rate is negligible in λ [47].

b) Variable equations: The significant advantage of the scheme above is that it can be simply extended to more general equations for equi-join like $A_i = B_j$ ($i \in [m_A]$, $j \in [m_B]$). To do that, the client adopt different KeyGen to the two tables but use the same γ in KeyGen. In this way, the client uploads different queries and thus different $\mathsf{sk}_\mathbf{y}$ to the two tables to filter records but finally outputs the results with the same γ after $\mathsf{Decrypt}$, achieving the equality check. Moreover, the client only needs to store two suites of ciphertexts to enable both filter and equi-join, i.e., for record $\mathbf{x}$, it stores two vectors

$$\mathsf{ct}_\mathbf{x}^1 = (g_1^{\alpha(x'_1 + r_1^1)}, g_1^{\alpha(x'_2 + r_2^1)}, ..., g_1^{\alpha(x'_m + r_m^2)}),$$
$$\mathsf{ct}_\mathbf{x}^2 = (g_1^{x'_1 + \alpha r_1^2}, g_1^{x'_2 + \alpha r_2^2}, ..., g_1^{x'_m + \alpha r_m^2}).$$

where there are two random vectors $\mathbf{r^1}, \mathbf{r^2} \in \mathbb{Z}_q^m$ generated by the client in Setup as a part of msk. Then if the client requests equi-join for column i, it organizes the new $\mathsf{ct}_\mathbf{x}$ by extracting $g_1^{x'_i + \alpha r_i^2}$ from $\mathsf{ct}_\mathbf{x}^2$ and other components from $\mathsf{ct}_\mathbf{x}^1$. After that, the client can issue queries as usual to conduct equi-join. Compared with the scheme with fixed equation, the total overhead of such a complete and variable scheme is at most 2 times storage overhead in client and server.

B. Equality Test

In this section, we consider more complex filter operations which select records according to the relations between columns. Formally, these queries can be generalized to the following form:

$$\text{SELECT * FROM T WHERE } \sum_{j=1}^{k} w_j A_{i_j} = p.$$

where $\{A_{i_1}, ..., A_{i_k}\}$ are columns the client filters on. We remark this form can be regarded as a generalization of two common queries in databases:

- *Inter-column equality:* The condition of $A_{i_1} = A_{i_2}$ can be considered as $\sum_{j=1}^{2} w_j A_{i_j} = 0$ where $(w_1, w_2) = (1, -1)$.
- *Signed aggregation:* It is clear that the form we aim to covers aggregation with signs. A typical example is condition $\sum_{j=1}^{k} A_{i_j} = p$ where the client sums some columns to test equality on a special value p.

We distinguish the schemes for equi-join and equality test between columns: the former allows equality check between items in different rows while the latter only reveal if items under different columns but in the same row are equal.

a) Construction: Before we introduce the construction, we identify that given a record $\mathbf{x} = (x_1, x_2, ..., x_m)$, the equality check can be completed via the inner product of $(\mathbf{x}, 1)$ and $(\mathbf{y}, -p)$ where $\mathbf{y} = (w_1, w_2, ..., w_m)$. The essence is how to determine if the inner product is zero without revealing other sensitive information. With the default $\mathbf{x}$ and $\mathbf{y}$, now we describe the detailed FE scheme below.

1) $\Pi.\mathsf{Setup}(1^\lambda)$: (Client, upload phase)
It produces the bilinear group $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, e)$ and selects two generators $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$. Importantly, it samples two vectors $\mathbf{r}, \mathbf{b} \in \mathbb{Z}_q^m$ and a random number $\delta \in \mathbb{Z}_q$. Finally, it defines $pp := (\mathbb{G}_1, \mathbb{G}_2.\mathbb{G}_T, q, e)$ and the master secret key $msk := (pp, g_1, g_2, \mathbf{r}, \mathbf{b}, \delta)$.
2) $\Pi.\mathsf{Encrypt}(msk, \mathbf{x})$: (Client, upload phase)
Firstly, it samples a number $\alpha \in \mathbb{Z}_q$. Then it converts $\mathbf{x}$ to ciphertexts by calculating:
$$(x_1, ..., x_m) \rightarrow (g_1^{\alpha b_1 (x_1 + r_1)}, ..., g_1^{\alpha b_{m-1}(x_m + r_m)}).$$
Finally, the client outputs the vector above with padding $g_1^{\alpha\delta}$ as $\mathsf{ct}_{\mathbf{x}}$.
3) $\Pi.\mathsf{KeyGen}(msk, \mathbf{y})$: (Client, query phase)
Firstly, it samples a number $\beta \in \mathbb{Z}_q$. Then it convert the vector $\mathbf{y}$ to $\mathsf{sk}_{\mathbf{y}}$ by calculating:
$$(y_1, ..., y_m) \rightarrow (g_2^{\beta b_1^{-1} y_1}, ..., g_2^{\beta b_m^{-1} y_m}).$$
Additionally, this algorithm calculates
$$t := g_2^{\beta \delta^{-1}(p + \sum_{i=1}^{m} r_i y_i)}.$$
Finally, the client outputs the vector above with padding t^{-1} as $\mathsf{sk}_{\mathbf{y}}$.
4) $\Pi.\mathsf{Decrypt}(\mathsf{ct}_{\mathbf{x}}, \mathsf{sk}_{\mathbf{y}})$: (Client, query phase)
It calculate $e(\mathsf{ct}_{\mathbf{x}}, \mathsf{sk}_{\mathbf{y}})$, i.e.,
$$e(g_1, g_2)^{\alpha\beta[\sum\limits_{i=1}^{m}(x_i y_i + r_i y_i - r_i y_i) - p]} = e(g_1, g_2)^{\alpha\beta\langle \mathbf{x}, \mathbf{y}\rangle - \alpha\beta p}.$$
This algorithm outputs 1 if the result above is equal to $e(g_1, g_2)^0$. Otherwise, it outputs 0.

VI. Security Analysis

In this section, we prove the security of all operations we support in this work. Here we only give the theorem and (informal) proof corresponding to the core and basic multi-dimensional filter for space. We refer the readers to the full version [54] for a complete analysis of each operation.

Theorem 2. *Our FE scheme for multi-dimensional filter in Section IV-B is SIM-secure in the generic group model, i.e., only the results of* Filter *are revealed to the untrusted server.*

a) Proof sketch: Suppose the adversary makes in total Q query calls to the encryption oracle and key generation oracle. Intuitively, the simulator should only get input of a list that records whether the input $\mathbf{x}^i$ to the i-th query call of the encryption oracle matches the input $\mathbf{y}^j$ to the j-th query call of the key generation oracle, where $i, j \in [Q]$. The simulator is then responsible for handling all types of queries that the adversary may submit in the real world, including encryption, key generation, and bilinear group operation queries. Internally, the simulator maintains a table mapping handles to formal polynomials formed through these queries. The main challenge lies in responding to zero-test queries. This is straightforward because if we remove the zero-test oracle from both the real and simulated games, they become identical. Without the ability to query zero-test, in both games, the adversary always receives randomly sampled handles and is never able to establish any relationship between these handles; hence, it cannot distinguish which game it is playing.

Therefore, we will primarily focus on explaining how the simulator respond to zero-test queries, so that its responses cannot be distinguished from real world query answers. Upon receiving zero-test queries, the simulator decomposes the input polynomial into "honest" and "dishonest" components. An "honest" query corresponds to the exact evaluation of $C \cdot Dec(\mathbf{x}^i, \mathbf{y}^j)$, for $i, j \in [Q]$ and $C \in \mathbb{N}$. If the input has only an "honest" component, the simulator responds the query by referencing its input list that maintains the information about the equivalence between $\mathbf{x}^i$ and $\mathbf{y}^j$. If the query contains "dishonest" component, the simulator always returns "nonzero". It is easy to see that the simulator is "correct" in the sense that it performs decryptions correctly. What remains to be shown is that, in the real world, the "dishonset" component can hardly evaluate to zero, meaning the simulator's answer cannot be distinguished from real world responses. The formal proof involves the construction of formal variables and the specification of the simulator, demonstrating that the ideal world simulator's responses accurately mimics the query answers in the real world.

b) Formal variables used in our proof: Let $Q = \mathsf{Poly}(\lambda)$ be the total number of queries made to the encryption and key generation oracles by the adversary $\mathcal{A}$. We define two non-disjoint sets of formal variables:

$$\mathcal{R} = \left\{\gamma_j^i, \delta_j^i\right\}_{i\in[Q], j\in[m]} \cup \left\{\alpha^i, \theta^i\right\}_{i\in[Q]}$$

$$\mathcal{T} = \left\{\alpha^i, \beta^i\right\}_{i\in[Q]} \cup \left\{r_j\right\}_{j\in[m]} \cup \left\{x_j^i, y_j^i\right\}_{i\in[Q], j\in[m]}$$

Suppose $\mathbf{x}^i$ represents the input to the i-th Enc query and $\mathbf{y}^i$ represents the input to the i-th KeyGen query for some $i \in [Q]$, we let the formal variables defined above represent the following:

- γ_j^i: the value $\alpha^i \cdot (\mathbf{x}_j^i + \mathbf{r}_j)$
- δ_j^i: the value $\beta^i \cdot \mathbf{y}_j^i$
- θ^i: the value $\beta^i \cdot \langle \mathbf{y}^i + \mathbf{r}^i, \mathbf{y}^i \rangle$
- α^i: the random value α sampled in the i-th Enc query
- β^i: the random value β sampled in the i-th KeyGen query
- r^j: the j-th value in the vector $\mathbf{r}$
- x_j^i: the j-th value in the vector $\mathbf{x}^i$
- y_j^i: the j-th value in the vector $\mathbf{y}^i$

c) Details of the simulator: The simulator $\mathcal{S}$ initializes empty maps M_1, M_2, and M_T. It answers queries in the following way:

- Enc: On input a vector $\mathbf{x}^i \in \mathbb{Z}_q^m$, the simulator $\mathcal{S}$ receives as input a new map M_{eq} and updates its state. Then, $\mathcal{S}$ samples a fresh handle $h_\alpha \leftarrow\!\!\$ \{0,1\}^\lambda$ and saves the mapping $M_1[h] = -\alpha^i$. Next, for each $j \in [m]$, $\mathcal{S}$ samples a fresh handle $h_j \leftarrow\!\!\$ \{0,1\}^\lambda$ and saves $M_1[h_j] = \gamma_j^i$. Finally, $\mathcal{S}$ returns the a ciphertext $\mathsf{ct} = (h_1, \ldots, h_m, h)$.
- KeyGen: On input a vector $\mathbf{y}^i \in \mathbb{Z}_q^m$, the simulator $\mathcal{S}$ receives as input a new map M_{eq} and updates its state. Then, for each $j \in [m]$, $\mathcal{S}$ samples a fresh handle $h_j \leftarrow\!\!\$ \{0,1\}^\lambda$ and saves $M_2[h_j] = \delta_j^i$. It also samples a fresh handle $h \leftarrow\!\!\$ \{0,1\}^\lambda$ and saves $M_2[h] = \theta^i$. Finally, $\mathcal{S}$ returns with a ciphertext $\mathsf{sk} = (h_1, \ldots, h_m, h)$.
- Op: On input of two handles h_1 and h_2, $\mathcal{S}$ checks if there exist some map M_ℓ such that $h_1, h_2 \in M_\ell$, where $\ell \in [1, 2, T]$. If the check fails, $\mathcal{S}$ returns $\perp$ otherwise it samples a fresh handle $h \leftarrow\!\!\$ \{0,1\}^\lambda$ and adds the following to the map $M_\ell[h] = M_\ell[h_1] + M_\ell[h_2]$.
- Pair: On input of two handles h_1 and h_2, $\mathcal{S}$ checks if $h_1 \in M_1$ and $h_2 \in M_2$. If the checks fails $\mathcal{S}$ returns $\perp$ otherwise it samples a fresh handle $h \leftarrow\!\!\$ \{0,1\}^\lambda$ and adds it to the map $M_T[h] = M_1[h_1] \cdot M_2[h_2]$.
- ZT: On input of a handle h, $\mathcal{S}$ checks if $h \in M_\ell$, where $\ell \in [1, 2, T]$, the simulator then proceeds as follows:
  1) The simulator checks if $M_\ell[h]$ evaluates to identically zero.
  2) If $\ell \neq T$, outputs "non-zero".
  3) $\mathcal{S}$ decomposes $M_T[h]$ to the following polynomial:
  $$\sum_{i,j \in [Q]} c_{i,j} \cdot p_{i,j}\left(\alpha^i, \theta^j, \{\gamma_k^i, \delta_k^j\}_{k \in [m]}\right) + f(\mathcal{T})$$
  where $p_{i,j}$ is defined as:
  $$\sum_{k \in [m]} \gamma_k^i \cdot \delta_k^j - \alpha^i \cdot \theta^j$$
  4) If the polynomial f is not empty, $\mathcal{S}$ returns "non-zero".
  5) For each $p_{i,j}$, the simulator $\mathcal{S}$ looks up $M_T[(i,j)]$, if they are all True, then it returns "zero" and "non-zero" otherwise.

Proof. We provide a formal proof demonstrating that the aforementioned simulator is correct and it accurately responds to the zero-test query, matching the distribution observed in the real world game. We examine each step involved in how the simulator responds to the zero-test query. Let $p = M_\ell[h]$, where h is in handle adversary queries.

1) We first show that checking whether p is exactly a zero polynomial can be efficiently done. Recall that the adversary makes at most $Q = \mathsf{Poly}(\lambda)$ queries to the key generation, encryption, and group operation oracles, and it only receives handles corresponding to formal variables defined in $\mathcal{R}$ or $\mathcal{R} \times \mathcal{R}$. The simulator then replace variables in p with formal variables defined in set $\mathcal{T}$. We observe that the monomial of highest degree the polynomial p may contain is the term $\gamma^i \cdot \theta^j$, for $i, j \in [Q]$, which can be efficiently represented by variables in $\mathcal{T}$. It is clear then this polynomial is polynomially-sized and checking whether it is an identically zero polynomial is efficient to do. If it is, the simulator correctly outputs "zero".
2) We now explain why the simulator always output non-zero when ℓ is 1 or 2. Consider the case where $\ell = 1$, by construction, the only monomials the adversary obtains are responses to the encryption query. We can write the polynomial p as:
$$p = \sum_{i \in [Q]} \alpha^i \cdot \left(c_0 + \sum_{j \in [m]} c_j (r_j + x_j^i) \right),$$
where $c_0, \ldots, c_m$ are some constants. In the real game, because r_j is randomly sampled, then regardless to the choice of x_j^i, the term $r_j + x_j^i$ should not be identically zero with overwhelming probability. Then since α^i and r_j are distributed uniformly and independently over $\mathbb{Z}_q$ and the polynomial p has degree at max 2 in these monomials, we conclude by Schwartz-Zippel (Lemma 2.9) that p evaluates to non-zero in the real distribution with overwhelming probability. Correctness of the simulation then follows with overwhelming probability. Also note that the adversary is making at most $\mathsf{Poly}(\lambda)$ queries, so the accumulated probability of making many zero-test queries remains to be negligible.
Similarly when $\ell = 2$, we write the polynomial p as:
$$p = \sum_{i \in [Q]} \beta^i \cdot \left(c_0 \cdot \langle \mathbf{y}^i, \mathbf{y}^i + \mathbf{r} \rangle + \sum_{j \in [m]} c_j y_j^i \right),$$
where $c_0, \ldots, c_m$ are some constants. Recall that y_i^j are some outputs of a non-zero hash function and hence they can't be exactly zero regardless to the choice of input y. Then the idea follows similar to above and the polynomial has overwhelmingly probability to be non-zero.
3) Akin to step 1), the adversary can perform this step efficiently.
4) If the polynomial f is not empty, for $i, j \in [Q]$ and $k, l \in [m]$, f may contain the following type of monomials:
   - $\alpha^i \cdot \theta^j$, which is of order 4,
   - $\alpha^i \cdot \delta_k^j$, which is of order 3,

- $\gamma_k^i \cdot \theta^j$, which is of order 5,
- $\gamma_k^i \cdot \delta_k^j$, which is of order 4.

We begin by showing that when the polynomial f is non-empty, it cannot be the zero polynomial. To establish this, we decompose the monomials by their degrees and show that each is distinct. It is evident that monomials of differing degrees are distinct. Following this, we explicitly express the two monomials of degree 4.

$$\alpha^i \cdot \theta^j = \alpha^i \cdot \beta^j \cdot \langle \mathbf{y}^j, \mathbf{y}^j \rangle + \alpha^i \cdot \beta^j \cdot \langle \mathbf{y}^j, \mathbf{r} \rangle$$
$$\gamma_k^i \cdot \delta_k^j = \alpha^i \cdot (\mathbf{x}_k^i + \mathbf{r}_k) \cdot \beta^j \cdot \mathbf{y}_k^j$$

Clearly, these two monomials are different, as they involve different formal variables. Secondly, we demonstrate that in the real world, if f is non-empty, it is a non-zero polynomial with overwhelming probability. This is because the values involved are independent and uniformly distributed, as they contain different random variables. By again invoking the Schwartz-Zippel lemma, we can assert that the polynomial f is highly likely to be non-zero.

5) If the simulator has reached to this step, it means that then the adversary is asking for the correct evaluation of some $Dec(x_i, y_j)$. It is clear that if for each pair of (i, j), $x_i = y_j$, then the polynomial above should be evaluated to zero. Otherwise, we get random value

□

VII. Implementation

In this section, we implement our constructions for different operations and compare them with the SOTA work [47]. We remark that although [47] is originally designed for equi-join after filter over multiple columns, it can simply remove the equi-join to execute only filter as the comparison object of our schemes for filter. In all our experiments, we denote the scheme in [47] as FixSch as it fix the columns and operations before encryption. On the contrary, our schemes allow more flexible calculations and queries to enhance the practicality thus is denoted by FleSch.

A. Setup and Dataset

a) Setup: We compare methods with typical efficiency metrics in EDB including: 1) **processing time**; 2) **communication overhead**; 3) **server and client storage occupied**. The three metrics exactly describe the performance and evaluate the practicality of methods in production, thus are commonly concerned [12, 17, 43, 51]. We conduct all methods in two separate server and client to be consistent with practical scenarios and popular EDBs [14, 43, 44]. The server is equipped with a powerful Ubuntu 22.04.3 machine. It adopts Intel Xeon Platinum 8160 CPU (96 cores, 2.10 GHz), owns 376GB memory and sufficient storage space in hard disk. The client machine is limited on resources: it is played by Aliyun cloud machine (4 vCPUs from Intel Xeon Platinum 8269CY, 2.50GHz) with only 16GB memory. The bandwidth between the client and server machines is 100Mbps under WAN and the corresponding latency is 30ms. We implement our proposed schemes, as well as the equi-join scheme from [47], using C++17. For the bilinear pairing group, we utilize the Relic library[3] and select the BLS12-381 curve due to its provision of 128-bit security and support for efficient optimal ate pairing. The semantically secure ciphertexts were achieved using AES-CBC encryption from the OpenSSL package[4].

b) Dataset: We select a total of three large datasets to evaluate the method performance in practice. The first one is a synthetic dataset denoted by RND. It is generated by ourselves with rows from 2^{10} to 2^{24} and columns from 1 to 2^6 to test the method scalability. The item length under each column is set as 16 bytes to cover usual integers and strings. Please notice the method performance on calculation actually is identical if the item length is no more than 381 bits since the calculation is *data-independent* within $\mathbb{Z}_q$. So the client can enable longer item lengths without additional overhead in calculation. The second dataset denoted by TPC-H is following [47] to conduct filter and equi-join operations on TPC-H tables including two tables *Orders* $:=$ {*orderkey, custkey, orderstatus, totalprice, orderdate, orderpriority, clerk, shippriority, comment*} and *Customer* $:=$ {*custkey, name, address, nationkey, phone, acctbal, mktsegment, comment*}. There are 150,000 and 1,500,000 rows in the two tables, respectively. The third dataset is the credit card data taken from the UCI machine learning repository [5] and also adopted in database community [13]. There are 30,000 rows and 25 columns in this dataset, describing the banking information about credit cards. We use this dataset to test equality test operations and denote it as Credit.

B. Client-side Efficiency

Before we introduce the query efficiency, we should first clarify the overhead for initialization in the client side and encrypting the whole database, i.e., the cost of Setup and Encrypt in FE schemes. We list the theoretical overhead of the two methods in Table I and test the actual performance on RND in Figure 2. Both of them show the significant efficiency gap between FixSch and FleSch. This results from the expensive matrix calculation in FixSch. For Setup, it requires calculating the inverse of a matrix in $\mathbb{Z}_q^{m \times m}$, incurring the $O(m^3)$ complexity in calculation and $O(m^2)$ in storage. But our scheme does not need the matrix and guarantees linear complexity and storage. Thus as shown in Figure 2(a), the cost of FixSch increases much faster than that of FleSch in Setup with m increases. The time usage can be nearly 10^6 ms (17 mins) when $m = 200$ while ours needs only 104 ms, i.e., the speedup is **8886**×. Even for the Encrypt, our scheme is still comparative to FixSch when $m \leq 100$ and faster than it when $m > 100$ due to the matrix calculation in FixSch. It is predictable in very large databases when m is larger, the speedup will be more impressive.

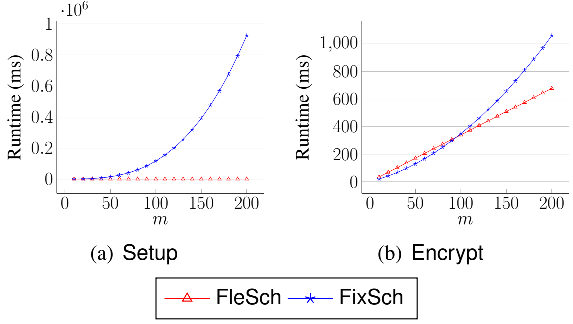
[3] https://github.com/relic-toolkit/relic

[4] https://github.com/openssl/openssl

[5] https://archive.ics.uci.edu/dataset/350/default+of+credit+card+clients

Methods	Setup	Encrypt	Client-side storage
FixSch	$O(m^3)$	$O(m^2)$	$O(m^2)$
FleSch	$O(m)$	$O(m)$	$O(m)$

TABLE I: Theoretical complexity.



(a) Setup

(b) Encrypt

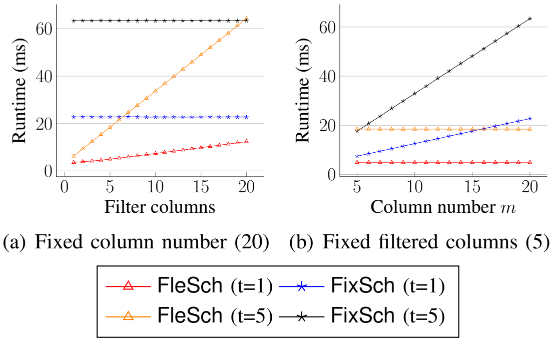
Fig. 2: Performance in client side ($t = 5$).

C. Filter Efficiency

The total cost of methods in filter consists of three parts: 1) client calculation cost (KeyGen), 2) server calculation cost (Decrypt), and 3) communication cost between client and server. We separately compare the two methods on the three components and finally evaluate the total cost.

a) Separate costs: We first evaluate the method performance on server calculation when filtering databases, i.e., the time usage of testing if a row satisfies the conditions required. We apply the RND dataset to show the performance under *different column number* m and different columns filtered on. The database is set to be 2^{10} to compute the average time on calculation for a row fairly. Note the calculation is directly and linearly traversing each record, the total calculation time on the whole database with different numbers of rows also can be estimated by the product of average time here and the corresponding row numbers. The experimental results are shown in Figure 3 where the performance varies from parameters including the column number m, the number of columns filtered on, and the polynomial degree t. It is shown under the same parameters, FleSch is always much faster than FixSch because of its flexibility introduced in Section IV-A. Especially, when the number of filtered columns is much smaller than m, FleSch can achieve a crucial speedup, e.g., when only 1 column out of 20 columns is filtered, our method has a speedup of **6.3**× and **10.1**× under $t = 1$ and $t = 5$, respectively. Moreover, Figure 3(b) shows that the flexibility allows our scheme to have a stronger dominance on FixSch under a larger m. Similarly, Figure 4(a) indicates the client-side calculation in our scheme is also linear to the number of filtered columns, performing much more efficiently than FixSch. Both the calculation in the client and server side of FixSch is always identical no matter how many columns from the m columns are filter because the matrix calculation requires that all the m components have to be used in the calculation process.

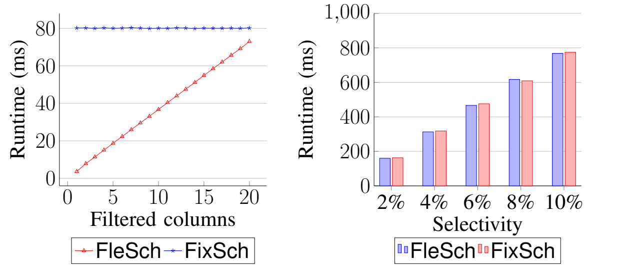
For communication overhead, as both the two schemes only return the semantically secure ciphertexts attached to FE ciphertexts, they perform nearly identically in communication cost for retrieving the same number of records. We evaluate them on the RND dataset with 10^4 rows and setting the selectivity (i.e., the proportion of records selected by filter) to be $\{2\%, 4\%, 6\%, 8\%, 10\%\}$. The time usage for communication is shown in Figure 4(b) and it is clear that the communication time increases linearly to the number of selected rows. But even when there are 10% rows (1000 rows) retrieved, the time cost is no more than 0.8 s, affecting the time usage of the whole filter little as the calculation per row can be at most 63 ms (calculation on only 15 rows costs over 0.8 s).



(a) Fixed column number (20)

(b) Fixed filtered columns (5)

Fig. 3: Average calculation cost per row for filter.



(a) Client-side calculation

(b) Communication time

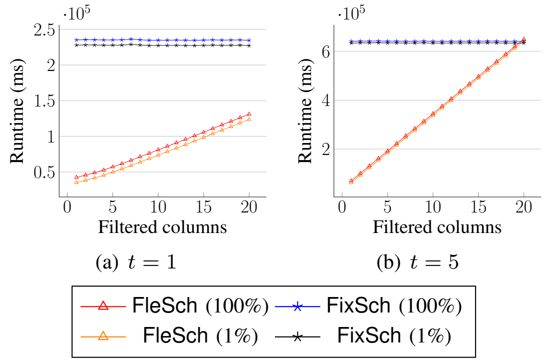
Fig. 4: Comparison on client-side calculation and communication overhead.

b) Total costs: Finally, we evaluate the total time usage of the whole filter operation. Recall our speedup aims at the calculation part, so a larger selectivity implies a smaller speedup to the whole operation. So we pick the extreme values $\{1\%, 100\%\}$ for selectivity to evaluate the upper and lower bound of the speedup. The two methods are conducted on RND with $10,000$ rows and 20 columns. The results in Figure 5 extensively show that our scheme achieves at most a 10× speedup compared to FixSch, e.g., when there are only 1 column filtered, $t = 5$, and selectivity is 1%, the speedup is the most and achieves 10.1×.

D. Extended Operations

As we introduced in Section V, our construction can be modified to support more operations including: 1) equi-join between tables and 2) equality test between columns and on column aggregations. We evaluate our scheme on these operations to further show its enhanced practicality.

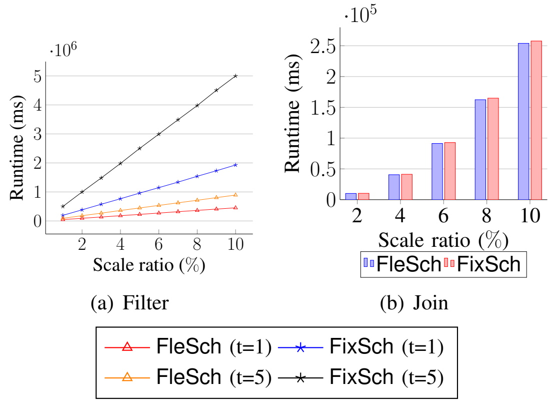
a) Equi-join: We compare our scheme FleSch with FixSch in equi-join on the dataset TPC-H. Figure 6 depicts their performance under different row numbers to show the



(a) $t = 1$

(b) $t = 5$

Fig. 5: Comparison on the total time of filter.



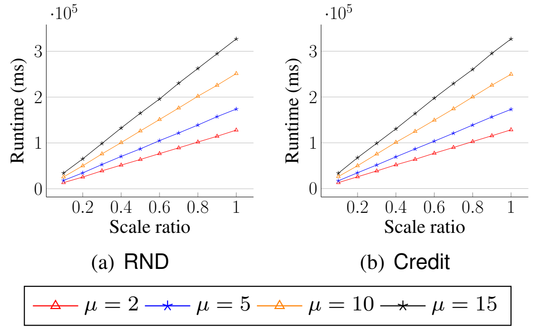
(a) Filter

(b) Join

Fig. 6: Cost of equi-join on TPC-H.

scalability. To adjust the row number, we use different scale ratios (like [47]) to extract rows from TPC-H for test, e.g., the scale ratio 1% implies 1% of rows in this dataset are used for the operation. The two methods complete equi-join by first filtering columns according to conditions and then using nested loop join to concatenate records satisfying the join equations. Our scheme differs from FixSch in the first step but is nearly identical to it in the second step. So we separately compare them in the two steps and show them in the subfigures. The total time usage of equi-join can be directly calculated by adding the two parts. It is shown both the two methods spend time linearly to the row numbers on filter. The average time usage per row is similar to that in Figure 3 because the schemes for equi-join and pure filter differ on only the encryption of a single column. Also, FleSch is around $2 \sim 5\times$ faster than FixSch in filter, and improves the efficiency of the whole equi-join operation by $45\% \sim 80\%$. As here the two schemes adopt the costly nested loop for join, the improvement proportion actually can be much higher with more efficient join methods like hash join [38].

b) Equality-test: As far as we know, our scheme is the only method that can test equality between columns and on column aggregations without fully homomorphic encryption (FHE) [21]. FHE approaches are still too expensive for production, e.g., the SOTA work called ArcEDB [56] needs nearly 100 ms for testing equality between two 64-bit integers. Moreover, there is still no open-sourced practical implementation of FHE-based EDBs which can complete both equality test



(a) RND

(b) Credit

Fig. 7: Cost of equality test on RND and Credit. μ denotes the number of columns aggregated.

and reveal the test results to retrieve records. Contrarily, our FE scheme can complete both of the tow tasks within only 3.6 ms. So we only evaluate our scheme for equality test and show results in Figure 7. We conduct the experiments on both RND and Credit to test the performance under different scale ratios (i.e., the number of rows extracted from the dataset for equality test), aggregated columns, and retrieved columns. There are only a small portion of rows retrieved in the sparse Credit dataset, which describes the performance when only a few records are transferred. As a comparison object, we generated items in RND with high frequency to simulate scenarios where nearly all records are retrieved. The two datasets capture complex real-world scenarios and client's queries. Interestingly, the subfigures show that the performance on them is very close under the same number of aggregated columns and row numbers. This result from the communication cost only occupies a small proportion of the whole operation. We observe the total time cost increases linearly to both the aggregated columns and number of rows. In the worst case we tested where 100% of $10,000$ records are retrieved and 15 columns ($\mu = 15$ in Figure 7) are aggregated for equality test, our scheme requires only 5.4 mins.

E. Storage Overhead

In the end, we discuss the storage overhead required by adapting the proposed scheme. Within the Relic library, G1 elements are represented by 152 bytes, and field elements and G2 elements by 296 bytes. Taking parameters used in previous experiments as an example, when $m = 20$ and $t = 1$, FleSch requires 6.2 KB storage for the master secret key, while FixSch requires 473.6 KB storage. Their ciphertext sizes are 3.2 KB and 6.1 KB, respectively. When $t = 5$, the difference in msk sizes becomes more pronounced. FleSch requires only 71 KB of storage, whereas the FixSch requires 8.6 MB, which is roughly $120\times$ larger. In this scenario, FleSch has a larger ciphertext size of 36.5 KB because we need to double the size for the polynomial coefficients, and ciphertext size for FixSch is 18.3 KB. This server-side storage increase enables greater flexibility for FleSch and permits much faster join operations when filtering only a portion of the columns.

VIII. Related Work

There have been plenty of works [44, 50, 51, 53] about encrypted databases (EDBs) proposed in the last twenty years and a large proportion of them pay attention to the leakage from queries over multi-dimensional data. Durak et al. [16] are the first to show how to break EDBs more effectively based on the union distribution of multiple columns instead of naively breaking each single column separately and independently. The most typical work relevant to this work on attacking EDB is a line of works [31, 33, 34] which applies the leakage from querying a single column to recover this column. When querying on multi-dimensional data via decomposing the query to multiple queries where each query filters a unique single column (like popular works [35, 36, 43, 44]), the adversary can apply these attacks to recover each column effectively. Moreover, it can explore the correlation between columns and further apply the correlation to attack the database more successfully [4, 16]. In short, attacking the database with the union of leakage from querying each single column is much easier than that with only the final result on querying the multiple columns [18].

To address such additional unexpected leakage towards every single column, there are two kinds of solutions to date. The first one is to build EDB based on fully homomorphic encryption (FHE) [21]. As FHE allows any computation circuit over encrypted data, it allows the server to calculate the encrypted query results like the final results of queries on multiple columns [3, 56]. However, such EDBs still suffer from the inefficiency of FHE and they have to be equipped with additional expensive techniques [15] to reveal the query results to the server in order to retrieve records conveniently. The second kind of solution is to apply functional encryption (FE) which can be used to both calculate the final query results and reveal query results to the server. It usually can achieve better efficiency as FE is specialized to EDBs. Nevertheless, FE is not as expressible as FHE, which means it is limited to only partial queries in reality. And it is still often far from practicality due to slow calculation and some unexpected disadvantages, e.g., the inflexibility of the scheme in [47] as we mentioned in Section IV-A.

In this paper, we follow [26, 47] to reduce the leakage on querying multi-dimensional data but *significantly* improve the practicality of FE schemes towards multi-dimensional filter and equi-join between two tables. This contribution comes from our observation of the inflexibility of [47] and we address this serious disadvantage elegantly with new FE schemes. Besides, we also modify our schemes to support the equality test between columns and on column aggregations, extending the expressiveness of FE-based EDBs.

A. Conclusion

In this work, we propose new FE schemes to achieve practical query processing over encrypted multi-dimensional data while reducing the query leakage as much as possible. We support a series of typical queries in databases including the multi-dimensional filter, equi-join between tables, and equality test between columns and on column aggregations. The leakage is only the well-defined function results corresponding to different queries. Notably, compared with prior works [26, 47], we are the first to achieve such leakages with the *flexibility* on column selections, achieving an impressive practicality enhancement.

References

[1] Diego F Aranha, Laura Fuentes-Castañeda, Edward Knapp, Alfred Menezes, and Francisco Rodríguez-Henríquez. Implementing pairings at the 192-bit security level. In *Pairing-Based Cryptography–Pairing 2012: 5th International Conference, Cologne, Germany, May 16-18, 2012, Revised Selected Papers 5*, pages 177–195. Springer, 2013.

[2] Mihir Bellare, Marc Fischlin, Adam O'Neill, and Thomas Ristenpart. Deterministic encryption: Definitional equivalences and constructions without random oracles. In *Advances in Cryptology–CRYPTO 2008: 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings 28*, pages 360–378. Springer, 2008.

[3] Song Bian, Zhou Zhang, Haowen Pan, Ran Mao, Zian Zhao, Yier Jin, and Zhenyu Guan. He3db: An efficient and elastic encrypted database via arithmetic-and-logic fully homomorphic encryption. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 2930–2944, 2023.

[4] Vincent Bindschaedler, Paul Grubbs, David Cash, Thomas Ristenpart, and Vitaly Shmatikov. The tao of inference in prifvacy-protected databases. *Cryptology ePrint Archive*, 2017.

[5] Alexandra Boldyreva, Nathan Chenette, Younho Lee, and Adam O'neill. Order-preserving symmetric encryption. In *Advances in Cryptology-EUROCRYPT 2009: 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings 28*, pages 224–241. Springer, 2009.

[6] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In *Proceedings of the 24th Annual International Conference on Theory and Applications of Cryptographic Techniques*, EUROCRYPT'05, page 440–456, Berlin, Heidelberg, 2005. Springer-Verlag.

[7] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In *Annual international cryptology conference*, pages 41–55. Springer, 2004.

[8] Dan Boneh and Matt Franklin. Identity-based encryption from the weil pairing. In Joe Kilian, editor, *Advances in Cryptology — CRYPTO 2001*, pages 213–229, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[9] Dan Boneh, Kevin Lewi, Mariana Raykova, Amit Sahai, Mark Zhandry, and Joe Zimmerman. Semantically secure order-revealing encryption: Multi-input functional encryption without obfuscation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 563–594. Springer, 2015.

[10] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In *Theory of*

Cryptography: 8th Theory of Cryptography Conference, TCC 2011, Providence, RI, USA, March 28-30, 2011. Proceedings 8, pages 253–273. Springer, 2011.

[11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In Yuval Ishai, editor, *Theory of Cryptography*, pages 253–273, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[12] Xinle Cao, Yuhan Li, Dmytro Bogatov, Jian Liu, and Kui Ren. Secure and practical functional dependency discovery in outsourced databases. Cryptology ePrint Archive, Paper 2023/1969, 2023. https://eprint.iacr.org/2023/1969.

[13] Xinle Cao, Jian Liu, Hao Lu, and Kui Ren. Cryptanalysis of an encrypted database in sigmod '14. *Proc. VLDB Endow.*, 14(10):1743–1755, jun 2021.

[14] Zhao Chang, Dong Xie, Sheng Wang, and Feifei Li. Towards practical oblivious join. pages 803–817, 06 2022.

[15] Seung Geol Choi, Dana Dachman-Soled, S. Dov Gordon, Linsheng Liu, and Arkady Yerukhimovich. Compressed oblivious encoding for homomorphically encrypted search. *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021.

[16] F Betül Durak, Thomas M DuBuisson, and David Cash. What else is revealed by order-revealing encryption? In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1155–1166, 2016.

[17] Saba Eskandarian and Matei Zaharia. Oblidb: oblivious query processing for secure databases. *Proc. VLDB Endow.*, 13(2):169–183, oct 2019.

[18] Francesca Falzon, Evangelia Anna Markatou, Akshima, David Cash, Adam Rivkin, Jesse Stern, and Roberto Tamassia. Full database reconstruction in two dimensions. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 443–460, 2020.

[19] David Freeman, Michael Scott, and Edlyn Teske. A taxonomy of pairing-friendly elliptic curves. *Journal of cryptology*, 23:224–280, 2010.

[20] Sanjam Garg, Payman Mohassel, and Charalampos Papamanthou. Tworam: Efficient oblivious ram in two rounds with applications to searchable encryption. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016*, pages 563–592, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[21] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 169–178, 2009.

[22] Paul Grubbs, Thomas Ristenpart, and Vitaly Shmatikov. Why your encrypted database is not secure. In *Proceedings of the 16th workshop on hot topics in operating systems*, pages 162–168, 2017.

[23] Paul Grubbs, Kevin Sekniqi, Vincent Bindschaedler, Muhammad Naveed, and Thomas Ristenpart. Leakage-abuse attacks against order-revealing encryption. *2017 IEEE Symposium on Security and Privacy (SP)*, pages 655–672, 2017.

[24] Hakan Hacigümüş, Bala Iyer, Chen Li, and Sharad Mehrotra. Executing sql over encrypted data in the database-service-provider model. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 216–227, 2002.

[25] Florian Hahn and Florian Kerschbaum. Searchable encryption with secure and efficient updates. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 310–320, 2014.

[26] Florian Hahn, Nicolas Loza, and Florian Kerschbaum. Joins over encrypted data with fine granular security. In *2019 IEEE 35th international conference on data engineering (ICDE)*, pages 674–685. IEEE, 2019.

[27] Susan Hohenberger and Brent Waters. Attribute-based encryption with fast decryption. In *Public-Key Cryptography–PKC 2013: 16th International Conference on Practice and Theory in Public-Key Cryptography, Nara, Japan, February 26–March 1, 2013. Proceedings 16*, pages 162–179. Springer, 2013.

[28] Antoine Joux. A one round protocol for tripartite diffie–hellman. In Wieb Bosma, editor, *Algorithmic Number Theory*, pages 385–393, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[29] Seny Kamara and Tarik Moataz. Sql on structurally-encrypted databases. In *Advances in Cryptology–ASIACRYPT 2018: 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2–6, 2018, Proceedings, Part I 24*, pages 149–180. Springer, 2018.

[30] Jonathan Katz and Yehuda Lindell. Introduction to modern cryptography. *(No Title)*, 2014.

[31] Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O'neill. Generic attacks on secure outsourced databases. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1329–1340, 2016.

[32] Sam Kim, Kevin Lewi, Avradip Mandal, Hart William Montgomery, Arnab Roy, and David J. Wu. Function-hiding inner product encryption is practical. In *IACR Cryptology ePrint Archive*, 2018.

[33] Evgenios M Kornaropoulos, Charalampos Papamanthou, and Roberto Tamassia. The state of the uniform: Attacks on encrypted databases beyond the uniform query distribution. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 1223–1240. IEEE, 2020.

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

[35] Kevin Lewi and David J Wu. Order-revealing encryption: New constructions, applications, and lower bounds. In *Proceedings of the 2016 ACM SIGSAC Conference on*

Computer and Communications Security, pages 1167–1178, 2016.

[36] Dongjie Li, Siyi Lv, Yanyu Huang, Yijing Liu, Tong Li, Zheli Liu, and Liang Guo. Frequency-hiding order-preserving encryption with small client storage. *Proceedings of the VLDB Endowment*, 14(13):3295–3307, 2021.

[37] Mingyu Li, Xuyang Zhao, Le Chen, Cheng Tan, Huorong Li, Sheng Wang, Zeyu Mi, Yubin Xia, Feifei Li, and Haibo Chen. Encrypted databases made secure yet maintainable. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, pages 117–133, 2023.

[38] Gang Luo, Curt J Ellmann, Peter J Haas, and Jeffrey F Naughton. A scalable hash ripple join algorithm. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 252–262, 2002.

[39] Victor Miller. The weil pairing, and its efficient calculation. *J. Cryptology*, 17:235–261, 09 2004.

[40] Muhammad Naveed, Seny Kamara, and Charles V Wright. Inference attacks on property-preserving encrypted databases. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 644–655, 2015.

[41] Vassiliy Ilyich Nechaev. Complexity of a determinate algorithm for the discrete logarithm. *Mathematical Notes-New York*, 55(1):165–172, 1994.

[42] Adam O'Neill. Definitional issues in functional encryption. Cryptology ePrint Archive, Paper 2010/556, 2010. https://eprint.iacr.org/2010/556.

[43] Rishabh Poddar, Tobias Boelter, and Raluca Ada Popa. Arx: A strongly encrypted database system. *IACR Cryptol. ePrint Arch.*, 2016:591, 2016.

[44] Raluca Ada Popa, Catherine MS Redfield, Nickolai Zeldovich, and Hari Balakrishnan. Cryptdb: processing queries on an encrypted database. *Communications of the ACM*, 55(9):103–111, 2012.

[45] Théo Ryffel, David Pointcheval, Francis Bach, Edouard Dufour-Sans, and Romain Gay. Partially encrypted deep learning using functional encryption. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[46] Jacob T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM (JACM)*, 27:701 – 717, 1980.

[47] Masoumeh Shafieinejad, Suraj Gupta, Jin Yang Liu, Koray Karabina, and Florian Kerschbaum. Equi-joins over encrypted data for series of queries. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, pages 1635–1648, 2022.

[48] Victor Shoup. Lower bounds for discrete logarithms and related problems. In *Advances in Cryptology—EUROCRYPT'97: International Conference on the Theory and Application of Cryptographic Techniques Konstanz, Germany, May 11–15, 1997 Proceedings 16*, pages 256–266. Springer, 1997.

[49] Junichi Tomida. Tightly secure inner product functional encryption: Multi-input and function-hiding constructions. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology – ASIACRYPT 2019*, pages 459–488, Cham, 2019. Springer International Publishing.

[50] Stephen Tu, M. Frans Kaashoek, Samuel Madden, and Nickolai Zeldovich. Processing analytical queries over encrypted data. *Proc. VLDB Endow.*, 6(5):289–300, mar 2013.

[51] Qian Wang, Meiqi He, Minxin Du, Sherman SM Chow, Russell WF Lai, and Qin Zou. Searchable encryption over feature-rich data. *IEEE Transactions on Dependable and Secure Computing*, 15(3):496–510, 2016.

[52] Wai Kit Wong, David Wai-lok Cheung, Ben Kao, and Nikos Mamoulis. Secure knn computation on encrypted databases. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 139–152, 2009.

[53] Wai Kit Wong, Ben Kao, David Wai Lok Cheung, Rongbin Li, and Siu Ming Yiu. Secure query processing with data interoperability in a cloud database environment. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, page 1395–1406, New York, NY, USA, 2014. Association for Computing Machinery.

[54] Cao Xinle, Feng Weiqi, Xu Quanqing, Yang Chuanhui, Liu Jian, and Ren Kui. Reduce leakages in query processing on encrypted multi-dimensional data with practicality. *https://anonymous.4open.science/r/EncryptedQuery-DF68*.

[55] Zhang Xu, Zhenyu Wu, Zhichun Li, Kangkook Jee, Junghwan Rhee, Xusheng Xiao, Fengyuan Xu, Haining Wang, and Guofei Jiang. High fidelity data reduction for big data security dependency analyses. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 504–516, 2016.

[56] Zhou Zhang, Song Bian, Zian Zhao, Ran Mao, Haoyi Zhou, Jiafeng Hua, Yier Jin, and Zhenyu Guan. ArcEDB: An arbitrary-precision encrypted database via (amortized) modular homomorphic encryption. Cryptology ePrint Archive, Paper 2024/1064, 2024. https://eprint.iacr.org/2024/1064.

[57] Joe Zimmerman. How to obfuscate programs directly. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 439–467. Springer, 2015.

[58] Richard Zippel. Probabilistic algorithms for sparse polynomials. In *Symbolic and Algebraic Computation*, 1979.