Practical Secure Conjunctive Query Revisited

Xinle Cao
AntGroup & Zhejiang University
caoxinle.cxl@antgroup.com

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

Rui Zhang
Zhejiang University
zhangrui98@zju.edu.cn

Jinfei Liu
Zhejiang University
jinfeiliu@zju.edu.cn

Jian Liu
Zhejiang University
liujian2411@zju.edu.cn

***Abstract*—Encrypted databases (EDBs) have emerged as a promising research direction within the database community over the past two decades. Designed to ensure both data confidentiality and functionality in cloud databases, EDBs introduce new challenges in secure database design. One critical challenge is how to support conjunctive queries over multiple attributes securely and practically. In this paper, we propose a novel encrypted database system,** FlexDB**, tailored for secure conjunctive query processing. Built upon a set of newly devised encryption schemes,** FlexDB **supports a wide range of query functionalities, including single/multi-value filter, equi-join, equality test, and even range query. More importantly, it achieves strong security guarantees by ensuring that intermediate results on individual attributes remain hidden throughout query execution. At the same time,** FlexDB **offers improved theoretical complexity and practical performance in both storage overhead and computational efficiency. These advances in functionality, security, and efficiency make** FlexDB **a highly practical solution for secure conjunctive query processing in EDBs. We validate these improvements through extensive experiments on both synthetic and real-world datasets. We have open-sourced the well-developed implementation of** FlexDB **at https://github.com/WeiqiNs/SecureConjunctiveQuery.**

I. Introduction

Encrypted databases (EDBs) [1], [2], [3], [4], [5] have emerged as a promising research direction, driven by the widespread adoption and convenience of cloud computing [6]. In this scenario, the client (also referred to as the data owner in related literature [7], [8]) aims to store its database on a cloud server to reduce costs associated with maintaining database management systems and purchasing storage hardware. Due to concerns about data privacy and security, the client encrypts the data before outsourcing it to the cloud. This prevents the untrusted cloud server from learning sensitive information while still allowing the client to perform SQL queries over the encrypted data. Traditional *all-or-nothing* encryption schemes [9], [10] are insufficient for this purpose, as they do not support efficient querying. To this end, various novel cryptographic techniques [11], [12], [13], [14] and EDB systems [15], [16], [17], [18], [19] have been proposed to address this practical yet challenging problem.

A. Problem and Motivation

a) Problem: Despite significant advances in EDBs over the past two decades, there are still many critical problems unresolved. One of the most well-known and pressing issues is how to prevent unexpected information leakage during conjunctive query processing. To illustrate this challenge, consider a relational table $\mathcal{T}_A$ with attributes $\{A_1, A_2, \ldots, A_{m_A}\}$, where m_A denotes the number of attributes. Suppose the client submits a typical conjunctive query of the form:

$$\texttt{SELECT} \ \star \ \texttt{FROM} \ \mathcal{T}_A \ \texttt{WHERE} \ A_1 \in s_1 \ \texttt{AND} \ \cdots \ \texttt{AND} \ A_{m_A} \in s_{m_A},$$

where each $s_i \subseteq \mathcal{D}_i$ denotes the set of values used to filter attribute A_i, and $\mathcal{D}_i$ represents the domain of attribute A_i.

Most existing EDB systems [1], [2], [5], [20] process such queries by filtering each attribute independently and then computing the intersection of the resulting record sets. This approach is largely driven by the availability of mature encryption techniques [12], [13], [14] for handling single-attribute conditions. Specifically, these systems compute intermediate result sets $\{R_1, R_2, \ldots, R_{m_A}\}$, where each $R_i \subseteq \mathrm{ID}(\mathcal{T}_A)$ contains the identifiers (e.g., primary keys or row indices) of records satisfying the filter condition on attribute A_i. The final result set R returned to the client is then computed as the intersection of all intermediate results, i.e., $R = \bigcap_{i=1}^{m_A} R_i$.

However, this natural strategy may lead to severe leakage of sensitive data, as the intermediate results (e.g., individual sets R_i) are directly exposed to the untrusted server. Such leakages compromise strong security guarantees and have been identified as dangerous by several prior works [21], [22], [23], [24]. Therefore, a stronger security guarantee is required: *leaking nothing besides the final intersection result R*, which is also the central security goal in this work. Nevertheless, achieving such a strict requirement in a practical manner remains a significant challenge.

b) Motivation: Secure conjunctive query (SCQ), as one of the most prominent research topics in EDBs, has been extensively studied in a long line of work [21], [22], [25], [26], [27], [28], [29]. Unfortunately, most existing solutions suffer from either limited functionality or high computational and communication overheads, rendering them impractical for real-world deployment. We summarize representative approaches and their limitations below:

- **SSE-based SCQ:** A notable class of SCQ schemes are those [23], [27], [30] based on *searchable symmetric encryption (SSE)*, such as the well-known OXT protocol [23].

These schemes are primarily designed for conjunctive keyword search in document databases. When adapted to relational databases, they only support single-value filters, where the client specifies a single value per attribute. Moreover, none of these schemes achieve the strict security guarantee we mentioned earlier.

- **FHE-based SCQ.** Another emerging technique used in SCQ is *fully homomorphic encryption (FHE)* [31], [32], [33], which allows arbitrary computation over encrypted data. However, FHE alone cannot directly reveal the outcomes of these computations, meaning it cannot return the query result set R to the client for record retrieval. Instead, the client must rely on costly post-processing techniques [34] to extract the selected records. As a result, FHE-based EDB systems still face significant challenges in efficiently supporting expressive query functionalities.

These limitations highlight the need for a new design paradigm that enables SCQ with strong security guarantees, while also supporting rich functionality with practicality.

B. Contribution Overview

a) New FE Schemes: In this paper, we explore an alternative cryptographic technique known as *functional encryption (FE)*, aiming to overcome the inherent limitations of both SSE and FHE in the context of EDB. As one of the most prominent techniques in EDBs, FE schemes support specific functionalities over encrypted data. Compared to SSE, it offers greater expressiveness; compared to FHE, it allows direct result retrieval without expensive post-processing. However, while FE is theoretically capable of supporting complex functionalities and strong security guarantees, constructing concrete and practical FE schemes remains a major challenge. Existing FE-based EDB systems are often limited in functionality and impractical for real-world deployment. For instance, the state-of-the-art (SOTA) work [25] does not support dynamic attributes, i.e., no attribute can be added or removed after initialization.

To bridge the gap between theoretical feasibility and practical deployment, we propose a family of novel FE schemes tailored specifically for SCQ. Our constructions offer two key properties that significantly improve upon prior FE schemes [25], [26], [35]:

1) **Flexible Complexities:** Our schemes are designed to adapt their computational complexity based on the number of attributes being filtered. Specifically, when filtering only l out of m attributes, the time complexity can be reduced to $\mathcal{O}(ln)$ instead of $\mathcal{O}(mn)$ used in prior works [25], where n denotes the total number of records. This flexibility enables our schemes to achieve both improved theoretical efficiency and better practical performance.
2) **Expressive Functionalities:** Our schemes support a wide range of expressive operations across all attributes, including single/multi-value filter, equi-join, equality test, and even range query. Notably, each attribute can simultaneously support multiple functionalities, rather than being restricted to only one.

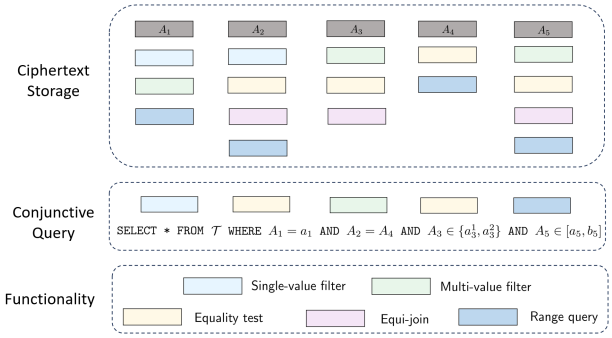


Fig. 1: Conjunctive query processing in FlexDB.

b) New EDB System: Based on the newly designed FE schemes, we develop a new EDB system named FlexDB to process secure conjunctive queries with practicality. We formally define a security model that ensures FlexDB reveals no information about intermediate results on individual attributes during conjunctive query execution. Importantly, FlexDB combines multiple FE schemes in a composable manner. That means, while each FE scheme is designed to support a single functionality, FlexDB seamlessly integrates them to handle conjunctive queries that require different functionalities across variable attributes. We illustrate this key feature in Fig. 1. To the best of our knowledge, combining multiple techniques and schemes to process such queries is common in EDBs [1], [2]. However, these works often expose sensitive intermediate results, leading to substantial leakage of private data.

Therefore, FlexDB represents a practical and secure EDB system that achieves both expressive functionality and strong leakage guarantees for conjunctive queries, making it a critical step toward real-world deployment of EDBs.

c) Our contributions: We summarize our contributions:

- **New FE schemes:** We design novel FE schemes for various functionalities, including single/multi-value filter, equi-join, equality test, and range query. They successfully protect the intermediate results on individual attributes when the client adopts them separately for conjunctive queries that require the same single functionality on all attributes.
- **New EDB system:** We introduce a new EDB system named FlexDB which non-trivially combines our FE schemes to support conjunctive queries that may require multiple different functionalities on attributes. Based on the new schemes and combination, FlexDB is much more practical than prior works [25], [26], [30] for conjunctive query processing.
- **Extensive evaluation:** We conduct a thorough empirical evaluation of FlexDB and compare it with theSOTA work [25]. The results demonstrate that FlexDB offers greater query flexibility and achieves superior performance, particularly when supporting multiple query types simultaneously. We also open-source a C++ library that includes implementations of both FlexDB and the SOTA system [25].

II. Background

A. Preliminaries

a) Notations: We use $[n]$ to denote the set $\{1, 2, \ldots, n\}$, and $\mathbb{Z}_n$ to represent the ring $\{0, 1, \ldots, n-1\}$ under modulo

n. The set of positive integers is denoted by $\mathbb{N}$. For two sets S_1 and S_2, we define the set difference $S_1 \setminus S_2 := \{x \in S_1 \mid x \notin S_2\}$. A function $\varepsilon : \mathbb{N} \to [0,1]$ is called *negligible*, if for every constant $c > 0$, there exists an integer n_0 such that for all $n > n_0$, it holds that $\varepsilon(n) < \frac{1}{n^c}$. We write $\varepsilon(n) = \mathsf{negl}(n)$ to indicate that ε is a negligible function in n. All algorithms considered in this paper are assumed to be probabilistic polynomial-time (PPT).

b) Bilinear Groups: Let $(\mathbb{G}_1, \mathbb{G}_2)$ be two distinct groups of prime order q, with generators $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$. A bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ maps elements to a target group $\mathbb{G}_T$ of the same order q. Following the notation of Kim et al. [35], we write the group operations in the groups *multiplicatively* and use $\mathbf{1}$ to denote the multiplicative identity. That is, for any $\delta \in 1, 2, T$ and $x \in \mathbb{G}_\delta$, $x \cdot \mathbf{1} = x$. Formally, a tuple $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, e)$ represents an asymmetric bilinear group [36], [37], [38] if it satisfies the following properties:

- *Computable:* The group operations in $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ and the map e are both efficiently computable.
- *Non-degenerate:* $e(g_1, g_2) \neq \mathbf{1}$.
- *Bilinear:* $\forall x, y \in \mathbb{Z}_q, e(g_1^x, g_2^y) = e(g_1, g_2)^{xy}$.

For simplicity, given a vector $\vec{x} = (x_1, ..., x_n) \in \mathbb{Z}_q^n$ where $n \in \mathbb{N}^+$, a group $\mathbb{G}$ with prime order q, and a group element $g \in \mathbb{G}$, we use $g^{\vec{x}}$ to denote the vector $(g^{x_1}, ..., g^{x_n})$. Naturally, we also adopt the following conventions for vector operations: for any $c \in \mathbb{Z}_q$, we let $(g^{\vec{x}})^c = g^{c\vec{x}}$, and for any vectors $\vec{x}, \vec{y} \in \mathbb{Z}_q^n$, we let $g^{\vec{x}} \cdot g^{\vec{y}} = g^{\vec{x}+\vec{y}}$. Additionally, the map e (also called *pairing* operation) on vectors describes the inner product of them:

$$e(g_1^{\vec{x}}, g_2^{\vec{y}}) = \Pi_{i \in [n]} e(g_1^{x_i}, g_2^{y_i}) = e(g_1, g_2)^{\langle \vec{x}, \vec{y} \rangle}.$$

c) Functional Encryption: In this work, we adopt *functional encryption* (FE) [11], [39], [40] to enable conjunctive queries securely. FE schemes encrypt data while permitting only well-defined leakages of a pre-defined function F to the untrusted server for query processing. A well-known type of FE in the database community is *order-preserving encryption* [5], [41], [42] which reveals order information to support range queries. In this work, we define a general FE encryption scheme $\Pi = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$, associated with a specific function F as below:

- $\mathsf{Setup}(1^\lambda, 1^m) \to (\mathsf{pp}, \mathsf{msk})$: On input encodings of a security parameter λ and the attribute number m, it outputs the public parameters pp (implicitly available to all other algorithms) and the master secret key msk.
- $\mathsf{KeyGen}(\mathsf{msk}, \vec{y}, \mathsf{aux}) \to \mathsf{sk}_{\vec{y}}$: On input the master secret key msk, a query vector $\vec{y} \in \mathbb{Z}_q^l$, and auxiliary information aux, it outputs a secret key $\mathsf{sk}_{\vec{y}}$.
- $\mathsf{Encrypt}(\mathsf{msk}, \vec{x}) \to \mathsf{ct}_{\vec{x}}$: On input the master secret key msk and a record vector $\vec{x} \in \mathbb{Z}_q^m$, it outputs a ciphertext $\mathsf{ct}_{\vec{x}}$.
- $\mathsf{Decrypt}(\mathsf{ct}, \mathsf{sk}) \to z$: On input a ciphertext ct and a functional secret key sk, it outputs either a message $z \in \mathbb{Z}_q$ or a symbol $\perp$.

To define the correctness of a FE scheme Π associated with F, consider any valid auxiliary input aux and any pair of vectors $(\vec{x}, \vec{y}) \in \mathbb{Z}_q^m \times \mathbb{Z}_q^l$. Let $(\mathsf{pp}, \mathsf{msk}) \leftarrow \Pi.\mathsf{Setup}(1^\lambda, 1^m)$, $\mathsf{sk}_{\vec{y}} \leftarrow \Pi.\mathsf{KeyGen}(\mathsf{msk}, \vec{y})$, and $\mathsf{ct}_{\vec{x}} \leftarrow \Pi.\mathsf{Encrypt}(\mathsf{msk}, \vec{x})$. Then Π is correct for function F, if the following holds:

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

The record vector $\vec{x}$ and query vector $\vec{y}$ stand for the encoding of a record and a specific query, respectively.

B. Related Work

Since the introduction of EDB in [6], there has been a long-standing battle between encryption techniques and attacks. We broadly divide the development of EDB into two major phases. In the first phase [1], [12], [43], [44], EDBs are designed to allow certain controlled leakages, including *access patterns* (i.e., which encrypted records are accessed) and exact *communication volume* (i.e., how many records are retrieved). These EDBs enable efficient query processing since the server can compute query results independently once a query is issued by the client. However, such schemes are quickly demonstrated to suffer from *generic attacks* [45], [46], [47]: the adversary can infer sensitive information according to only the above two leakages. Moreover, multiple such schemes are broken by even more efficient and simpler attacks [7], [48], [49] due to some extra unexpected leakages. These results cast doubt on the practicality of securely deploying such schemes.

To address these concerns, a second generation of EDBs emerged [50], [51], [52], [53]. These works aim to significantly reduce or eliminate the aforementioned two leakages and offer stronger security guarantees at the cost of *substantial* performance overhead. Representative approaches include those based on oblivious RAM (ORAM) [54], [55] and private information retrieval (PIR) [56], [57], such as `ObliDB` [58], `SEAL` [59], and others [60]. These techniques conceal leakage of access patterns and communication volume to some degree, marking a significant step forward in secure database design. However, they typically require multiple rounds of interaction and high communication bandwidth between the client and the server, making them impractical under typical network conditions and limiting their applicability in production.

This work aligns with a growing body of recent research [5], [25], [30], [61] that aims to enhance the security of EDBs within the first phase, i.e., without resorting to expensive oblivious primitives. While our approach cannot fully resist the generic attacks [45], [46], it significantly reduces the amount of leakage during conjunctive query processing. In other words, rather than aiming for complete leakage elimination, we focus on practically hardening the system to make the attacks substantially more difficult to succeed.

a) Conjunctive Query: The functionalities considered in this work have been explored in various existing works [2], [5], [62], [63]. However, most of them fail to achieve both expressive functionality and strong security when processing conjunctive queries. As discussed in Sec. I-A, SSE-based EDBs leak sensitive information about individual attributes even considering only the single-value filter functionality. The most recent work in this line [62] introduces equi-join

functionality at the cost of further leakage related to attribute frequencies. On the other hand, FHE-based EDBs can support strong security on arbitrary functionality, but suffer from severe performance overhead. For example, when processing equi-join, FHE-based EDBs require costly encrypted nested-loop joins, which render them impractical in real-world settings, as demonstrated empirically in [64].

In this work, we primarily compare our EDB with FE-based EDBs [25], [26], especially the SOTA work [25]. Hahn et al. [26] are the first to combine FE and SSE to support conjunctive queries, but their use of SSE weakens the overall security guarantee. As noted in [25], their scheme leaks additional information about equi-join attributes under multiple queries. To address this, Shafieinejad et al. [25] rely solely on FE to support conjunctive queries, achieving stronger security. Unfortunately, their approach inherits two key impracticalities from their underlying FE schemes: ❶ The computational complexity remains $\mathcal{O}(mn)$ regardless of whether the client filters one or all m attributes, where m and n denote the total number of attributes and records in the target table, respectively; ❷ Each attribute supports only a single functionality either equi-join or multi-value filter and this choice is fixed upon initialization. Moreover, it supports only static attributes: no attributes can be added or removed after initialization, which limits its practical applicability. Instead, our work overcomes all of these limitations by designing new FE schemes and combining them in non-trivial ways, while still maintaining the same strong security guarantees as in [25].

III. Design Goal

A. Functionality Goal

In this section, we introduce the functionalities that our new EDB provides. To clarify them formally, we consider a relational database $\mathcal{DB}$ consisting of one or multiple tables. Suppose a table $\mathcal{T}_A$ in $\mathcal{DB}$ includes n_A records and m_A attributes denoted by $\{A_1, A_2, ..., A_{m_A}\}$. Then we focus on conjunctive queries that operate on multiple attributes simultaneously. The target functionalities are defined as follows.

a) Independent Filter: The core and most fundamental functionality supported by our system is the *single/multi-value filter*, which independently selects records based on exact matches under one or more attributes:

$$\texttt{SELECT} \star \texttt{FROM } \mathcal{T}_A \texttt{ WHERE } A_{i_1} \in s_1 \texttt{ AND } \cdots \texttt{ AND } A_{i_{l_A}} \in s_{l_A},$$

where $\{i_1, i_2, \ldots, i_{l_A}\} \subset [m_A]$. The distinction between single-value and multi-value filters lies in the cardinality of each filter set s_{i_j}: a single-value filter corresponds to the case where $|s_{i_j}| = 1$, while a multi-value filter allows $|s_{i_j}| > 1$. For multi-value filtering, the client specifies a maximum size t for each filter set s_{i_j} in advance, ensuring that $\forall j \in [l_A], |s_{i_j}| \leq t$. These functionalities are widely used in practical applications, e.g., a university may query students from multiple departments and grade levels simultaneously.

While the multi-value filter filters only several discrete values, *range query* selects intervals containing consecutive values. In this case, each filter set s_{i_j} corresponds to an interval $[a_{i_j}, b_{i_j}]$. To the best of our knowledge, although conjunctive range queries have been studied in much prior works [65], [66], [67], none of these solutions achieve the same level of strong security guarantees as ours without relying on expensive FHE [31], [32], [68].

b) Correlated Filter: We further support an additional functionality named *equality test*. Unlike single-value filter, which compares individual attribute values against given constants, this functionality evaluates if a (weighted) aggregation of multiple attribute values equals a specified target value pt:

$$\texttt{SELECT} \star \texttt{FROM } \mathcal{T}_A \texttt{ WHERE } \sum_{j=1}^{l_A} w_{i_j} A_{i_j} = pt.$$

Such functionality has practical applications in real-world scenarios, e.g., a delivery company determines if a package was sent and received within the same city. To achieve this, the company can perform an equality test with the condition:

$$1 \times \text{SEND\_CITY} + (-1) \times \text{RECEIVE\_CITY} = 0,$$

where the weights are $\{1, -1\}$ and the target value is $pt = 0$.

The combination of aggregation and comparison operations has actually been explored in recent works [7], [31], [32], [69]. However, achieving such functionality without relying on FHE remains an open challenge. In this work, we provide a practical solution for realizing this composition, even if our current design focuses specifically on only equality checking.

c) Equi-Join: Another key functionality supported by our EDB is *equi-join*, one of the most fundamental operations in relational databases [70]. In addition to table $\mathcal{T}_A$ with attributes $\{A_1, A_2, \ldots, A_{m_A}\}$, we consider another table $\mathcal{T}_B$ consisting of attributes $\{B_1, B_2, \ldots, B_{m_B}\}$. The equi-join functionality connects these two tables based on an equality condition between two specified attributes (e.g., $A_1 = B_1$), while also allowing conjunctive filtering on other attributes. To illustrate this functionality, we present a typical example as following:

$$\texttt{SELECT} \star \texttt{FROM } \mathcal{T}_A \texttt{ JOIN } \mathcal{T}_B \texttt{ ON } A_1 = B_1 \texttt{ WHERE } (A_{i_1^A}, \ldots, A_{i_{l_A}^A}) \in s_1^A \times \cdots \times s_{l_A}^A \texttt{ AND } (B_{i_1^B}, \ldots, B_{i_{l_B}^B}) \in s_1^B \times \cdots \times s_{l_B}^B,$$

where $\{i_1^A, \ldots, i_{l_A}^A\} \subset \{2, \ldots, m_A\}$ and $\{i_1^B, \ldots, i_{l_B}^B\} \subset \{2, \ldots, m_B\}$. We do not extend the equi-join to multiple tables in this paper, but it is trivial to do that by repeatedly applying our solutions to two tables.

Composition. Although the above functionalities are presented separately, they are naturally intended to be composed within the same query. For instance, a client may wish to apply a multi-value filter on attribute A_1, a single-value filter on A_2, and an equality test between A_3 and A_4, all within a single conjunctive query while preserving strong security guarantees. This work aims to support compositions in a practical manner, thereby enabling a truly deployable EDB with expressive functionality.

B. Security Model

a) Adversary: In this paper, we adopt the classic setting [7], [71] and standard security model [2], [72], [73]

used in EDBs. The client uploads its encrypted database $\mathcal{DB}$ to a remote server and requires the system to support all functionalities introduced in Sec. III-A. We consider an *honest-but-curious* adversary: the server follows the protocol as specified by the client and does not actively deviate from it or attempt to compromise query issuance. However, the adversary can access the server during the whole protocol execution, i.e., it can passively observe anything available in the server during a long period of time [2], [7]. The adversary targets to infer sensitive information about the database from all its views inside the server and thus can be exactly captured by the untrusted cloud server [5], [7].

b) Security Notion: We now formalize the security notion for our EDB under the passive adversary. Following [2], [63], we adopt a *simulation-based* security definition. Intuitively, this ensures that no information about the encrypted data is revealed beyond what is inherently exposed by the *functionality type and final query results* [11], [39]. Let $\mathcal{T}_A := \{A_1, A_2, \ldots, A_{m_A}\}$ and $\mathcal{T}_B := \{B_1, B_2, \ldots, B_{m_B}\}$ denote two tables in $\mathcal{DB}$ with n_A and n_B records respectively. The leakage profile of executing a query Q is defined as:

$$\mathcal{L}(A, B, Q) = \{\mathcal{L}_1(A, B, Q),\ \mathcal{L}_2(A, B, Q),\ \mathcal{L}_3(A, B, Q)\},$$

where each component captures different aspects of information leaked during query execution. We define them more precisely as follows:

$$\begin{aligned}
&\mathcal{L}_1(A, B, Q) := \mathsf{Func}(A, B, Q),\\
&\mathcal{L}_2(A, B, Q) := \mathsf{Filter}(A, B, Q),\\
&\mathcal{L}_3(A, B, Q) := \mathsf{Join}(A, B, \mathcal{L}_2(A, B, Q)).
\end{aligned}$$

We elaborate on each leakage component below:

❶ $\mathsf{Func}(A, B, Q)$: This represents the types of functionalities performed by the query Q. Formally, it is defined as: $\{(A_i, \mathsf{F}_i)\}_{i=1}^{m_A} \cup \{(B_i, \mathsf{F}_i^B)\}_{i=1}^{m_B}$, where F_i denotes the set of functionalities applied to attribute A_i, which may include single/multi-value filter, equality test, equi-join, and range query. An attribute may be involved in multiple functionalities (i.e., $|\mathsf{F}_i| > 1$). If $|\mathsf{F}_i| = 0$, the attribute is not referenced in the query.

❷ $\mathsf{Filter}(A, B, Q)$: This captures the final set of records selected by query Q. It is defined as (R_A, R_B), where $R_A \subset [n_A]$ and $R_B \subset [n_B]$, indicating which rows in $\mathcal{T}_A$ and $\mathcal{T}_B$ are selected by Q.

❸ $\mathsf{Join}(A, B, \mathcal{L}_2(A, B, Q))$: This describes the equi-join results over the selected records. As in [25], it can be represented as a set of pairs: $\{(r_{d_1}, r_{d'_1}),\ \ldots,\ (r_{d_k}, r_{d'_k})\}$, where each pair $(r_{d_i}, r_{d'_i})$ indicates that the two records share the same value on the join attributes. These records may belong to the same table or different tables.

Based on the above leakage profiles, we now define the security guarantee. Following [2], [63], we say that our EDB composed of multiple FE schemes is $\mathcal{L}$-secure if any PPTadversary $\mathcal{A}$ cannot distinguish between two experiments: $(\mathbf{Real}, \mathbf{Ideal})$, even when given full access to all views available at the server. $\mathbf{Real}$ refers to the execution of our EDB using multiple FE schemes to process a query Q in the real world. $\mathbf{Ideal}$ refers to a simulation in the ideal world, which is generated by a PPTsimulator $\mathcal{S}$ based solely on the leakage function $\mathcal{L}$, along with public parameters such as the database size and attribute lengths [2]. The formal definition is as follows, and the corresponding security proof is deferred to the full version [74] for space.

Definition 1 (SIM-Security)**.** *Let $\mathcal{L}$ be the leakage function defined above and λ be the security parameter. We say an EDB built from functional encryption schemes $\Pi := \{\Pi_1, \Pi_2, \ldots\}$ is* SIM-secure *if for all sequences of queries $\mathcal{Q} = \{Q_1, Q_2, ...\}$ and for every* PPT *adversary $\mathcal{A}$, there exists a* PPT *simulator $\mathcal{S}$ such that:*

$$\left|\Pr[\mathbf{Real}_{\mathcal{A}}^{\Pi}(\lambda, \mathcal{Q}) = 1] - \Pr[\mathbf{Ideal}_{\mathcal{A},\mathcal{S},\mathcal{L}}(\lambda, \mathcal{Q}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

Remark. Note $\mathsf{Filter}(A, B, Q)$ and $\mathsf{Join}(A, B, \mathcal{L}_2(A, B, Q))$ achieves a significant leakage reduction compared with prior works [1], [2], [71]. The former avoids revealing the query results on individual attributes while the latter prevents the adversary from learning any information about unselected records on join attributes.

IV. Core Schemes

In this section, we propose new stateless and non-interactive FE schemes for single/multi-value filter, range query, equality test, and equi-join in conjunctive queries. They are the basis of our new EDB system for expressive functionality. Here we temporarily design them with *limited functionalities and fixed attributes*. In Sec. V, we will explain how to combine these schemes to achieve *flexible and expressive* functionality on variable attributes, as our design goal.

A. Single/Multi-value Filter

Consider a table $\mathcal{T}_A$ in the database $\mathcal{DB}$ with attributes $\{A_1, A_2, \ldots, A_{m_A}\}$. The FE scheme aims to filter values under all attributes, i.e., conjunctive queries of the form:

$$\texttt{SELECT} \star \texttt{FROM}\ \mathcal{T}_A\ \texttt{WHERE}\ A_1 \in s_1\ \texttt{AND}\ \cdots\ \texttt{AND}\ A_{m_A} \in s_{m_A},$$

where $\{s_1, \ldots, s_{m_A}\}$ are filter sets. For practical initialization, the client specifies the maximum size of each filter set during setup. Different attributes can have different maximal sizes. If needed, these limits can be updated post-initialization with incremental communication. For simplicity, here we assume each filter set contains at most t elements.

a) Intuition: The first step in our design is to enable filtering on a single attribute. Let a record be represented as a vector $\vec{x} = (x_1, \ldots, x_{m_A})$, where x_i denotes the value under attribute A_i. The goal of the filter is to determine whether $x_i \in s_i$, where $s_i = \{s_i^1, \ldots, s_i^t\}$. Following [25], this condition can be encoded using a polynomial whose roots correspond to the filter values. Specifically, define the polynomial:

$$f_i(x) = (x - s_i^1)(x - s_i^2)\cdots(x - s_i^t) = \sum_{j=0}^{t} c_j x^j.$$

It holds that $f_i(x_i) = 0$ if and only if $x_i \in s_i$. Thus, the filter can be implemented by evaluating $f_i(x_i)$ and checking whether the result is zero. Unlike [25], which evaluates $f_i(x)$ using the inner product between $(x^0, \ldots, x^t)$ and the coefficient vector $(c_0, \ldots, c_t)$, we require that *no coefficient in the vector is zero* for both security and efficiency reasons. To achieve this, we decompose $f_i(x)$ into two polynomials $f_i^1(x)$ and $f_i^2(x)$ with non-zero coefficients:

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

where $c_j \equiv c_j^1 + c_j^2 \mod q$ and $c_j^1, c_j^2 \neq 0$ for all j. We then compute the evaluations of $f_i^1(x)$ and $f_i^2(x)$ separately, and sum the results to obtain $f_i(x)$.

The second step involves combining the filter results across multiple attributes. To do so securely, we introduce randomness into each per-attribute evaluation. Specifically, we compute $f_i(x_i) + \delta_i$, where δ_i is a random offset associated with attribute A_i. This prevents the adversary from inferring intermediate filter results and thereby preserves the security. Finally, we aggregate the per-attribute results:

$$\sum_{i=1}^{m_A} [f_i(x_i) + \delta_i] - \Delta,$$

where $\Delta = \sum_{i=1}^{m_A} \delta_i$. The above expression is equal to 0 if $\forall i \in [m_A], f_i(x_i) = 0$. However, for the opposite direction, the aggregation is zero does not necessarily imply $f_i(x_i) = 0$. In reality, the data distributions make it very possible that the aggregation of non-zero polynomial evaluations is 0 and results in non-negligible *false positive (FP)* rate, i.e., records may be selected unexpectedly in processing the conjunctive query. This motivates us to multiply another random number γ_i in each polynomial $f_i(x)$ to calculate

$$\sum_{i=1}^{m_A} \gamma_i [f_i(x_i) + \delta_i] - \Delta,$$

where $\Delta = \sum_{i=1}^{m_A} \gamma_i \delta_i$. Now it is guaranteed that $\gamma_i f_i(x_i)$ is a random number in $\mathbb{Z}_q$ if $f_i(x_i) \neq 0$, so the aggregation is equal to 0 with a probability of $1/q$ when there exists i such that $f_i(x_i) \neq 0$. The value of q is always assumed to be exponential in the security parameter λ, so the FP rate is guaranteed to be negligible in λ. We do not follow [25], [21] to add randomness in polynomial evaluations by converting plaintexts to hash values because this cannot guarantee a negligible FP rate and also disables the plaintexts from being aggregated, which is needed in achieving the equality test functionality.

Besides, we further apply an optimization that merges the constant terms (i.e., those corresponding to x^0) across all attribute polynomials. Since each such term multiplies the same value $x^0 = 1$, we can combine them into a single coefficient. This reduces the number of required ciphertexts and multiplications from $2m_A$ to just 2, significantly improving performance without compromising functionality.

b) Construction: We present the detailed construction in Alg. 1. The client performs three subroutines to encrypt the database and process queries:

- During Setup, the client generates the bilinear pairing group parameters and sends them to the server [75]. Additionally, the client generates several random values, including a scalar μ and four vectors $(\vec{r}_1, \vec{r}_2, \vec{b}_1, \vec{b}_2)$. The vectors $(\vec{b}_1, \vec{b}_2)$ are used in multiplication operations and therefore must contain no zero entries.
- During Encrypt, for each attribute value x_i in the record vector $\vec{x} = (x_1, \ldots, x_{m_A})$, the client encrypts the tuple $\big((x_i)^1, \ldots, (x_i)^t\big)$, corresponding to the powers of x_i. Each component $(x_i)^v$, where $1 \leq v \leq t$, is encrypted using three parameters (α, b, r) such that the exponent becomes $\alpha b[(x_i)^v + r]$, ensuring semantic security.
  As we integrate the constant coefficients across all polynomials, the term $(x_i)^0 = 1$ is encrypted only twice. These two ciphertexts are stored in the final components of $\mathsf{ct}_{\vec{x}}^1$ and $\mathsf{ct}_{\vec{x}}^2$, respectively. Finally, the client includes $g^{\alpha\mu}$ in $\mathsf{ct}_{\vec{x}}$ to cancel out randomness during query processing.
- During KeyGen, the client constructs the filtering polynomials based on the filter sets $\{s_1, \ldots, s_m\}$. After decomposing each polynomial into two parts with non-zero coefficients, the client encrypts each non-constant coefficient using b^{-1} and a random scalar γ per polynomial. The constant terms of all polynomials are combined and incorporated into u. The value u serves to cancel the aggregate randomness, effectively playing the role of Δ in the intuition description.

For Decrypt on the server side, correctness follows from the inner product structure described in [25]. Intuitively, decryption verifies an identity of the form: $\langle \vec{x} + \vec{r}, \vec{c} \rangle - \langle \vec{r}, \vec{c} \rangle = \langle \vec{x}, \vec{c} \rangle$, where $\langle \cdot, \cdot \rangle$ denotes the inner product operation. We omit the full derivation due to space constraints. For the complexity, this scheme preserves only $\mathcal{O}(m)$ computational complexity in each subroutine. It requires a total of $(2m_A t + 1)$ pairing operations in Decrypt.

c) Multi-client FE: An additional optimization of our scheme is the reduction of client-side storage to $\mathcal{O}(1)$, which significantly benefits multi-client scenarios. In Alg. 1, the client stores four vectors: $(\vec{r}_1, \vec{r}_2, \vec{b}_1, \vec{b}_2)$. The security of the scheme relies on these vectors being *random*. Crucially, it suffices for these vectors to be *pseudorandom*, rather than truly random. This allows us to generate them using a pseudorandom function (PRF) [76], thereby eliminating the need to always store vectors in memory. In practice, each client only needs to store the PRF key. Upon invoking KeyGen, the client can deterministically regenerate the required vectors using the PRF, ensuring both efficiency and correctness. This approach reduces per-client storage from $\mathcal{O}(m_A t)$ to $\mathcal{O}(1)$, making our FE scheme highly scalable in multi-client deployments.

Furthermore, we observe that when clients issue conjunctive queries with only single-value filter, our scheme can incorporate techniques from an existing FE construction [28] to further reduce communication and computational overhead without changing our ciphertexts. This is because single-value filter inherently reveals the coefficients of non-constant terms in the

Algorithm 1 The FE scheme for single/multi-value filter

Client-side routines:

- $\mathsf{Setup}(1^\lambda, m_A, t) \to (pp, msk)$
  a) Generate an asymmetric bilinear group $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, e)$ with generators $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$.
  b) Choose a random scalar $\mu \in \mathbb{Z}_q^*$, two random vectors $\vec{r}_1, \vec{r}_2 \in \mathbb{Z}_q^{t \cdot m_A}$, and two random vectors $\vec{b}_1, \vec{b}_2 \in (\mathbb{Z}_q^*)^{t \cdot m_A}$, where all entries are non-zero.
  c) Output the public parameters $pp := (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, e, g_1, g_2)$ to the server, and retain the master secret key $msk := (\mu, \vec{r}_1, \vec{r}_2, \vec{b}_1, \vec{b}_2)$ locally.
  For notational convenience, for $p \in \{1, 2\}$, $i \in [m_A]$, and $j \in [t]$, let $r_{i,j}^p$ and $b_{i,j}^p$ denote the $((i-1) \cdot t + j)$-th entries of $\vec{r}_p$ and $\vec{b}_p$, respectively.
- $\mathsf{Encrypt}(msk, \vec{x}) \to \mathsf{ct}_{\vec{x}}$
  a) Sample a random scalar $\alpha \in \mathbb{Z}_q^*$.
  b) Construct ciphertexts $\mathsf{ct}_{\vec{x}} = (\mathsf{ct}_{\vec{x}}^1, \mathsf{ct}_{\vec{x}}^2, g_1^{\alpha\mu})$, where for each $p \in \{1, 2\}$, define:

$$\mathsf{ct}_{\vec{x}}^p := (\mathsf{ct}_1^p, \ldots, \mathsf{ct}_{m_A}^p), \quad \text{and} \quad \forall i \in [m_A], \quad \mathsf{ct}_i^p := \left(g_1^{\alpha b_{i,1}^p [(x_i)^1 + r_{i,1}^p]}, \ldots, g_1^{\alpha b_{i,t}^p [(x_i)^t + r_{i,t}^p]} \right).$$

- $\mathsf{KeyGen}(msk, \vec{y}) \to \mathsf{sk}_{\vec{y}}$
  a) For each attribute A_i, construct the filter polynomial $f_i(x)$, and decompose $f_i(x)$ into two polynomials $f_i^1(x)$ and $f_i^2(x)$ with non-zero coefficients.
  b) Sample a random scalar $\beta \in \mathbb{Z}_q^*$ and a random vector $\vec{\gamma} = (\gamma_1, \ldots, \gamma_{m_A}) \in \mathbb{Z}_q^{m_A}$ to mask the per-attribute evaluations.
  c) Let $c_{i,d}^1$ and $c_{i,d}^2$ denote the d-th coefficient of $f_i^1(x)$ and $f_i^2(x)$, respectively, for $d \in \{0, 1, \ldots, t\}$.
  d) Define the correction term:

$$u := \sum_{i=1}^{m_A} \gamma_i (c_{i,0}^1 + c_{i,0}^2) - \sum_{i=1}^{m_A} \sum_{j=1}^{t} \gamma_i (r_{i,j}^1 c_{i,j}^1 + r_{i,j}^2 c_{i,j}^2).$$

  e) Output the decryption key as $\mathsf{sk}_{\vec{y}} = (\mathsf{sk}_{\vec{y}}^1, \mathsf{sk}_{\vec{y}}^2, g_2^{\mu^{-1}\beta u})$, where for each $p \in \{1, 2\}$,

$$\mathsf{sk}_{\vec{y}}^p = (\mathsf{sk}_1^p, \ldots, \mathsf{sk}_{m_A}^p), \quad \text{and} \quad \forall i \in [m_A], \quad \mathsf{sk}_i^p := \left(g_2^{\beta\gamma_i (b_{i,1}^p)^{-1} c_{i,1}^p}, \ldots, g_2^{\beta\gamma_i (b_{i,t}^p)^{-1} c_{i,t}^p} \right).$$

Server-side routine:

- $\mathsf{Decrypt}(\mathsf{ct}_{\vec{x}}, \mathsf{sk}_{\vec{y}}) \to \{0, 1\}$
  a) Compute the pairing product:

$$\mathsf{res} = e(\mathsf{ct}_{\vec{x}}^1, \mathsf{sk}_{\vec{y}}^1) \cdot e(\mathsf{ct}_{\vec{x}}^2, \mathsf{sk}_{\vec{y}}^2) \cdot e(g_1^{\alpha\mu}, g_2^{\mu^{-1}\beta u}).$$

  b) Output 1 if $\mathsf{res} = e(g_1, g_2)^0$, i.e., the identity element in $\mathbb{G}_T$; otherwise, output 0.

polynomial representation, enabling us to compress the size of $\mathsf{sk}_{\vec{y}}$ and reduce the number of pairing operations to just two.

B. Other Functionalities

In this section, we build upon the single/multi-value filter scheme to construct schemes for range query, equality test, and equi-join. These extensions share a highly similar structure with the base scheme; therefore, we focus on presenting the design intuitions and omit detailed algorithmic descriptions.

a) Range Query: Inspired by existing SSE literature [65], [77], we represent a range query as a conjunction of bit-level single-value filters over the binary representation of the queried interval. For example, suppose the plaintext space is $[0, 127]$, where each value can be uniquely represented as a 7-bit string. To retrieve all plaintexts within the interval $[17, 25]$, prior approaches [65], [77] extend the original range to the smallest aligned interval $[16, 31]$ and then use a trapdoor to identify records falling within that superset. In contrast, our approach allows the client to directly specify a bitwise pattern that matches the desired range. Specifically, the values in $[16, 31]$ correspond exactly to those whose 7-bit representations begin with the prefix "001", i.e., strings matching the pattern "$001\| * * * *$." Compared to [65], [77], our method offers a strictly stronger security guarantee in conjunctive range queries, e.g., we can hide if two conjunctive queries request the same interval in one attribute, but they cannot.

Technically, there are two modifications to the scheme in Alg. 1. Firstly, the client encrypts each plaintext bit-by-bit, enabling single-value filter at the bit level. Secondly, for bits corresponding to wildcards (denoted by "$*$"), instead of applying a multi-value filter over $\{0, 1\}$, we set the associated polynomial to zero. This effectively imposes no constraint on

that bit. As previously discussed, we decompose this zero polynomial into the sum of two non-zero polynomials. This not only prevents the server from detecting whether a filter was applied on a specific bit for query privacy, but also enables us to uniformly set $t = 1$ for all bits. Remarkably, the dummy single-value filter for "$*$" hinders the usage of techniques in [28], i.e., we cannot apply the optimization in range query.

b) Equality Test: Given a record $\vec{x} = (x_1, x_2, \ldots, x_{m_A})$, the equality test functionality aims to determine whether:

$$\sum_{i=1}^{m_A} w_i x_i - pt = 0,$$

where w_i denotes the weight assigned to attribute A_i, and pt is a plaintext value specified by the client. Recall that in the single/multi-value filtering scheme, the server evaluates an expression of the form $\sum_{i=1}^{m_A} \gamma_i[f_i(x_i) + \delta_i] - \Delta$ where $\Delta = \sum_{i=1}^{m_A} \gamma_i \delta_i$. To support the equality test, we make the following three key modifications:

❶ For all $i \in [m_A]$, set $f_i(x_i) = x_i$;
❷ For all $i \in [m_A]$, set $\gamma_i = w_i$;
❸ Adjust the term Δ such that: $\Delta = \sum_{i=1}^{m_A} w_i \delta_i + pt$.

These changes can be seamlessly integrated into Alg. 1. As an example, modification ❸ can be implemented by updating the decryption key as follows:

$$\mathsf{sk}_{\vec{y}} = (\mathsf{sk}^1_{\vec{y}}, \mathsf{sk}^2_{\vec{y}}, g_2^{\mu^{-1}\beta u}) \quad \mapsto \quad \mathsf{sk}_{\vec{y}} = (\mathsf{sk}^1_{\vec{y}}, \mathsf{sk}^2_{\vec{y}}, g_2^{\mu^{-1}\beta(u-pt)}).$$

It is worth noting that this new functionality can be realized entirely within the KeyGen subroutine, the ciphertexts used for single/multi-value filtering remain valid under equality test. This backward compatibility stems from our design choice to encrypt the actual plaintext values directly, rather than hash values, as is common in schemes like [21], [25].

c) Equi-Join: To support the essential equi-join operation, we follow the approach of [25], [26], which advocates making the encryption deterministic once a record has been selected by other functionalities. Suppose we are given two tables $\mathcal{T}_A = \{A_1, A_2, \ldots, A_{m_A}\}$ and $\mathcal{T}_B = \{B_1, B_2, \ldots, B_{m_B}\}$. Let $\vec{x}_A = (x^A_1, \ldots, x^A_{m_A})$ and $\vec{x}_B = (x^B_1, \ldots, x^B_{m_B})$ denote the corresponding record vectors in these two tables. Suppose the equi-join condition is on attributes A_i and B_j, we construct their ciphertexts as follows:

$$\mathsf{ct}_{\vec{x}_A} := \left(g_1^{\mathsf{H}(x_i^A)+\alpha^A r^A},\ \mathsf{ct}'_{\vec{x}_A}\right), \mathsf{ct}_{\vec{x}_B} := \left(g_1^{\mathsf{H}(x_j^B)+\alpha^B r^B},\ \mathsf{ct}'_{\vec{x}_B}\right),$$

where $\mathsf{H}(\cdot)$ is a private hash function mapping from the plaintext space to $\mathbb{Z}_q$, and $\mathsf{ct}'_{\vec{x}_A}$, $\mathsf{ct}'_{\vec{x}_B}$ are ciphertext components used for other functionalities such as multi-value filter.

The KeyGen subroutine now needs to produce secret keys that fulfill two purposes:

❶ Select records based on conditions over $\mathsf{ct}'_{\vec{x}_A}$ and $\mathsf{ct}'_{\vec{x}_B}$;
❷ Remove the randomness applied to attributes A_1 and B_1, so that the encryption becomes deterministic for those attributes once the record is selected.

We illustrate how to adapt the single/multi-value filtering scheme (Alg. 1) to achieve this dual functionality. Recall that under the original scheme, the client can generate a decryption key $\mathsf{sk}'_{\vec{y}_A}$ such that: $e\left(\mathsf{ct}'_{\vec{x}_A},\ \mathsf{sk}'_{\vec{y}_A}\right) = e(g_1, g_2)^{\sum_{i=1}^{m_A} \gamma_i^A [f_i^A(x_i^A)+\delta_i^A] - \Delta^A}$, where $\Delta^A = \sum_{i=1}^{m_A} \gamma_i^A \gamma_i^A$. To achieve deterministic encryption on attribute A_1, the client samples a non-zero random scalar $\eta \in \mathbb{Z}_q^*$ and redefines Δ^A as: $\Delta^A = \sum_{i=1}^{m_A} \gamma_i^A \delta_i^A + \eta \alpha^A r^A$. Then, it outputs the updated secret key: $\mathsf{sk}_{\vec{y}_A} := \left(g_2^{\eta},\ \mathsf{sk}'_{\vec{y}_A}\right)$. Now the decryption becomes:

$$\begin{aligned} & e\left(\mathsf{ct}_{\vec{x}_A},\ \mathsf{sk}_{\vec{y}_A}\right) \\ &= e(g_1, g_2)^{\eta \mathsf{H}(x_i^A) + \eta\alpha^A r^A + \sum_{i=1}^{m_A} \gamma_i^A [f_i^A(x_i^A)+\delta_i^A] - \Delta^A} \\ &= e(g_1, g_2)^{\eta \mathsf{H}(x_i^A)}. \end{aligned}$$

Similarly, the client constructs $\mathsf{sk}_{\vec{y}_B} = (g_2^{\eta},\ \mathsf{sk}'_{\vec{y}_B})$ such that: $e\left(\mathsf{ct}_{\vec{x}_B},\ \mathsf{sk}_{\vec{y}_B}\right) = e(g_1, g_2)^{\eta \mathsf{H}(x_j^B)}$. Clearly, if $x_i^A = x_j^B$, then the above evaluations yield the same result, which implies deterministic encryption and enables the server to perform equi-join over selected records.

V. FlexDB Design

In this section, we present the design of our new EDB named FlexDB, which is tailored to support *expressive* functionality for conjunctive queries. We highlight two key innovations that significantly enhance its practicality:

- **Flexible Attributes:** FlexDB supports dynamic attribute management in conjunctive queries. This means that the client can perform operations on a subset of all available attributes, and further add or remove attributes as needed, without requiring a complete reconfiguration of the system.
- **Flexible Functionalities:** FlexDB enables composable functionalities over individual attributes, including single/multi-value filter, equality test, range query, and equi-join — all within a unified framework for conjunctive queries.

Compared with the state-of-the-art work [25], which fixes both the set of attributes and their associated functionalities, FlexDB offers substantially greater flexibility. For instance, [25] must process all attributes in a table even when a conjunctive query only references a small subset of them. Notably, our enhancements stem directly from the design philosophy underlying our new FE schemes: we encrypt each attribute independently while enabling queries over attributes as a whole. This decoupling of encryption and query composition is the key factor of FlexDB's expressiveness and practicality.

A. Flexible Attribute

To encrypt attribute values, FlexDB requires the client to identify the attribute sets for different functionalities. One attribute does not need all functionalities, e.g., a single-value filter may be sufficient for an attribute such as GENDER. Based on these sets, FlexDB applies our FE schemes independently to each attribute and stores the resulting ciphertexts on the server. Interestingly, the ciphertexts for multi-value filter can also be used for equality test and equi-join. This reusability allows FlexDB to minimize storage overhead.

For query processing, suppose there are m_A attributes in table $\mathcal{T}_A$, each encrypted for multiple functionalities. FlexDB enables the client to select any subset of these attributes

for querying while ignoring the rest. If only l_A attributes are selected, the computational complexity becomes $\mathcal{O}(l_A n_A)$ instead of $\mathcal{O}(m_A n_A)$ as in [25], where n_A denotes the number of records in $\mathcal{T}_A$. This efficiency gain stems from the relatively independent encryption of each attribute. Intuitively, the client treats the selected l_A attributes as a virtual sub-table. According to our algorithms, the client generates secret keys for only these l_A attributes. The server then computes over the corresponding ciphertexts to evaluate: $\sum_{j=1}^{l} \gamma_{i_j} \left[f_{i_j}(x_{i_j}) + \delta_{i_j}\right] - \Delta$, where $\Delta = \sum_{j=1}^{l} \gamma_{i_j} \delta_{i_j}$. During this process, the server observes that certain attributes are not involved. We consider this leakage acceptable: the disclosure of functionality types is common in EDBs [1], [2], [50], [71], and concealing it would incur prohibitive costs. For instance, if the client queries only 2 out of 30 attributes, hiding this information can result in a performance slowdown of up to $15\times$.

Moreover, FlexDB supports dynamic attributes also referred to as *flexible schema* in database literature [78], and commonly required in modern data platforms [79], [80], [81]. Specifically, FlexDB allows clients to dynamically add or remove attributes by simply appending or deleting the corresponding column of ciphertexts. This capability is crucial for real-world deployments, where it is impractical to require clients to predefine all attributes before encryption and maintain them unchanged indefinitely. FlexDB serves as a practical and secure demonstration of how to achieve dynamic schema management in EDBs.

B. Flexible Functionalities

The design of FlexDB aims to support expressive functionality over variable attributes. Therefore, in addition to flexible attribute management, FlexDB integrates multiple fundamental EF schemes for single functionalities to support composable queries. These queries can be arbitrary combinations of single/multi-value filter, equality test, range query, and equi-join. A typical example of such a query is:

$$\texttt{SELECT} \star \texttt{FROM} \mathcal{T}_A \texttt{JOIN} \mathcal{T}_B \texttt{ON} A_1 = B_1 \texttt{WHERE} A_2 = a_2 \texttt{ AND } A_3 \in \{a_3^1, a_3^2\} \texttt{ AND } A_4 = A_5 .$$

Such composability is crucial for real-world applications, as it significantly broadens the scope of supported queries and enables practical deployment in production. To realize this capability in the example above, FlexDB extracts ciphertexts from different FE schemes corresponding to each functionality: equi-join ciphertexts for $\{A_1, B_1\}$, single-value filter ciphertexts for A_2, multi-value filter ciphertexts for A_3, and equality test ciphertexts for $\{A_4, A_5\}$. This modular approach is feasible, but it introduces a critical security concern: since the underlying FE schemes operate independently, the server may learn information leaked by each individual scheme, e.g., whether a record satisfies $A_4 = A_5$ even if that record is not ultimately selected by the full conjunctive query. To guarantee the leakage as minimal as defined in Sec. III-B, we achieve the compatibility by assigning new randomness to these schemes independently, and combining them for query processing.

Intuitively, in each of our FE schemes, the goal of Decrypt is to compute two values of the form $(g_T^v, g_T^{v'})$, where $g_T = e(g_1, g_2)$, and v and v' represent the actual and expected results, respectively. The server checks whether $g_T^v = g_T^{v'}$ to determine the query results. For example, in single/multi-value filter, the server computes g_T^v and $g_T^{v'}$, where $v = \sum_{i=1}^{m_A} \gamma_i [f_i(x_i) + \delta_i] - \Delta$ and $v' = 0$. Therefore, while the three FE schemes should calculate $(g_T^{v_1}, g_T^{v_2}, g_T^{v_3})$, now we require them to calculate the following results instead:

$$(g_T^{v_1 + \alpha r_1}, g_T^{v_2 + \alpha r_2}, g_T^{v_3 - \alpha(r_1 + r_2)}).$$

where α, r_1 and r_2 are random numbers. The product of the three results above is still $g_T^{v_1+v_2+v_3}$, which completes both filter and equi-join as before. But the server cannot infer the values of $(g_T^{v_1}, g_T^{v_2}, g_T^{v_3})$ to obtain the information revealed by each individual scheme, leaving FlexDB to be secure as defined in our security model. For construction details, we describe them in the full version [74].

Arbitrary Equi-join: Here we explain how to enable arbitrary equi-join between tables $\mathcal{T}_A$ and $\mathcal{T}_B$ for further flexibility. In detail, the client preserves two suites of ciphertexts, i.e., for $K \in \{A, B\}$ and a record $\vec{x}_K$ in T_K, it stores $(\mathsf{ct}^1_{\vec{x}_K}, \mathsf{ct}^2_{\vec{x}_K})$ where $\mathsf{ct}^2_{\vec{x}_K}$ is used for other functionalities like single/multi-value filter and

$$\mathsf{ct}^1_{\vec{x}_K} = (g_1^{\mathsf{H}(x_1^K) + \alpha^K r_1^K}, g_1^{\mathsf{H}(x_2^K) + \alpha^K r_2^K}, ..., g_1^{\mathsf{H}(x_{m_K}^K) + \alpha^K r_{m_K}^K})$$

The vector $\vec{r}_K = (r_1^K, r_2^K, ..., r_{m_K}^K)$ is another random vector generated and stored by the client. To process any arbitrary equation $A_i = B_j$ $(i \in [m_A], j \in [m_B])$ plus filters on attributes, the client picks the i-th component from $\mathsf{ct}^1_{\vec{x}_A}$ and j-th component from $\mathsf{ct}^1_{\vec{x}_B}$ to compose

$$(g_2^{\mathsf{H}(x_j^A) + \alpha^A r_j^A}, \mathsf{ct}^2_{\vec{x}_A}), (g_2^{\mathsf{H}(x_j^B) + \alpha^B r_j^B}, \mathsf{ct}^2_{\vec{x}_B}).$$

As described in Sec. IV-B, the above two vectors are the same as the ciphertexts for equi-join between A_i and B_j. In this way, the client successfully reorganizes the ciphertexts to apply the FE scheme for equi-join. Conclusively, FlexDB extends the equi-join between two fixed attributes to any two attributes in two tables at the cost of no more than twice times storage. On the contrary, [25] takes $m_A m_B$ storage expansion to enable the equi-join between any two attributes in tables $\mathcal{T}_A$ and $\mathcal{T}_B$.

VI. Experimental Evaluation

We implement FlexDB in C++ to demonstrate its enhanced practicality over prior works. For a fair comparison, we also re-implement the scheme of Shafieinejad et al. [25] in C++, as their public implementation is in Python. For simplicity, we refer to their method as SGL^+ throughout our experiments. While SGL^+ was not originally designed to support all the functionalities considered in this paper, it can be modified accordingly. We ensure that all such modifications are applied as fairly as possible. All implementations, including that of SGL^+, have been open-sourced at https://github.com/WeiqiNs/SecureConjunctiveQuery.

System	Setup	Encrypt	Storage
SGL$^+$	$\mathcal{O}((m(t+1))^3)$	$\mathcal{O}((m(t+1))^2)$	$\mathcal{O}((m(t+1))^2)$
FlexDB	$\mathcal{O}(mt+1)$	$\mathcal{O}(mt+1)$	$\mathcal{O}(1)$

TABLE I: Theoretical complexity in the client side.

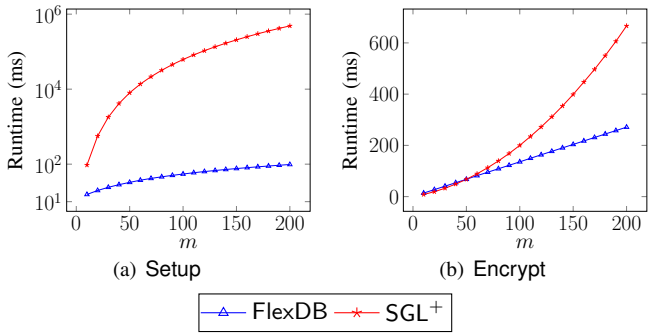


Fig. 2: Multi-filter client-side performance ($t = 5$).

A. Experimental Setup

a) Setup: We compare FlexDB with SGL$^+$ across standard practicality metrics: ❶ processing time, ❷ communication overhead, and ❸ storage cost. These three metrics exactly describe the performance of a method in executing queries, and thus are commonly considered in the literature [2], [58], [72], [82]. Our experiments are conducted using two separate machines located in Virginia, US, and San Francisco, US, communicating over a wide-area network (WAN) to reflect realistic network conditions. The measured round-trip latency is 35 ms, and the available bandwidth is 100 Mbps. The server machine runs Ubuntu 24.04 and is equipped with an Intel Xeon Platinum 8160 CPU with 32 virtual cores, 64 GB of RAM, and sufficient storage, whereas the client machine runs the same operating system with 2 virtual cores and 4 GB of RAM. For cryptographic operations, we utilize the RELIC library [83] to generate bilinear pairing groups, selecting the BLS12-381 curve for its 128-bit security level (i.e., $\lambda = 128$) and support for efficient Ate pairing. We employ AES-CBC and BLAKE2b from the OpenSSL package [84] to generate semantically secure ciphertexts and hashes, respectively.

b) Dataset: We select two large datasets to evaluate the performance of FlexDB. The first is a synthetic dataset, denoted by RND, which we generate with row counts ranging from 2^{10} to 2^{24} and column counts from 1 to 2^6. Each column entry is limited to at most 16 bytes to represent typical integer and string values. The second dataset, denoted by TPC-H, follows the setting in [25] and supports filter and equi-join operations on the TPC-H benchmark tables [85]. Specifically, we use the *Orders* and *Customer* tables, which contain 9 and 8 columns and 1,500,000 and 150,000 rows, respectively.

c) Evaluation Goal: Our experiments are designed to support the following claims:

- **Claim 1**: When supporting a single functionality, FlexDB offers greater query flexibility than SGL$^+$.
- **Claim 2**: When supporting multiple functionalities, FlexDB significantly outperforms SGL$^+$ in efficiency.

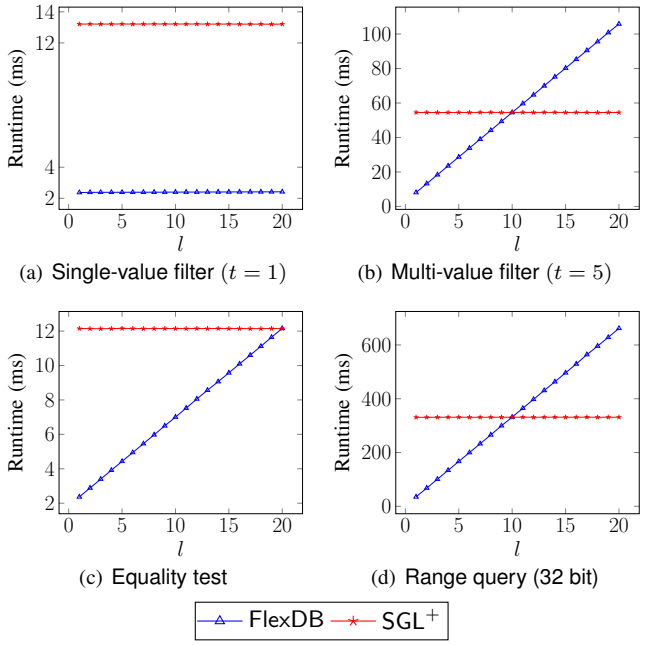


Fig. 3: Server performance on all functionalities ($m = 20$).

Recall that l denotes the number of operated attributes and m is the total number of attributes. In our experiments, we set $m = 20$ to reflect a common table size and vary l from 1 to 20. However, we emphasize that in real-world applications, l is often much smaller than m, as queries typically operate on only a small subset of attributes. For instance, a medical database may contain hundreds of attributes per patient, but a query might filter on only a few fields such as age or diagnosis. In such cases, the performance of SGL$^+$ degrades significantly due to its dependence on m. In contrast, FlexDB 's performance scales with l, allowing it to outperform SGL$^+$ even when supporting a single functionality. We use SF, MF, ET, and RQ to denote the single-value filter, multi-value filter, equality test, and range query functionalities, respectively. For MF, we set the number of filtered values t to 5 by default.

B. Client-side Performance

We first examine the client-side overhead of initialization and encryption, namely the cost of Setup and Encrypt in the underlying FE schemes. For fairness we compare FlexDB and SGL$^+$ only for the MF functionality, motivated by: ❶ MF is the core primitive studies in SGL$^+$; ❷ supporting additional functionalities increases the ciphertext length, which penalizes SGL$^+$ more. Table I lists the asymptotic complexities, and Fig. 2 shows the measured runtimes on RND. Both sub-figures demonstrate a significant efficiency gap in favor of FlexDB, as SGL$^+$ requires expensive matrix operations.

During Setup, SGL$^+$ inverts a matrix in $\mathbb{Z}_q^{m(t+1)\times m(t+1)}$, incurring $\mathcal{O}((m(t+1))^3)$ computation and $\mathcal{O}((m(t+1))^2)$ storage. FlexDB avoids this step entirely and achieves linear time and space instead. As Fig. 2(a) shows, the Setup runtime of SGL$^+$ rises steeply with m: at $m = 200$, it nears $5 * 10^5$ ms (about 8 minutes), whereas FlexDB needs only 98 ms, a $4962\times$ speed-up. For Encrypt, the two schemes are

comparable when $m \leq 100$, but FlexDB becomes faster once $m > 100$ because SGL$^+$ still performs matrix multiplications. We therefore expect an even larger advantage for FlexDB in settings with higher-dimensional tables.

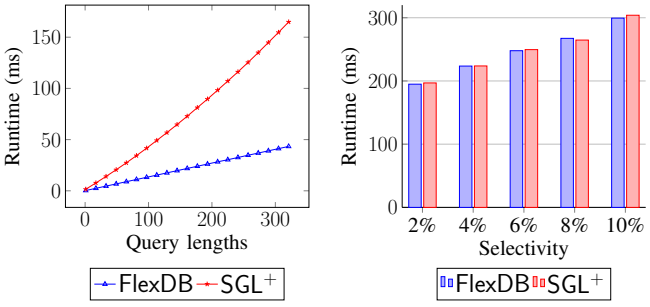
C. Single-Function Performance

We demonstrate the query flexibility of FlexDB by comparing its performance with SGL$^+$ when only a single functionality is enabled. The total cost of performing an operation is divided into three components: ❶ client-side computation cost (KeyGen), ❷ server-side computation cost (Decrypt), and ❸ communication cost between the client and server. We compare FlexDB and SGL$^+$ across each of these components individually, followed by an assessment of the total cost.

a) Separate costs: We first evaluate the server-side performance, i.e., the time required to decide whether a record satisfies a query (Decrypt), for FlexDB and SGL$^+$. Using the RND dataset with $m = 20$ attributes, we vary l, the number of queried attributes. To accurately estimate the runtime, we fix the database size to 2^{10} rows and execute the query over all rows to compute the average per-row running time. Note that for queries of identical size, the required computation per ciphertext is fixed. Therefore, the total server-side running time for a table with any number of rows can be estimated by multiplying the per-row cost by the total number of rows. The experimental results shown in Fig. 3 indicate that the runtime of SF, MF, ET, and RQ in FlexDB grows linearly with l, whereas in SGL$^+$ it scales with the total number of attributes m. We observe that when $l > \frac{m}{2}$, FlexDB has worse performance in MF and RQ. This is because, for functionalities that rely on underlying polynomial evaluations, we double the polynomial length to remove zero coefficients. As a result, the performance of Decrypt, which depends primarily on the size of the query, is affected. However, when $l \ll m$, FlexDB delivers substantial speed-ups, e.g., with $l = 1$ and $m = 20$, it is $\mathbf{6.7}\times$ faster for MF and $\mathbf{9.5}\times$ faster for RQ than SGL$^+$. These results highlight FlexDB's efficiency in supporting flexible query sizes and demonstrate its superior performance in scenarios involving sparse queries.

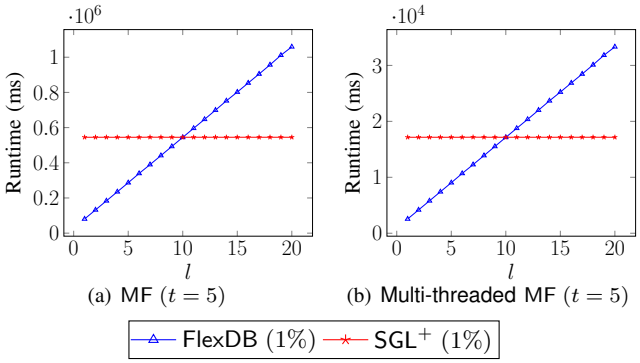
Fig. 4(a) shows the client-side cost of query generation for both FlexDB and SGL$^+$. We report this cost in terms of the resulting query size, as different functionalities generate queries of varying lengths, even when targeting the same number of attributes. While we show results for multiple query lengths in SGL$^+$, its query-generation cost depends solely on m rather than l, and remains fixed once the functionality is chosen. When comparing queries of the same size, SGL$^+$ consistently incurs higher cost than FlexDB, due to its underlying functional encryption scheme, where key generation involves a costly matrix multiplication step.

For communication overhead, both systems return only the semantically secure ciphertexts attached to FE ciphertexts. Hence, they incur nearly identical communication costs when retrieving the same number of records. We evaluate this overhead using the RND dataset with 10^4 rows, varying the selectivity (i.e., the proportion of records selected by the query) across $2\%, 4\%, 6\%, 8\%, 10\%$. The measured communication times are shown in Fig. 4(b), where we observe a linear increase with the number of selected rows. Even at 10% selectivity (i.e., retrieving 1000 rows), the communication time remains below 0.3 seconds. This has minimal impact on the overall query time, especially considering that per-row computation, particularly for RQ, can take longer than it.



(a) Client query generation (b) Communication time

Fig. 4: Client key generation and data retrieval time.



(a) MF ($t = 5$) (b) Multi-threaded MF ($t = 5$)

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

b) Total costs: Finally, we evaluate the total runtime of MF, which combines client-side query generation, server-side computation over 10,000 rows from the RND dataset, and the client–server communication. Because the communication overhead is negligible compared with the server workload, we fix the retrieval rate at 1% to emulate a realistic query and expect retrieving all records would yield comparable speed-ups. Fig. 5(a) confirms that FlexDB 's overall runtime acceleration matches its server-side gains, since server processing dominates the end-to-end cost.

We also note that server-side computation parallelizes naturally, as each ciphertext can be processed independently and the client query remains unchanged. By evenly partitioning the ciphertext and distributing the workload across 32 server cores, we observe a $31.8\times$ reduction in server computation time. With this parallelization, the overall speed-up of MF remains $6.7\times$ when $l = 1$ as shown in Fig. 5(b).

D. Composed Functionality Efficiency

We evaluate the efficiency of the two systems when all four functionalities SF, MF, ET, and RQ are enabled simultaneously. The results are depicted in Fig. 6(a). To emulate realistic workloads, we fix the other three functionalities to query two attributes whenever one functionality is under test.

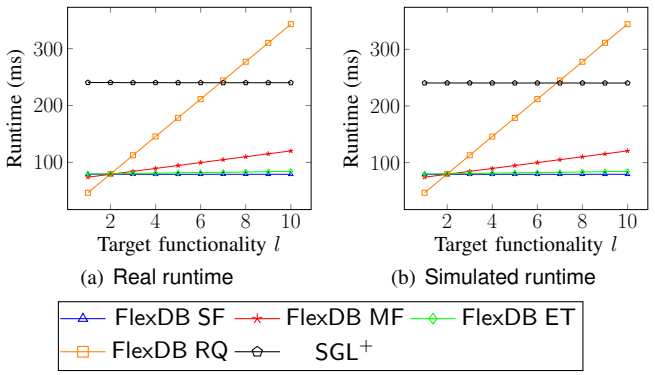


Fig. 6: Server performance on composed functionalities.

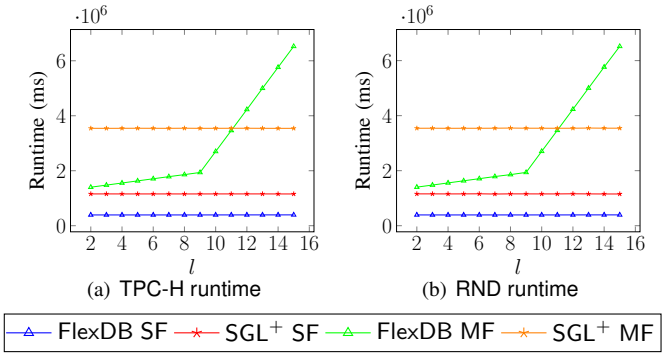


Fig. 7: Cost of equi-join on TPC-H and RND.

For example, the FlexDB SF curve corresponds to queries in which MF, ET, and RQ each operates on two attributes, while SF varies its filter from one to ten attributes. For a fair comparison, we assume that only ten attributes in SGL$^+$ support range queries, as its query size is fixed by the supported functionalities. Allowing range queries over all attributes would significantly increase the query length and further degrade its performance. Since all SGL$^+$ queries yield identical performance, we plot only a single line for it. We observe that FlexDB achieves a $2\times$ to $3\times$ speed-up for SF, MF, and ET. Although FlexDB performs worse than SGL$^+$ for RQ when $l > 7$, such queries are uncommon in practice. As noted in [65], range queries typically involve no more than four attributes. Under this realistic setting, FlexDB achieves a $1.7\times$ to $5.1\times$ speed-up for RQ. These speed-ups can further improve as the number of queried attributes decreases. Finally, we emphasize that server-side runtime is dominated by the number of pairings between the query and the ciphertexts. Fig. 6(b) confirms this observation by showing that the execution time of a synthetic workload with an equivalent number of random pairings closely matches that of actual queries.

E. Real-world dataset Efficiency

Finally, we compare the server-side performance of FlexDB and SGL$^+$ for equi-join queries on the TPC-H dataset. Because equi-join performance mostly depends on the join condition (e.g., SF versus MF), we report the time required for all steps except the join itself. Fig. 7(a) shows that FlexDB retains the flexibility and speed-ups observed in earlier experiments. The FlexDB MF curve becomes steeper as l increases because we first select additional attributes from Customers and then from Orders, the latter containing more rows. We stress that the join cost is identical for both systems, as each potential match involves comparing two elements in the target group. Given the simplicity of the join condition, the join step can be efficiently accelerated using standard hash-join techniques. Given that the join phase is not the primary performance bottleneck, the overall speed-up for complete equi-join queries is expected to closely follow the improvements reported in the pre-join phases. In Fig. 7(b), we also evaluate equi-join on the RND dataset, configured to match the sizes of the two tables from the TPC-H dataset. We observe nearly identical performance, which is expected since the server-side computations operate on fixed-size group elements that incorporate randomness, making the computation effectively *data-independent* regardless of the underlying input values.

F. Storage Overhead

We evaluate the storage overhead introduced by FlexDB and SGL$^+$. For client-side storage, in the setting of supporting only MF queries with $m = 20$ and $t = 5$, FlexDB requires 13 KB for msk, while SGL$^+$ incurs 679 KB, which is approximately $52\times$ more. Moreover, FlexDB 's client storage can be reduced to a single PRF key when randomness is generated via a PRF, incurring only minor runtime overhead. This trade-off is not viable for SGL$^+$, as part of its msk requires costly matrix inversion. For server-side storage (i.e., ciphertext size), when supporting a single functionality, FlexDB and SGL$^+$ are comparable for SF and ET, with ciphertext sizes of 1.03 KB and 1.13 KB, respectively, when $m = 20$. For MF and RQ, FlexDB requires about twice the storage of SGL$^+$, as it doubles the length of polynomial coefficients to avoid zero entries. For MF with $m = 20$ and $t = 5$, the ciphertext sizes are 9.9 KB for FlexDB and 5.1 KB for SGL$^+$. Nevertheless, as shown in previous experiments, this trade-off allows FlexDB to support much more flexible queries, whose sizes can be much smaller than those of SGL$^+$, and to achieve substantial speed-ups when multiple functionalities are used. With multiple functionalities, the storage overhead depends on the specific functionality applied to each attribute. Notably, in FlexDB, the ciphertext component for MF can be reused by both SF and ET on the same attribute.

VII. Conclusion

In this work, we proposed a suite of novel FE schemes that support simple conjunctive queries over multiple common functionalities, including single/multi-value filtering, equality testing, range queries, and equi-joins while offering strong security guarantees. Specifically, our schemes reveal only the final query result across all involved attributes, rather than leaking intermediate information about individual attributes. Building upon these new FE schemes, we designed and implemented FlexDB, a flexible EDB capable of processing complex and composable conjunctive queries in a secure and practical manner. To the best of our knowledge, FlexDB is the first EDB to simultaneously achieve both expressive functionality and strong security with practicality.

REFERENCES

[1] R. A. Popa, C. M. Redfield, N. Zeldovich, and H. Balakrishnan, "Cryptdb: processing queries on an encrypted database," *Communications of the ACM*, vol. 55, no. 9, pp. 103–111, 2012.

[2] R. Poddar, T. Boelter, and R. A. Popa, "Arx: A strongly encrypted database system." *IACR Cryptol. ePrint Arch.*, vol. 2016, p. 591, 2016.

[3] Y. Zheng, H. Zhu, R. Lu, S. Zhang, Y. Guan, F. Wang, J. Shao, and H. Li, "Secure similarity queries over vertically distributed data via tee-enhanced cloud computing," *IEEE Transactions on Information Forensics and Security*, 2024.

[4] H. Hu, J. Xu, C. Ren, and B. Choi, "Processing private queries over untrusted data cloud through privacy homomorphism," in *2011 IEEE 27th International Conference on Data Engineering*. IEEE, 2011, pp. 601–612.

[5] D. Li, S. Lv, Y. Huang, Y. Liu, T. Li, Z. Liu, and L. Guo, "Frequency-hiding order-preserving encryption with small client storage," *Proceedings of the VLDB Endowment*, vol. 14, no. 13, pp. 3295–3307, 2021.

[6] H. Hacigümüş, B. Iyer, C. Li, and S. 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*, 2002, pp. 216–227.

[7] X. Cao, J. Liu, H. Lu, and K. Ren, "Cryptanalysis of an encrypted database in sigmod '14," *Proc. VLDB Endow.*, vol. 14, no. 10, p. 1743–1755, jun 2021. [Online]. Available: https://doi.org/10.14778/3467861.3467865

[8] Y. Lu *et al.*, "Privacy-preserving logarithmic-time search on encrypted data in cloud." in *NDSS*, 2012.

[9] R. L. Rivest, "All-or-nothing encryption and the package transform," in *Fast Software Encryption: 4th International Workshop, FSE'97 Haifa, Israel, January 20–22 1997 Proceedings 4*. Springer, 1997, pp. 210–218.

[10] A. Desai, "The security of all-or-nothing encryption: Protecting against exhaustive key search," in *Annual International Cryptology Conference*. Springer, 2000, pp. 359–375.

[11] D. Boneh, A. Sahai, and B. 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*. Springer, 2011, pp. 253–273.

[12] A. Boldyreva, N. Chenette, Y. Lee, and A. 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*. Springer, 2009, pp. 224–241.

[13] M. Bellare, M. Fischlin, A. O'Neill, and T. 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*. Springer, 2008, pp. 360–378.

[14] F. Hahn and F. Kerschbaum, "Searchable encryption with secure and efficient updates," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, 2014, pp. 310–320.

[15] H. Li, J. Shi, Q. Tian, Z. Li, Y. Fu, B. Shen, and Y. Tu, "E nc 2 db: A hybrid and adaptive encrypted query processing framework," in *International Conference on Database Systems for Advanced Applications*. Springer, 2024, pp. 54–70.

[16] H. Hu, J. Xu, X. Xu, K. Pei, B. Choi, and S. Zhou, "Private search on key-value stores with hierarchical indexes," in *2014 IEEE 30th International Conference on Data Engineering*. IEEE, 2014, pp. 628–639.

[17] Y. Peng, L. Wang, J. Cui, X. Liu, H. Li, and J. Ma, "Ls-rq: A lightweight and forward-secure range query on geographically encrypted data," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 1, pp. 388–401, 2020.

[18] Z. Han, Q. Ye, and H. Hu, "Otki-f: An efficient memory-secure multi-keyword fuzzy search protocol," *Journal of Computer Security*, vol. 31, no. 2, pp. 129–152, 2023.

[19] Y. Miao, J. Ma, X. Liu, J. Weng, H. Li, and H. Li, "Lightweight fine-grained search over encrypted data in fog computing," *IEEE Transactions on Services Computing*, vol. 12, no. 5, pp. 772–785, 2018.

[20] K. Lewi and D. J. Wu, "Order-revealing encryption: New constructions, applications, and lower bounds," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 1167–1178.

[21] P. Golle, J. Staddon, and B. Waters, "Secure conjunctive keyword search over encrypted data," in *Applied Cryptography and Network Security: Second International Conference, ACNS 2004, Yellow Mountain, China, June 8-11, 2004. Proceedings 2*. Springer, 2004, pp. 31–45.

[22] L. Ballard, S. Kamara, and F. Monrose, "Achieving efficient conjunctive keyword searches over encrypted data," in *International conference on information and communications security*. Springer, 2005, pp. 414–426.

[23] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Roşu, and M. Steiner, "Highly-scalable searchable symmetric encryption with support for boolean queries," in *Advances in Cryptology–CRYPTO 2013: 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*. Springer, 2013, pp. 353–373.

[24] F. B. Durak, T. M. DuBuisson, and D. Cash, "What else is revealed by order-revealing encryption?" in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 1155–1166.

[25] M. Shafieinejad, S. Gupta, J. Y. Liu, K. Karabina, and F. Kerschbaum, "Equi-joins over encrypted data for series of queries," in *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, 2022, pp. 1635–1648.

[26] F. Hahn, N. Loza, and F. Kerschbaum, "Joins over encrypted data with fine granular security," in *2019 IEEE 35th international conference on data engineering (ICDE)*. IEEE, 2019, pp. 674–685.

[27] S. Lai, S. Patranabis, A. Sakzad, J. K. Liu, D. Mukhopadhyay, R. Steinfeld, S.-F. Sun, D. Liu, and C. Zuo, "Result pattern hiding searchable encryption for conjunctive queries," in *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, 2018, pp. 745–762.

[28] J. W. Byun, D. H. Lee, and J. Lim, "Efficient conjunctive keyword search on encrypted data storage system," in *European Public Key Infrastructure Workshop*. Springer, 2006, pp. 184–196.

[29] S. Patel, G. Persiano, J. Y. Seo, and K. Yeo, "Efficient boolean search over encrypted data with reduced leakage," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2021, pp. 577–607.

[30] Y. Wang, S.-F. Sun, J. Wang, X. Chen, J. K. Liu, and D. Gu, "Practical non-interactive encrypted conjunctive search with leakage suppression," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 4658–4672. [Online]. Available: https://doi.org/10.1145/3658644.3670355

[31] Z. Zhang, S. Bian, Z. Zhao, R. Mao, H. Zhou, J. Hua, Y. Jin, and Z. 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. [Online]. Available: https://eprint.iacr.org/2024/1064

[32] S. Bian, Z. Zhang, H. Pan, R. Mao, Z. Zhao, Y. Jin, and Z. 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*, 2023, pp. 2930–2944.

[33] X. Ren, L. Su, Z. Gu, S. Wang, F. Li, Y. Xie, S. Bian, C. Li, and F. Zhang, "Heda: multi-attribute unbounded aggregation over homomorphically encrypted database," *Proceedings of the VLDB Endowment*, vol. 16, no. 4, pp. 601–614, 2022.

[34] S. G. Choi, D. Dachman-Soled, S. D. Gordon, L. Liu, and A. Yerukhimovich, "Compressed oblivious encoding for homomorphically encrypted search," *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:237532340

[35] S. Kim, K. Lewi, A. Mandal, H. W. Montgomery, A. Roy, and D. J. Wu, "Function-hiding inner product encryption is practical," in *IACR Cryptology ePrint Archive*, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:7665590

[36] D. Boneh and M. Franklin, "Identity-based encryption from the weil pairing," in *Advances in Cryptology — CRYPTO 2001*, J. Kilian, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 213–229.

[37] A. Joux, "A one round protocol for tripartite diffie–hellman," in *Algorithmic Number Theory*, W. Bosma, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 385–393.

[38] V. Miller, "The weil pairing, and its efficient calculation," *J. Cryptology*, vol. 17, pp. 235–261, 09 2004.

[39] A. O'Neill, "Definitional issues in functional encryption," Cryptology ePrint Archive, Paper 2010/556, 2010, https://eprint.iacr.org/2010/556. [Online]. Available: https://eprint.iacr.org/2010/556

[40] Z. Ying, H. Li, J. Ma, J. Zhang, and J. Cui, "Adaptively secure ciphertext-policy attribute-based encryption with dynamic policy updating," *Science China. Information Sciences*, vol. 59, no. 4, p. 042701, 2016.

[41] X. Cao, J. Liu, Y. Shen, X. Ye, and K. Ren, "Frequency-revealing attacks against frequency-hiding order-preserving encryption," *Proc. VLDB Endow.*, vol. 16, no. 11, p. 3124–3136, jul 2023. [Online]. Available: https://doi.org/10.14778/3611479.3611513

[42] A. Roy Chowdhury, B. Ding, S. Jha, W. Liu, and J. Zhou, "Strengthening order preserving encryption with differential privacy," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 2519–2533.

[43] D. Liu and S. Wang, "Programmable order-preserving secure index for encrypted database query," in *2012 IEEE fifth international conference on cloud computing*. IEEE, 2012, pp. 502–509.

[44] F. Kerschbaum, "Frequency-hiding order-preserving encryption," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015, pp. 656–667.

[45] G. Kellaris, G. Kollios, K. Nissim, and A. O'neill, "Generic attacks on secure outsourced databases," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 1329–1340.

[46] M.-S. Lacharité, B. Minaud, and K. G. Paterson, "Improved reconstruction attacks on encrypted data using range query leakage," in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 297–314.

[47] P. Grubbs, M.-S. Lacharité, B. Minaud, and K. G. Paterson, "Pump up the volume: Practical database reconstruction from volume leakage on range queries," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 315–331.

[48] M. Naveed, S. Kamara, and C. V. Wright, "Inference attacks on property-preserving encrypted databases," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015, pp. 644–655.

[49] P. Grubbs, K. Sekniqi, V. Bindschaedler, M. Naveed, and T. Ristenpart, "Leakage-abuse attacks against order-revealing encryption," *2017 IEEE Symposium on Security and Privacy (SP)*, pp. 655–672, 2017. [Online]. Available: https://api.semanticscholar.org/CorpusID:18085613

[50] X. Cao, W. Feng, J. Liu, J. Zhou, W. Fang, L. Wang, Q. Xu, C. Yang, and K. Ren, "Towards practical oblivious map," *Cryptology ePrint Archive*, 2024.

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

[52] S. Kamara and T. Moataz, "Computationally volume-hiding structured encryption," in *Advances in Cryptology–EUROCRYPT 2019: 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19–23, 2019, Proceedings, Part II 38*. Springer, 2019, pp. 183–213.

[53] Z. Chang, D. Xie, S. Wang, and F. Li, "Towards practical oblivious join," 06 2022, pp. 803–817.

[54] C. W. Fletcher, L. Ren, A. Kwon, M. Van Dijk, and S. Devadas, "Freecursive oram: [nearly] free recursion and integrity verification for position-based oblivious ram," in *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2015, pp. 103–116.

[55] L. Tang, Q. Ye, H. Hu, and M. H. Au, "Secure traffic monitoring with spatio-temporal metadata protection using oblivious ram," *IEEE Transactions on Intelligent Transportation Systems*, vol. 24, no. 12, pp. 14 903–14 913, 2023.

[56] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan, "Private information retrieval," *Journal of the ACM (JACM)*, vol. 45, no. 6, pp. 965–981, 1998.

[57] H. Sun and S. A. Jafar, "The capacity of private information retrieval," *IEEE Transactions on Information Theory*, vol. 63, no. 7, pp. 4075–4088, 2017.

[58] S. Eskandarian and M. Zaharia, "Oblidb: oblivious query processing for secure databases," *Proc. VLDB Endow.*, vol. 13, no. 2, p. 169–183, oct 2019. [Online]. Available: https://doi.org/10.14778/3364324.3364331

[59] I. Demertzis, D. Papadopoulos, C. Papamanthou, and S. Shintre, "{SEAL}: Attack mitigation for encrypted databases via adjustable leakage," in *29th USENIX security symposium (USENIX Security 20)*, 2020, pp. 2433–2450.

[60] S. Sharma, Y. Li, S. Mehrotra, N. Panwar, K. Kumari, and S. Roychoudhury, "Information-theoretically secure and highly efficient search and row retrieval," *Proceedings of the VLDB Endowment*, vol. 16, no. 10, pp. 2391–2403, 2023.

[61] X. Wang, J. Ma, X. Liu, R. H. Deng, Y. Miao, D. Zhu, and Z. Ma, "Search me in the dark: Privacy-preserving boolean range query over encrypted spatial data," in *IEEE INFOCOm 2020-IEEE conference on computer communications*. IEEE, 2020, pp. 2253–2262.

[62] K. Du, J. Wang, J. Wu, and Y. Wang, "Scalable equi-join queries over encrypted database," Cryptology ePrint Archive, Paper 2024/1391, 2024. [Online]. Available: https://eprint.iacr.org/2024/1391

[63] D. Cash, J. Jaeger, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Rosu, and M. Steiner, "Dynamic searchable encryption in very-large databases: Data structures and implementation," 2014.

[64] K. J. Maliszewski, J. Quiané-Ruiz, J. Traub, and V. Markl, "What is the price for joining securely? benchmarking equi-joins in trusted execution environments," *Proc. VLDB Endow.*, vol. 15, no. 3, pp. 659–672, 2021. [Online]. Available: http://www.vldb.org/pvldb/vol15/p659-maliszewski.pdf

[65] F. Falzon, E. A. Markatou, Z. Espiritu, and R. Tamassia, "Range search over encrypted multi-attribute data," *Proceedings of the VLDB Endowment*, vol. 16, no. 4, 2022.

[66] Q. Liu, S. Wu, S. Pei, J. Wu, T. Peng, and G. Wang, "Secure and efficient multi-attribute range queries based on comparable inner product encoding," in *2018 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 2018, pp. 1–9.

[67] E. A. Markatou, F. Falzon, Z. Espiritu, and R. Tamassia, "Attacks on encrypted response-hiding range search schemes in multiple dimensions," *Proceedings on Privacy Enhancing Technologies*, 2023.

[68] S. Bian, H. Pan, J. Hu, Z. Zhang, Y. Fu, J. Hua, Y. Chen, B. Zhang, Y. Jin, J. Dong *et al.*, "Engorgio: An arbitrary-precision unbounded-size hybrid encrypted database via quantized fully homomorphic encryption," *Cryptology ePrint Archive*, 2025.

[69] X. Cao, J. Liu, H. Lu, Y. Liu, W. Liu, T. Wei, and K. Ren, "On the interoperability of encrypted databases," *IEEE Transactions on Dependable and Secure Computing*, 2024.

[70] A. V. Aho, C. Beeri, and J. D. Ullman, "The theory of joins in relational databases," *ACM Transactions on Database Systems (TODS)*, vol. 4, no. 3, pp. 297–314, 1979.

[71] W. K. Wong, B. Kao, D. W. L. Cheung, R. Li, and S. M. 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*, ser. SIGMOD '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 1395–1406. [Online]. Available: https://doi.org/10.1145/2588555.2588572

[72] X. Cao, Y. Li, D. Bogatov, J. Liu, and K. Ren, "Secure and practical functional dependency discovery in outsourced databases," Cryptology ePrint Archive, Paper 2023/1969, 2023, https://eprint.iacr.org/2023/1969. [Online]. Available: https://eprint.iacr.org/2023/1969

[73] M. Li, X. Zhao, L. Chen, C. Tan, H. Li, S. Wang, Z. Mi, Y. Xia, F. Li, and H. Chen, "Encrypted databases made secure yet maintainable," in *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, 2023, pp. 117–133.

[74] C. Xinle, F. Weiqi, X. Quanqing, Y. Chuanhui, Z. Rui, L. Jinfei, and L. Jian, "Practical secure conjunctive query revisited," *https://github.com/WeiqiNs/SecureConjunctiveQuery*, 2025.

[75] A. Lewko, T. Okamoto, A. Sahai, K. Takashima, and B. Waters, "Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption," in *Advances in Cryptology–EUROCRYPT 2010: 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30–June 3, 2010. Proceedings 29*. Springer, 2010, pp. 62–91.

[76] J. Katz and Y. Lindell, "Introduction to modern cryptography," *(No Title)*, 2014.

[77] I. Demertzis, S. Papadopoulos, O. Papapetrou, A. Deligiannakis, and M. Garofalakis, "Practical private range search revisited," in *Proceedings of the 2016 International Conference on Management of Data*, 2016, pp. 185–198.

[78] S. Acharya, P. Carlin, C. Galindo-Legaria, K. Kozielczyk, P. Terlecki, and P. Zabback, "Relational support for flexible schema scenarios," *Proceedings of the VLDB Endowment*, vol. 1, no. 2, pp. 1289–1300, 2008.

[79] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's highly available key-value store," *ACM SIGOPS operating systems review*, vol. 41, no. 6, pp. 205–220, 2007.

[80] B. F. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H.-A. Jacobsen, N. Puz, D. Weaver, and R. Yerneni, "Pnuts: Yahoo!'s hosted data serving platform," *Proceedings of the VLDB Endowment*, vol. 1, no. 2, pp. 1277–1288, 2008.

[81] Z. H. Liu and D. Gawlick, "Management of flexible schema data in rdbmss-opportunities and limitations for nosql-." in *CIDR*, 2015.

[82] Q. Wang, M. He, M. Du, S. S. Chow, R. W. Lai, and Q. Zou, "Searchable encryption over feature-rich data," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 3, pp. 496–510, 2016.

[83] "Relic library," *https://github.com/relic-toolkit/relic*.

[84] "Openssl package," *https://github.com/openssl/openssl*.

[85] M. Barata, J. Bernardino, and P. Furtado, "Ycsb and tpc-h: Big data and decision support benchmarks," in *2014 IEEE International Congress on Big Data*, 2014, pp. 800–801.

[86] D. Boneh, X. Boyen, and H. Shacham, "Short group signatures," in *Annual international cryptology conference*. Springer, 2004, pp. 41–55.

[87] D. Boneh, X. Boyen, and E.-J. Goh, "Hierarchical identity based encryption with constant size ciphertext," in *Proceedings of the 24th Annual International Conference on Theory and Applications of Cryptographic Techniques*, ser. EUROCRYPT'05. Berlin, Heidelberg: Springer-Verlag, 2005, p. 440–456. [Online]. Available: https://doi.org/10.1007/11426639_26

[88] V. I. Nechaev, "Complexity of a determinate algorithm for the discrete logarithm," *Mathematical Notes-New York*, vol. 55, no. 1, pp. 165–172, 1994.

[89] V. 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*. Springer, 1997, pp. 256–266.

[90] D. Freeman, M. Scott, and E. Teske, "A taxonomy of pairing-friendly elliptic curves," *Journal of cryptology*, vol. 23, pp. 224–280, 2010.

[91] D. F. Aranha, L. Fuentes-Castañeda, E. Knapp, A. Menezes, and F. 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*. Springer, 2013, pp. 177–195.

[92] J. Zimmerman, "How to obfuscate programs directly," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2015, pp. 439–467.

[93] M. Ambrona, G. Barthe, R. Gay, and H. Wee, "Attribute-based encryption in the generic group model: Automated proofs and new constructions," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 647–664.

[94] C. E. Z. Baltico, D. Catalano, D. Fiore, and R. Gay, "Practical functional encryption for quadratic functions with applications to predicate encryption," in *Annual International Cryptology Conference*. Springer, 2017, pp. 67–98.

[95] R. Zippel, "Probabilistic algorithms for sparse polynomials," in *Symbolic and Algebraic Computation*, 1979. [Online]. Available: https://api.semanticscholar.org/CorpusID:15629042

[96] J. T. Schwartz, "Fast probabilistic algorithms for verification of polynomial identities," *Journal of the ACM (JACM)*, vol. 27, pp. 701 – 717, 1980. [Online]. Available: https://api.semanticscholar.org/CorpusID:8314102

APPENDIX

A. Notation Table

Notation	Description
$\mathcal{DB}$	the database
$\mathcal{A}, \mathcal{B}$	tables in $\mathcal{DB}$
A, B	attributes in $\mathcal{A}, \mathcal{B}$
n_A, n_B	number of records in $\mathcal{A}, \mathcal{B}$
m_A, m_B	number of attributes in $\mathcal{A}, \mathcal{B}$
Q	a conjunctive query towards $\mathcal{DB}$
l	the number of operated attributes
k	the number of selected records
$[n]$	the integer set $\{1, 2, ..., n\}$
$\mathbb{Z}_n$	the integer set $\{0, 1, 2, ..., n-1\}$
$\mathbb{N}^+$	the positive integer set $\{1, 2, ...\}$
S_1/S_2	the set $\{x \in S_1 \| x \notin S_2\}$
$\vec{x}$	a record vector
$\|\vec{x}\|_2$	the Euclidean norm of $\vec{x}$
$\mathbb{G}$	the group
q	the group prime order
g	the group generator
$e(\cdot, \cdot)$	the pairing operation
λ	security parameter
F	function for the conjunctive query
pp	public parameters
aux	auxiliary information
$\mathsf{negl}(\cdot)$	negligible function
a	a value for attribute filter
s	a filter set for attribute filter
t	the polynomial degree, also $\|s\|$
pt	the constant for equality test

TABLE II: Summary of notations.

B. Preliminary Continued

In this section, we provide additional preliminaries for our security proof. We begin by formally specifying the leakage functions permitted by our proposed schemes, followed by an introduction to the generic group model (GGM) used in our proof.

1) The Generic Group Model: The generic model of bilinear groups [86], [87] extends the generic group model (GGM) [88], [89]. In the GGM, all accesses to group elements are replaced and controlled by *handles*, which are random strings. The adversary interacts with a stateful oracle that implements group operations, including the pairing operation in the case of bilinear groups. This oracle maintains a mapping from handles to group elements and responds to all adversary queries using only handles without revealing any actual group elements. Since these handles are random, GGM ensures that group operations are executed as a black box.

However, without additional constraints, the GGM could theoretically return random strings without performing any actual group operations, i.e., there's no way to guarantee its correctness. Hence the model provides an additional oracle called the zero-test oracle, which checks whether an element (represented by some handle) is equal to zero. Without loss of generality, we also use the zero-test oracle to determine whether two handles correspond to the same element. We refer to [35], [90], [91] for more discussion about GGM. Formally, a generic bilinear group oracle is a stateful oracle $\mathcal{G}$ that handles bilinear group BG queries as the following:

- On a query $\mathsf{BG.Setup}(1^\lambda)$, the oracle $\mathcal{G}$ takes input the security parameter λ and outputs a prime q sampled with λ bits. In addition, $\mathcal{G}$ setups three empty maps (M_1, M_2, M_T) inside as the initial state so that any subsequent Setup will fail.
- On a query $\mathsf{BG.Encode}(x, \ell)$, the oracle $\mathcal{G}$ takes inputs an integer $x \in \mathbb{Z}_q$ and a label $l \in \{1, 2, T\}$. It produces a fresh random handle $h \leftarrow\$ \{0,1\}^\lambda$ and sets $M_l[h] = x$. Finally, it output h.
- On a query $\mathsf{BG.Op}(h_1, h_2, \ell)$, the oracle $\mathcal{G}$ takes inputs two handles (h_1, h_2) and the label $l \in \{1, 2, T\}$. It first checks that both h_1 and h_2 are present in map M_l; if not, it outputs $\perp$ to indicate invalid operations. If the check passes, $\mathcal{G}$ returns a fresh handle $h \leftarrow\$ \{0,1\}^\lambda$ and sets $M_l[h] = M_l[h_1] + M_l[h_2]$.
- On a query $\mathsf{BG.Pair}(h_1, h_2)$, the oracle $\mathcal{G}$ takes inputs two handles (h_1, h_2). It first checks if h_1 is present in M_1 and if h_2 is present in M_2; if not, it outputs $\perp$ to indicate invalid operations. If the check passes, $\mathcal{G}$ returns a fresh handle $h \leftarrow\$ \{0,1\}^\lambda$ and sets $M_T[h] = M_1[h_1] \cdot M_2[h_2]$.
- On a query $\mathsf{BG.ZT}(h, \ell)$, the oracle $\mathcal{G}$ takes inputs a handle h and a label $l \in \{1, 2, T\}$. It checks if $\exists h \in M_\ell$ such that $M_\ell[h] = 0$, outputs *True* if so and *False* otherwise.

a) Remark: While the generic bilinear map oracle is formally defined in terms of handles, we adopt the approach of [92], treating each oracle query as a query on a formal polynomial. The formal variables correspond to expressions provided through the $\mathsf{BG.Encode}$ query. Using the $\mathsf{BG.Add}$ and $\mathsf{BG.Pair}$ queries, one can construct new polynomial terms. Instead of checking whether an element is zero via the $\mathsf{BG.ZT}$ query, we verify whether the corresponding polynomial evaluates to zero.

b) Note on GGM: While security in GGM does not always translate directly to the standard model, many works [25], [35], [93], [94] related to EDBs have been proven secure under GGM. Notably, this includes attribute-based encryption [93] and more general functional encryption [94]. We stress that the use of GGM is to treat the underlying algebraic group as a black box, where elements are represented abstractly, and the adversary can only interact with them through group operations. This abstraction prevents the adversary from exploiting specific structural properties of group elements, making security proofs cleaner and more general.

c) Schwartz-Zippel lemma: Our proof also relies on the Schwartz-Zippel lemma [95], [96], which we state below:

Lemma 1. *(Schwartz-Zippel, as adapted in [35]) 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,*

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

In this section, we present formal proofs establishing that FlexDB satisfies the security defined in Definition 1 from Section III. To this end, we first prove the security of each underlying FE scheme used in our construction individually, and then demonstrate that they can be securely composed to preserve the overall security of FlexDB. Following the notation introduced in Section VI, we use SF, MF, ET, RQ, and EJ to denote single-value filter, multi-value filter, equality test, range query, and equi-join, respectively.

Theorem 1. FlexDB *satisfies SIM-security, as defined in Definition 1, in the generic group model when only the single-value filter functionality is used, i.e., only the results of* SF *are revealed to the adversary.*

Proof. To prove this theorem, we begin with a high-level proof sketch, followed by the introduction of formal variables used in the main proof. Finally, we discuss the construction of the simulator with oracle accesses provided by the generic group model and show its correctness.

d) 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 $\mathsf{SF}(\vec{x}^i, \vec{y}^j)$ where $\vec{x}^i$ is input to the i-th query call of the encryption oracle, $\vec{y}^j$ is input to the j-th query call of the key generation oracle, and $i, j \in [Q]$. 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 in the simulated game, so that its responses cannot be distinguished from real 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 $C \cdot Dec(\vec{x}^i, \vec{y}^j)$, for $C \in \mathbb{N}$ and $i, j \in [Q]$. If the input has only an "honest" component, the simulator responds the query by referencing its state, which maintains the information about the result of $\mathsf{SF}(\vec{x}^i, \vec{y}^j)$. If the query contains "dishonest" component, the simulator always returns "nonzero". It is easy to see that the simulator can perform decryptions correctly; what remains to be shown is that, in the real world, the "dishonest" component can hardly evaluate to zero, demonstrating that the ideal world simulator's responses accurately mimics the query answers in the real world.

e) 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:

$$\mathcal{R} = \left\{\alpha^i, \beta^i, \zeta^i\right\}_{i\in[Q]} \cup \left\{\omega_j^i\right\}_{i\in[Q],j\in[m]}$$

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

We define $\vec{x}_j^i$ as the j-th value in the input vector of the i-th Enc query for some $i \in [Q]$. Similarly, we define $\vec{y}_j^i$ as the j-th value in the input vector of the i-th KeyGen query for some $i \in [Q]$. Then, for all $i \in [Q]$ and $j \in [m]$, the formal variables introduced above represent the following:

- x_j^i: the value $\vec{x}_j^i$
- y_j^i: the value $\vec{y}_j^i$
- α^i: the random value α sampled in i-th Enc query
- β^i: the random value β sampled in i-th KeyGen query
- ω_j^i: the value $\alpha^i \cdot (x_j^i + r_j)$
- ζ^i: the value $\beta^i \cdot \sum_{j=1}^m (y_j^i + r_j)$

In the above two sets, the handles of formal variables in $\mathcal{R}$ are provided directly to the adversary. For easier analysis, we decompose these formal variables by subsets of formal variables contained within $\mathcal{T}$.

f) Details of the simulator: We first define the leakage map M_{SF} for the single dimensional filter functionality. Given an input of index pairs (i, j), this leakage map returns the $\mathsf{SF}(\vec{x}_i, \vec{y}_j)$. The simulator $\mathcal{S}$ begins by initializing empty maps M_1, M_2, and M_T. It then processes queries as follows:

- Enc: On input a vector $\vec{x}^i \in \mathbb{Z}_q^m$, the simulator $\mathcal{S}$ receives as input a new map M_{SF} and updates its state. Then, $\mathcal{S}$ samples a fresh handle $h_\alpha \leftarrow\$ \{0,1\}^\lambda$ and saves the mapping $M_1[h_\eta] = \eta^i$. Next, for each $j \in [m]$, $\mathcal{S}$ samples a fresh handle $h_j \leftarrow\$ \{0,1\}^\lambda$ and saves $M_1[h_j] = \omega_j^i$. Finally, $\mathcal{S}$ returns a ciphertext $\mathsf{ct} = (h_1, \ldots, h_m, h_\alpha)$.
- KeyGen: On input a vector $\vec{y}^i \in \mathbb{Z}_q^m$, the simulator $\mathcal{S}$ receives as input a new map M_{SF} and updates its state. The simulator samples fresh handles $h_\beta, h_\zeta \leftarrow\$ \{0,1\}^\lambda$ and saves $M_2[h_\beta] = \beta^i$ and $M_2[h_\zeta] = \zeta^i$. Finally, $\mathcal{S}$ returns a function key $\mathsf{sk} = (h_\beta, h_\zeta)$.
- 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 in formal variables in $\mathcal{R}$.

2) If $\ell \neq T$, outputs "non-zero".
3) $\mathcal{S}$ decomposes $M_T[h]$ to the following polynomial:

$$\sum_{i,j\in[Q]} c_{i,j} \cdot p_{i,j}\left(\eta^i, \{\omega_k^i\}_{k\in[m]}, \beta^j, \zeta^j\right) + f(\mathcal{T})$$

where $p_{i,j}$ is defined as:

$$\beta^j \cdot \sum_{k=1}^{m} \omega_k^i - \eta^i \cdot \zeta^j$$

4) If the polynomial f is not empty, $\mathcal{S}$ returns "non-zero".
5) Otherwise, for each polynomial $p_{i,j}$, the simulator $\mathcal{S}$ looks up its state for the output of $\mathsf{SF}(\vec{x}^i, \vec{y}^j)$. If all these outputs are 1, the simulator returns "True" and "False" otherwise.

g) Correctness of the simulator: We formally demonstrate that the simulator constructed above is efficient, and accurately responds to the zero-test query, ensuring that its output matches the distribution observed in the real-world game. To do so, we analyze each step involved in the simulator's response to the zero-test query. Let $p = M_\ell[h]$, where h is the handle queried by the adversary.

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 $\eta^i \cdot \omega^j$, 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 explain why the simulator always outputs a nonzero value when ℓ is 1 or 2. Consider the case where $\ell = 1$, by construction, the monomials the adversary obtains are responses to the encryption query. We can write the polynomial p with formal variables in $\mathcal{T}$ as:

$$p = \sum_{i\in[Q]} \alpha^i \cdot \left(c_0^i + \sum_{j\in[m]} c_j^i \cdot \left(x_j^i + r_j\right)\right)$$
$$= \sum_{i\in[Q]} \alpha^i \cdot \left(c_0^i + \sum_{j\in[m]} c_j^i \cdot r_j + \sum_{j\in[m]} c_j^i \cdot x_j^i\right),$$

where for all $i \in [Q]$, $c_0^i, c_0^1, \ldots, c_m^i$ are some constants. To show the correctness of this step, we argue that in the real game, when the polynomial over formal variables is not identically zero, i.e., when not all constants are zero, the above polynomial p is unlikely to evaluate to zero. We first emphasize that in this case, regardless of the choice of x_j^i, the polynomial p can be viewed as a function over the formal variables ${\alpha^i}_{i\in[Q]}$ and ${r_j}_{j\in[m]}$, with total degree at most 2. Since all these formal variables correspond to uniform random values sampled from $\mathbb{Z}_q$, the Schwartz–Zippel Lemma (Lemma 1) implies that the probability of the polynomial p evaluating to zero is at most $\frac{2}{q}$. Given that q is a prime of size λ, the security parameter, this probability is negligible in λ. Hence, the simulation proceeds correctly with overwhelming probability.

Similarly when $\ell = 2$, we write the polynomial p with formal variables in $\mathcal{T}$ as:

$$p = \sum_{i\in[Q]} \beta^i \left(c_0 + c_1 \cdot \sum_{j=1}^{m} \left(y_j^i + r_j\right)\right)$$
$$= \sum_{i\in[Q]} \beta^i \left(c_0 + c_1 \cdot \sum_{j=1}^{m} r_j + c_1 \cdot \sum_{j=1}^{m} y_j^i\right),$$

where c_0 and c_1 are constants. Similarly, the polynomial p can be viewed as a function over the formal variables ${\beta^i}_{i\in[Q]}$ and the sum $\sum_{j=1}^{m} r_j$. Since each r_j is sampled uniformly at random from $\mathbb{Z}_q$, their sum is also uniformly distributed over $\mathbb{Z}_q$. Hence, by invoking the Schwartz–Zippel Lemma, we again conclude that the probability of p evaluating to zero is negligible, and the simulation proceeds correctly with overwhelming probability.
3) Akin to step 1), the adversary can reorganize the polynomial to the desired format efficiently.
4) If the polynomial f is not empty, for $i, j \in [Q]$ and $k \in [m]$, f may contain the following types of monomials:

- $\alpha^i \cdot \beta^i$, which is of order 2,
- $\alpha^i \cdot \zeta^j$, which is of order 3,
- $\omega_k^i \cdot \beta^j$, which is of order 3,
- $\omega_k^i \cdot \zeta^j$, which is of order 4.

We show that when the polynomial f is non-empty, it can hardly evaluate to zero in the real game. To analyze this, we first categorize the monomials in f based on their degrees. It is evident that monomials of different degrees are distinct, as they consist of different numbers of uniform random values. Thus, we focus on the two monomials of degree 3: $\alpha^{i_1} \cdot \zeta^{j_1}$ and $\omega_k^{i_2} \cdot \beta^{j_2}$, for some $i_1, j_1, i_2, j_2 \in [Q]$. When $i_1 \neq i_2$ or $j_1 \neq j_2$, the two monomials are clearly independent, as they contain randomly and independently sampled values across different queries. Thus, we focus on the case where $i_1 = i_2$ and $j_1 = j_2$, meaning the two monomials share the same sampled randomness. In this scenario, we aim to determine whether, for some $i, j \in [Q]$, ζ^j can be expressed as a sum of some element in the power set of $\omega_1^i, \ldots, \omega_m^i$, excluding the full set itself. The exclusion of the full set ensures that this expression does not actually correspond to a valid decryption query. Suppose the elements chosen from the power set are $\omega_{k_1}^i, \ldots, \omega_{k_l}^i$, where $l < m$, and the indices satisfy $k_i < k_j$ for $i < j$

and $k_l \leq m$. Consider the polynomial:

$$f' = \sum_{h=1}^{l} \omega_{k_h}^i - \zeta^j$$

Regardless of the filter result $\mathsf{SF}(\vec{x}^i, \vec{y}^j)$, the polynomial f' cannot be identically zero in these variables. This follows from the fact that ζ^j is a sum of m random variables, while the power set considered contains fewer than m elements. Consequently, f' must be a nontrivial polynomial in some random variables, and we seek to determine the probability that it evaluates to zero. By applying the Schwartz-Zippel lemma, we conclude that f' evaluates to zero with at most $\frac{1}{q}$ probability. Since the adversary makes a polynomial number of oracle calls, we can assert that the simulator can correctly answer the zero-test oracle with only a negligible probability of error.

5) If the simulator has reached this step, the adversary is requesting the correct evaluation of $Dec(\vec{x}^i, \vec{y}^j)$ for some $i, j \in [Q]$. The simulator then refers to the map storing all SF results, i.e. the permitted function leakages. By construction, if $\mathsf{SF}(\vec{x}^i, \vec{y}^j) = 1$, then $Dec(\vec{x}^i, \vec{y}^j)$ must be zero. Consequently, if $\mathsf{SF}(\vec{x}^i, \vec{y}^j) = 1$ for all i, j, the polynomial above evaluates to zero, and the simulator outputs "True." Otherwise, a random value is obtained, and the zero-test oracle should output "False."

This completes the correctness proof of the simulator.

Correctness of filtering subsets: We explain why the scheme remains secure even when only a subset of the columns is filtered. In this setting the adversary learns which ciphertext columns each query targets, which is a form of auxiliary leakage that is inherent to any correct decryption. This extra information does not alter the core of the proof: the encryption algorithm is unchanged, and the analysis to the key generation oracle is identical, as the function keys are still sums of independently sampled random field elements. The only adjustment appears in the zero-test step, where the simulator is now given the indices of the filtered columns for every key generation query. Equipped with this leakage, the simulator can still faithfully emulate decryption and detect malformed polynomials. The rest of the analysis proceeds as in the original proof, and the security guarantees continue to hold.

□

Theorem 2. FlexDB *satisfies SIM-security, as defined in Definition 1, in the generic group model when only the multi-value filter functionality is used, i.e., only the results of* MF *are revealed to the adversary.*

Proof. Because the schemes we introduced share similarities in their construction, the proof of this theorem closely follows the proof of Theorem 1. Hence, we omit the proof overview and begin by defining some formal variables.

h) 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 $\vec{x}_j^i$ as the j-th value in the input vector of the i-th Enc query for some $i \in [Q]$. Slightly different from the SF functionality, for the MF functionality, the input to the key generation oracle consists of m sets, each with a maximum possible size of t. The key generation oracle first interpolates a polynomial over each subset of values to obtain $t+1$ coefficients. We denote by $p_{j,k}^i$, where $j \in [m]$ and $k \in [t]$, the coefficient corresponding to the degree-k term in the j-th polynomial of the i-th key generation query. Naturally, $p_{j,0}^i$ represents the constant term in these polynomials. Then, for all $i \in [Q]$, $j \in [m]$, and $k \in [t]$, we define the following set of variables.

$$\mathcal{T} = \{\delta, \delta'\} \cup \left\{\alpha^i, \beta^i\right\}_{i\in[Q]} \cup \left\{r_i^1, r_i^2, b_i^1, b_i^2, bi_i^1, bi_i^2, \right\}_{i\in[m\cdot t]} \cup \left\{x_j^i, \gamma_j^i\right\}_{i\in[Q], j\in[m]} \cup \left\{c_j^i, cc_j^i\right\}_{i\in[Q], j\in[m\cdot t]}$$

The specifications of these variables are given as follows.

- δ: a random value
- δ': the inverse of δ, such that $\delta \cdot \delta' = 1$
- α^i: the random value α sampled in i-th Enc query
- β^i: the random value β sampled in i-th KeyGen query
- γ_j^i: the random value γ_j sampled in i-th KeyGen query
- r_i^1: a random value
- r_i^2: a random value
- b_i^1: a non-zero random value
- b_i^2: a non-zero random value
- bi_i^1: the inverse of b_i^1, such that $b_i^1 \cdot bi_i^1 = 1$
- bi_i^2: the inverse of b_i^2, such that $b_i^2 \cdot bi_i^2 = 1$
- x_j^i: the value $\vec{x}_j^i$
- c_j^i: a non-zero random value sampled in i-th KeyGen query
- cc_j^i: the value $p_{h,k}^i - c_j^i$, where $h \in [m], k \in [t]$ and $j = (h-1) * t + k$

Using the variables defined above, we now specify the set of variables provided to the adversary.

$$\mathcal{R} = \left\{\eta^i, \tau^i\right\}_{i\in[Q]} \cup \left\{\omega_{1,j}^i, \omega_{2,j}^i, \zeta_{1,j}^i, \zeta_{2,j}^i\right\}_{i\in[Q], j\in[m\cdot t]}$$

We first define a helper variable u^i as follows:

$$u^i = \sum_{j=1}^{m} p_{j,0}^i - \sum_{j=1}^{m\cdot t} \gamma_{\lceil \frac{j}{t} \rceil} \cdot (r_j^1 \cdot c_j^i + r_j^2 \cdot cc_j^i)$$

- η^i: the value $\delta \cdot \alpha^i$
- τ^i: the value $\delta' \cdot \beta^i \cdot u^i$
- $\omega_{1,j}^i$: the value $\alpha^i \cdot b_j^1 \cdot ((x_h^i)^k + r_j^1)$, where $h \in [m], k \in [t]$ and $j = (h-1) \cdot t + k$
- $\omega_{2,j}^i$: the value $\alpha^i \cdot b_j^2 \cdot ((x_h^i)^k + r_j^2)$, where $h \in [m], k \in [t]$ and $j = (h-1) \cdot t + k$
- $\zeta_{1,j}^i$: the value $\beta^i \cdot \gamma_{\lceil \frac{j}{t} \rceil} \cdot bi_j^1 \cdot c_j^i$
- $\zeta_{2,j}^i$: the value $\beta^i \cdot \gamma_{\lceil \frac{j}{t} \rceil} \cdot bi_j^2 \cdot cc_j^i$

i) 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 $\vec{x}^i \in \mathbb{Z}_q^m$, the simulator $\mathcal{S}$ receives as input a new map M_{MF} and updates its state. Then, $\mathcal{S}$ samples fresh handles for the formal variable η^i and each

formal variable in the set $\{\omega^i_{1,j}, \omega^i_{2,j}\}_{j\in[m\cdot t]}$, and returns these handles as the ciphertext.

- KeyGen: On input a set $S := \{s_1, \ldots, s_m\}$, the simulator $\mathcal{S}$ receives as input a new map M_{MF} and updates its state. Then, $\mathcal{S}$ samples fresh handles for the formal variable τ^i and each formal variable in the set $\{\zeta^i_{1,j}, \zeta^i_{2,j}\}_{j\in[m\cdot t]}$, and returns these handles as the ciphertext.
- Op: identical to the simulator described in Theorem 1.
- Pair: identical to the simulator described in Theorem 1.
- 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 in formal variables in $\mathcal{R}$.
  2) If $\ell \neq T$, outputs "non-zero".
  3) $\mathcal{S}$ decomposes $M_T[h]$ to the following polynomial:

$$\sum_{i,j\in[Q]} c_{i,j}\cdot p_{i,j}\left(\eta^i, \tau^j, \left\{\omega^i_{1,k}, \omega^i_{2,k}, \zeta^j_{1,k}, \zeta^j_{2,k}\right\}_{k\in[m\cdot t]}\right) + f(\mathcal{T})$$

  where $p_{i,j}$ is defined as:

$$\sum_{k=1}^{m\cdot t}\left(\omega^i_{1,k}\cdot\zeta^j_{1,k} + \omega^i_{2,k}\cdot\zeta^j_{2,k}\right) - \eta^i\cdot\tau^j$$

  4) If the polynomial f is not empty, $\mathcal{S}$ returns "non-zero".
  5) Otherwise, for each polynomial $p_{i,j}$, the simulator $\mathcal{S}$ looks up its state for the output of $\mathsf{MF}(\vec{x}^i, \vec{y}^j)$. If all these outputs are 1, the simulator returns "True" and "False" otherwise.

j) Correctness of the simulator: We show formally that the simulator constructed above is correct, efficient, 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) For the same reason as described in Theorem 1.
2) In the case where $\ell = 1$, by construction, the only monomials observed by the adversary are from responses to the encryption queries, which involve the following formal variables: η^i, $\omega^i_{1,j}$, and $\omega^i_{2,j}$. We first emphasize that if the polynomial p is not identically zero in the formal variables of $\mathcal{R}$, then it is also not identically zero in those of $\mathcal{T}$. As a result, the adversary obtains a polynomial over the variables α^i, x^i_j, γ^i_j, r^1_k, r^2_k, b^1_k, and b^2_k, for $i \in [Q]$, $j \in [m]$, and $k \in [m \cdot t]$. As before, this polynomial can be treated as a function over the formal variables α^i, r^1_k, r^2_k, b^1_k, and b^2_k. We stress that these variables correspond to independently and uniformly sampled random values in $\mathbb{Z}_q$. The total degree of the polynomial is at most 4. By the Schwartz–Zippel Lemma (Lemma 1), the probability that such a polynomial evaluates to zero is at most $\frac{4}{q}$, which is negligible in λ.
   The case where $\ell = 2$ is similar, with a minor difference: the values c^i_j and cc^i_j are not independent, as they sum to a coefficient that is known to the adversary. To handle this dependency, we partition the polynomial p into two parts: one involving only the variables $\zeta^i_{1,j}$ and the other involving only $\zeta^i_{2,j}$. If the term τ^i appears in p, it can similarly be split into two sub-components following the same partition. This separation ensures that each resulting sub-polynomial depends on independent random variables. We then apply the Schwartz–Zippel Lemma to each sub-polynomial individually, noting that each has total degree at most 4. Thus, each sub-polynomial evaluates to zero with probability at most $\frac{4}{q}$. By the union bound, the overall probability that p evaluates to zero remains negligible, and the simulation remains correct with overwhelming probability.
3) For the same reason as described in Theorem 1.
4) If the polynomial f is not empty, for $i, j \in [Q]$, $k, l \in [m \cdot t]$, $u \in [m], v \in [t]$ and let $k = (u-1)\cdot t + v$, f may contain the following types of monomials:
   - $\eta^i \cdot \tau^j = \alpha^i \cdot \beta^j \cdot u^j$
   - $\eta^i \cdot \zeta^j_{1,l} = \delta \cdot \alpha^i \cdot \beta^j \cdot bi^1_l \cdot c^j_l$
   - $\eta^i \cdot \zeta^j_{2,l} = \delta \cdot \alpha^i \cdot \beta^j \cdot bi^2_l \cdot cc^j_l$
   - $\omega^i_{1,k} \cdot \tau^j = \alpha^i \cdot b^1_k \cdot ((x^i_u)^v + r^1_k) \cdot \delta' \cdot \beta^j \cdot u^j$
   - $\omega^i_{1,k} \cdot \zeta^j_{1,l} = \alpha^i \cdot b^1_k \cdot ((x^i_u)^v + r^1_k) \cdot \beta^j \cdot bi^1_l \cdot c^j_l$
   - $\omega^i_{1,k} \cdot \zeta^j_{2,l} = \alpha^i \cdot b^1_k \cdot ((x^i_u)^v + r^1_k) \cdot \beta^j \cdot bi^2_l \cdot cc^j_l$
   - $\omega^i_{2,k} \cdot \tau^j = \alpha^i \cdot b^2_k \cdot ((x^i_u)^v + r^2_k) \cdot \delta' \cdot \beta^j \cdot u^j$
   - $\omega^i_{2,k} \cdot \zeta^j_{1,l} = \alpha^i \cdot b^2_k \cdot ((x^i_u)^v + r^2_k) \cdot \beta^j \cdot bi^1_l \cdot c^j_l$
   - $\omega^i_{2,k} \cdot \zeta^j_{2,l} = \alpha^i \cdot b^2_k \cdot ((x^i_u)^v + r^2_k) \cdot \beta^j \cdot bi^2_l \cdot cc^j_l$

   We show that when the polynomial f is non-empty, it is unlikely to evaluate to zero in the real game. The discussion here is somewhat subtle because not all variables are independently random. Specifically, we know that $b^1_i \cdot bi^1_i = b^2_i \cdot bi^2_i = 1$. We begin by analyzing the following two monomials where the formal variables should cancel out:

$$\omega^i_{1,k}\cdot\zeta^j_{1,k} = \alpha^i\cdot((x^i_u)^v + r^1_k)\cdot\beta^j\cdot c^j_k$$

$$\omega^i_{2,k}\cdot\zeta^j_{2,k} = \alpha^i\cdot((x^i_u)^v + r^2_k)\cdot\beta^j\cdot cc^j_k$$

   We observe that while these variables cancel out, their cancellation does not make the remaining terms identical, as r^1_k and r^2_k in the above expressions are independently random. Another issue to consider is that $c^i_k + cc^i_k$ is some value known to the adversary. However, these terms in the monomials above never appear with the same common set of other variables, preventing their randomness from being eliminated. From this analysis, we conclude that when the polynomial f is non-empty, it evaluates identically to zero for variables in $\mathcal{T}$. The remaining analysis involves classifying the monomials by their degrees and applying the Schwartz-Zippel lemma to bound the probability of the polynomial evaluating to zero. This step is straightforward, as we have established that even when b^1_i and bi^1_i, or b^2_i and bi^2_i, appear in the same monomial, the monomials retain unique randomness and cannot cancel each other out. Consequently, the polynomial remains defined over a set of independent random variables.

5) For the same reason as described in Theorem 1.

This completes the correctness proof of the simulator. □

Theorem 3. FlexDB *satisfies SIM-security, as defined in Definition 1, in the generic group model when only the equality test functionality is used, i.e., only the results of* ET *are revealed to the adversary.*

Proof. We start with defining the formal variables used in the proof.

k) 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 $\vec{x}_j^i$ as the j-th value in the input vector of the i-th Enc query for some $i \in [Q]$. Similarly, we define $\vec{w}_j^i$ as the j-th value in the input vector and pt^i as the desired sum in the i-th KeyGen query for some $i \in [Q]$. One important point to emphasize is that we *do not* consider $\vec{w}_j^i$ to be zero. If the weight was zero, that location in the vector could simply be ignored by excluding it from the position vector. We now define the following sets of variables:

$$\mathcal{T} = \left\{\alpha^i, \beta^i, pt^i\right\}_{i\in[Q]} \cup \left\{r_i\right\}_{i\in[m]} \cup \left\{b_i, bi_i\right\}_{i\in[m+1]} \cup \left\{x_j^i, w_j^i\right\}_{i\in[Q], j\in[m\cdot t]}$$

$$\mathcal{R} = \left\{\eta^i, \tau^i\right\}_{i\in[Q]} \cup \left\{\omega_j^i, \zeta_j^i\right\}_{i\in[Q], j\in[m]}$$

The specifications of these variables are given as follows.

- α^i: the random value α sampled in i-th Enc query
- β^i: the random value β sampled in i-th KeyGen query
- r_i: a random value
- b_i: a non-zero random value
- bi_i: the inverse of b_i, such that $b_i \cdot bi_i = 1$
- pt^i: we abuse the notation here to represent pt^i
- x_j^i: the value $\vec{x}_j^i$
- w_j^i: the value $\vec{w}_j^i$
- η^i: the value $\alpha^i \cdot b_{m+1}$
- τ^i: the value $\beta^i \cdot bi_{m+1} \cdot (pt^i + \sum_{j=1}^m r_j \cdot w_j^i)$
- ω_j^i: the value $\alpha^i \cdot b_j \cdot (x_j^i + r_j)$
- ζ_j^i: the value $\beta^i \cdot bi_j \cdot w_j^i$

l) 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 $\vec{x}^i \in \mathbb{Z}_q^m$, the simulator $\mathcal{S}$ receives as input a new map M_{ET} and updates its state. Then, $\mathcal{S}$ samples fresh handles for the formal variable η^i and each formal variable in the set $\{\omega_j^i\}_{j\in[m]}$, and returns these handles as the ciphertext.
- KeyGen: On input an integer $pt \in \mathbb{Z}_q$ and a vector $\vec{w}^i \in \mathbb{Z}_q^m$, the simulator $\mathcal{S}$ receives as input a new map M_{ET} and updates its state. Then, $\mathcal{S}$ samples fresh handles for the formal variable τ^i and each formal variable in the set $\{\zeta_j^i\}_{j\in[m]}$, and returns these handles as the ciphertext.
- Op: identical to the simulator described in Theorem 1.
- Pair: identical to the simulator described in Theorem 1.
- 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 in formal variables in $\mathcal{R}$.
  2) If $\ell \neq T$, outputs "non-zero".
  3) $\mathcal{S}$ decomposes $M_T[h]$ to the following polynomial:
  $$\sum_{i,j\in[Q]} c_{i,j} \cdot p_{i,j}\left(\eta^i, \tau^j, \left\{\omega_k^i, \zeta_k^j\right\}_{k\in[m]}\right) + f(\mathcal{T})$$
  where $p_{i,j}$ is defined as:
  $$\sum_{k=1}^m \left(\omega_k^i \cdot \zeta_k^j\right) - \eta^i \cdot \tau^j$$
  4) If the polynomial f is not empty, $\mathcal{S}$ returns "non-zero".
  5) Otherwise, for each polynomial $p_{i,j}$, the simulator $\mathcal{S}$ looks up its state for the output of $\mathsf{ET}(\vec{x}^i, \vec{y}^j, pt^j)$. If all these outputs are 1, the simulator returns "True" and "False" otherwise.

m) Correctness of the simulator: We show formally that the simulator constructed above is correct, efficient, 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) For the same reason as described in Theorem 1.
2) Consider the case where $\ell = 1$; by construction, the monomials the obtained by the adversary correspond to the responses from the encryption query. We can express the polynomial p in terms of variables in $\mathcal{T}$ as:
$$p = \sum_{i\in[Q]} \alpha^i \cdot \left(c_0 \cdot b_{m+1} + \sum_{j\in[m]} c_j \cdot b_j \cdot \left(x_j^i + r_j\right)\right),$$
where $c_0, \ldots, c_m$ are some constants. To justify this step, we show that in the real game, when the polynomial of formal variables is not identically zero, the constructed polynomial p is unlikely to evaluate to zero. Specifically, in the real game, regardless of the input choice of x_j^i, the value $(x_j^i + r_j)$ can be treated as a random variable. Applying the Schwartz-Zippel lemma, we conclude that this polynomial evaluates to zero with negligible probability. In the case where $\ell = 2$, the polynomial is expressed as:
$$p = \sum_{i\in[Q]} \beta^i \cdot \left(c_0 \cdot bi_{m+1} \cdot \left(pt^i + \sum_{j=1}^m r_j \cdot w_j^i\right) + \sum_{j\in[m]} c_j \cdot bi_j \cdot w_j^i\right).$$
Recall that we previously stated w_j^i cannot be zero. Given this, we can consider this polynomial as being defined over the random variables β^i, $r_i i \in [m]$, and $bi_i i \in [m+1]$. Regardless of the choice of w_j^i and pt^i, we can invoke the Schwartz-Zippel lemma to conclude that this polynomial evaluates to zero with negligible probability.
3) For the same reason as described in Theorem 1.

4) If the polynomial f is not empty, for $i, j \in [Q]$, $k, l \in [m \cdot t]$, $u \in [m], v \in [t]$ and let $k = (u-1) \cdot t + v$, f may contain the following types of monomials:
   - $\eta^i \cdot \tau^j = \alpha^i \cdot \beta^j \cdot (pt^j + \sum_{k'=1}^{m} r_{k'} \cdot w_{k'}^j)$
   - $\eta^i \cdot \zeta_l^j = \alpha^i \cdot b_{m+1} \cdot \beta^j \cdot bi_l \cdot w_l^j$
   - $\omega_k^i \cdot \tau^j = \alpha^i \cdot b_k \cdot (x_k^i + r_k) \cdot \beta^j \cdot bi_{m+1} \cdot (pt^j + \sum_{k'=1}^{m} r_{k'} \cdot w_{k'}^j)$
   - $\omega_k^i \cdot \zeta_l^j = \alpha^i \cdot b_k \cdot (x_k^i + r_k) \cdot \beta^j \cdot bi_l \cdot w_l^j$

   We show that when the polynomial f is non-empty, it can hardly evaluate to zero in the real game. First, we recall that each w_l^j is a non-zero value known to the adversary. Therefore, we can treat them as coefficients for the remaining terms. The orders of the variables are then as follows:
   - $\eta^i \cdot \tau^j$, which is of order 3
   - $\eta^i \cdot \zeta_l^j$, which is of order 4
   - $\omega_k^i \cdot \tau^j$, which is of order 6
   - $\omega_k^i \cdot \zeta_l^j$, which is of order 3 or 5

   We focus on the term $\omega_k^i \cdot \zeta_l^j$, and when $k = l$, we have $b_k \cdot bi_l = 1$, making this term of order 3. In this case, we can pair it with $\eta^i \cdot \tau^j$. Similar to the argument in the proof of Theorem 1, when f does not correspond to a correct decryption query, there are at least some left out values of $r_{k'} \cdot w_{k'}^j$ multiplied with $\alpha^i \cdot \beta^j$. There will also be the pt^j values and x_k^i values multiplied by some weights; we treat their sum as a constant. Regardless of the exact value of this constant term, we can invoke the Schwartz-Zippel lemma to show that the probability of f evaluating to zero is negligible when considering these two monomials. Then, the other monomials have different degrees, which are clearly distinct.
5) For the same reason as described in Theorem 1.

This completes the correctness proof of the simulator. □

Theorem 4. FlexDB *satisfies SIM-security, as defined in Definition 1, in the generic group model when only the range query functionality is used, i.e., only the results of* RQ *are revealed to the adversary.*

Proof. By design, the range-query construction is a special case of the multi-value filter (MF) scheme, where the encryption oracle receives input vectors consisting only of ones and zeros. The security proof for MF is fully generic: it imposes no restrictions on the message vector $\vec{x}$ or the key generation input vector $\vec{y}$. In particular, we emphasize that the doubling of polynomial lengths during key generation ensures that the presence of zero polynomials does not pose any issue in the analysis. Therefore, the range query construction directly inherits the security guarantees of the multi-value filter scheme without any modification. □

Theorem 5. FlexDB *satisfies SIM-security, as defined in Definition 1, in the generic group model when only the equi-join functionality is used, i.e., only the results of* EJ *are revealed to the adversary.*

Proof. Our equi-join construction extends the previously discussed functionalities by adding an additional component to each ciphertext: an encoding of the attribute to be compared across tables. For ease of discussion, we denote this attribute by x_0. Specifically, this ciphertext component is constructed as the sum of a *non-zero* hash value $\mathsf{H}(x_0)$ and a random product $\alpha \cdot r$. Since the randomness r used in this component is freshly sampled and independent of all other terms in the ciphertext, the security analyses for SF, MF, ET, and RQ remain unaffected. What remains is to show that the leakage from the underlying functionality is now hidden, and only the result of the equi-join is revealed. In other words, when an equi-join query fails, the scheme ensures that it is indistinguishable whether the failure was due to the comparison across tables not being satisfied or the underlying functionality not being satisfied.

We illustrate this using the MF ciphertext as an example. As shown in Algorithm 1, the key generation algorithm outputs $\mathsf{sk}_{\vec{y}} = (\mathsf{sk}_{\vec{y}}^1, \mathsf{sk}_{\vec{y}}^2, g_2^{\mu^{-1}\beta u})$. When used as the underlying functionality in the equi-join, the secret key remains largely unchanged, with the following modifications: ❶ A fresh randomness η is attached to the query to pair with the value $\mathsf{H}(x_0) + \alpha \cdot r$ in the ciphertext; ❷ The term u in the third component $g_2^{\mu^{-1}\beta u}$ is adjusted to cancel out the added randomness $\alpha \cdot r$ in the equi-join attribute. As a result, when the underlying functionality is satisfied, the i-th decryption query on this table returns the value $\eta^i \cdot \mathsf{H}(x_0)$. Since η^i is freshly sampled for each query, the output reveals no additional information about the underlying functionality. This is evident by considering the case where the functionality is not satisfied: the returned value becomes $\eta^i \cdot \mathsf{H}(x_0) + r'$, which has the same distribution as $\eta^i \cdot \mathsf{H}(x_0)$ regardless of the actual value of r'. Similarly, it is indistinguishable from $\eta \cdot \mathsf{H}(x_0')$ for a fresh randomness η and a different input x_0'. This shows that all three cases: when the equi-join condition holds, when it fails due to a comparison mismatch, and when it fails due to the underlying functionality, yield identically distributed outputs. Therefore, the security of the equi-join scheme is preserved. □

We have now shown that for all the schemes considered, FlexDB is SIM-secure when only one of the discussed functionalities is used. We therefore present the main theorem below.

Theorem 6. FlexDB *satisfies SIM-security, as defined in Definition 1, in the generic group model when any subset of the functionality* $\{\mathsf{SF}, \mathsf{MF}, \mathsf{ET}, \mathsf{RQ}, \mathsf{EJ}\}$ *is used, i.e., only the well-defined leakage* $\mathcal{L}$ *is revealed to the adversary.*

Proof. The proof strategy for this theorem follows the same structure as that of Theorem 5, with the key insight being that the use of multiple functionalities does not compromise the security of any individual scheme. Specifically, when schemes such as SF and MF are combined, the additional components introduced by each scheme are based on independent randomness and do not interfere with each other's security analysis. It is therefore crucial to show that when a query involving

multiple functionalities over different attributes fails to satisfy the overall predicate, the adversary gains no partial information with that query. This ensures that the combined scheme reveals only the result of the composed query and nothing about the intermediate outcomes of individual functionalities.

We observe a recurring structure in the ciphertexts and function keys across the supported functionalities. Each ciphertext includes an additional component of the form $\delta \cdot \alpha^i$, where δ is a fixed group element and α^i is a freshly sampled scalar. Correspondingly, each function key includes a term that encodes randomness designed to pair with $\delta \cdot \alpha^i$ during decryption, such that the randomness in the ciphertext is canceled out precisely when the query satisfies the corresponding functionality. In our FlexDB construction, when multiple attributes are enabled simultaneously, the same value $\delta \cdot \alpha^i$ is reused across the different functionalities. For each function key, the associated randomness is aggregated so that the pairing with $\delta \cdot \alpha^i$ cancels out if and only if all predicates corresponding to the enabled functionalities are satisfied. We stress that regardless of which predicate fails, some residual randomness remains in the pairing result. Due to the design of the zero-test protocol, as long as any residual randomness is present, the adversary's view is identical across all such cases. It learns only that the zero-test has failed, but gains no information about which specific attributes satisfied or failed their corresponding functionalities.

This design guarantees that decryption succeeds only when all functionality-specific conditions are met, and ensures that no partial information is leaked when any component of the conjunctive predicate fails. As a result, the composed scheme maintains the security of each individual functionality and ensures that the FlexDB reveals only the final result of the composed query, consistent with the desired leakage profile. □

C. Other FE Schemes

In this section, we describe the other FE schemes for other single functionalities in Alg. 2, Alg. 3, and Alg. 4. Besides, Alg. 5 corresponds to the approach for composable functionalities.

Algorithm 2 The FE scheme for range query

Client-side routines:

- $\mathsf{Setup}(1^\lambda, m_A, bl) \to (pp, msk)$
  a) Generate an asymmetric bilinear group $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, e)$ with generators $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$.
  b) Choose a random scalar $\mu \in \mathbb{Z}_q^*$, two random vectors $\vec{r}_1, \vec{r}_2 \in \mathbb{Z}_q^{m_A \cdot bl}$, and two random vectors $\vec{b}_1, \vec{b}_2 \in (\mathbb{Z}_q^*)^{m_A \cdot bl}$, where all entries are non-zero.
  c) Output the public parameters $pp := (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, e, g_1, g_2)$ to the server, and retain the master secret key $msk := (\mu, \vec{r}_1, \vec{r}_2, \vec{b}_1, \vec{b}_2)$ locally.
  For notational convenience, for $p \in \{1, 2\}$, $i \in [m_A]$, and $j \in [bl]$, let $r_{i,j}^p$ and $b_{i,j}^p$ denote the $((i-1) \cdot bl + j)$-th entries of $\vec{r}_p$ and $\vec{b}_p$, respectively.
- $\mathsf{Encrypt}(msk, \vec{x}) \to \mathsf{ct}_{\vec{x}}$
  a) Sample a random scalar $\alpha \in \mathbb{Z}_q^*$.
  b) For simplicity, let $x_{i,j}$ corresponds to the j-bit of x_i.
  c) Construct ciphertexts $\mathsf{ct}_{\vec{x}} = (\mathsf{ct}_{\vec{x}}^1, \mathsf{ct}_{\vec{x}}^2, g_1^{\alpha\mu})$, where for each $p \in \{1, 2\}$, define:

$$\mathsf{ct}_{\vec{x}}^p := (\mathsf{ct}_1^p, \ldots, \mathsf{ct}_{m_A}^p), \quad \text{and} \quad \forall i \in [bl], \quad \mathsf{ct}_i^p := \left(g_1^{\alpha b_{i,1}^p (x_{i,1} + r_{i,1}^p)}, \ldots, g_1^{\alpha b_{i,bl}^p (x_{i,bl} + r_{i,bl}^p)} \right).$$

- $\mathsf{KeyGen}(msk, \vec{y}) \to \mathsf{sk}_{\vec{y}}$
  a) For the j-th bit of attribute A_i, construct the filter polynomial $f_{i,j}(x)$, and decompose $f_{i,j}(x)$ into two polynomials $f_{i,j}^1(x)$ and $f_{i,j}^2(x)$ with non-zero coefficients.
  b) Sample a random scalar $\beta \in \mathbb{Z}_q^*$ and a random vector $\vec{\gamma} = (\gamma_1, \ldots, \gamma_{m_A \cdot bl}) \in \mathbb{Z}_q^{m_A \cdot bl}$ to mask the per-attribute evaluations.
  c) Let $c_{(i,j,d)}^1$ and $c_{(i,j,d)}^2$ denote the d-th coefficient of $f_{i,j}^1(x)$ and $f_{i,j}^2(x)$, respectively, for $d \in \{0, 1\}$.
  d) Define the correction term:

$$u := \sum_{i=1}^{m_A} \sum_{j=1}^{bl} \gamma_{i,j} (c_{i,j,0}^1 + c_{i,j,0}^2) - \sum_{i=1}^{m_A} \sum_{j=1}^{bl} \gamma_{i,j} (r_{i,j}^1 c_{(i,j,1)}^1 + r_{i,j}^2 c_{(i,j,1)}^2).$$

  e) Output the decryption key as $\mathsf{sk}_{\vec{y}} = (\mathsf{sk}_{\vec{y}}^1, \mathsf{sk}_{\vec{y}}^2, g_2^{\mu^{-1}\beta u})$, where for each $p \in \{1, 2\}$,

$$\mathsf{sk}_{\vec{y}}^p = (\mathsf{sk}_1^p, \ldots, \mathsf{sk}_{m_A}^p), \quad \text{and} \quad \forall i \in [m_A], \quad \mathsf{sk}_i^p := \left(g_2^{\beta\gamma_i (b_{i,1}^p)^{-1} c_{(i,j,1)}^p}, \ldots, g_2^{\beta\gamma_i (b_{i,bl}^p)^{-1} c_{(i,bl,1)}^p} \right).$$

Server-side routine:

- $\mathsf{Decrypt}(\mathsf{ct}_{\vec{x}}, \mathsf{sk}_{\vec{y}}) \to \{0, 1\}$
  a) Compute the pairing product:

$$\mathsf{res} = e(\mathsf{ct}_{\vec{x}}^1, \mathsf{sk}_{\vec{y}}^1) \cdot e(\mathsf{ct}_{\vec{x}}^2, \mathsf{sk}_{\vec{y}}^2) \cdot e(g_1^{\alpha\mu}, g_2^{\mu^{-1}\beta u}).$$

  b) Output 1 if $\mathsf{res} = e(g_1, g_2)^0$, i.e., the identity element in $\mathbb{G}_T$; otherwise, output 0.

Algorithm 3 The FE scheme for equality test

Client-side routines:

- $\mathsf{Setup}(1^\lambda, m_A) \to (pp, msk)$
  a) Generate an asymmetric bilinear group $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, e)$ with generators $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$.
  b) Choose a random scalar $\mu \in \mathbb{Z}_q^*$, a random vector $\vec{r} \in \mathbb{Z}_q^{m_A}$, and a random vectors $\vec{b} \in (\mathbb{Z}_q^*)^{m_A}$, where all entries are non-zero.
  c) Output the public parameters $pp := (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, e, g_1, g_2)$ to the server, and retain the master secret key $msk := (\mu, \vec{r}, \vec{b})$ locally.
- $\mathsf{Encrypt}(msk, \vec{x}) \to \mathsf{ct}_{\vec{x}}$
  a) Sample a random scalar $\alpha \in \mathbb{Z}_q^*$.
  b) Construct ciphertexts $\mathsf{ct}_{\vec{x}} = (\mathsf{ct}_1, \mathsf{ct}_2, ..., \mathsf{ct}_{m_A}, g_1^{\alpha\mu})$, where for each $i \in [m]$, $\mathsf{ct}_i = g_1^{\alpha b_i (x_i + r_i)}$.
- $\mathsf{KeyGen}(msk, \vec{y}) \to \mathsf{sk}_{\vec{y}}$
  a) generate the weight vector $\vec{w} = \{w_1, ..., w_{m_A}\}$ and target number pt.
  b) Sample a random scalar $\beta \in \mathbb{Z}_q^*$.
  c) Define the correction term:
  $$u := -\sum_{i=1}^{m_A} w_i r_i - pt.$$
  d) Output the decryption key as $\mathsf{sk}_{\vec{y}} = (\mathsf{sk}_1, \mathsf{sk}_2, ..., \mathsf{sk}_{m_A}, g_1^{\mu^{-1}\beta u})$, where for each $i \in [m_A], \mathsf{sk}_i = g_2^{\beta w_i b_i^{-1}}$.

Server-side routine:

- $\mathsf{Decrypt}(\mathsf{ct}_{\vec{x}}, \mathsf{sk}_{\vec{y}}) \to \{0, 1\}$
  a) Compute the pairing product:
  $$\mathsf{res} = e(g_1^{\alpha\mu}, g_2^{\mu^{-1}\beta u}) \cdot \prod_{i=1}^{m_A} e(\mathsf{ct}_i, \mathsf{sk}_i).$$
  b) Output 1 if $\mathsf{res} = e(g_1, g_2)^0$, i.e., the identity element in $\mathbb{G}_T$; otherwise, output 0.

Algorithm 4 The FE scheme for equi-join with single/multi-value filter on attribute A_i in table $\mathcal{T}_A$

Client-side routines:

- $\mathsf{Setup}(1^\lambda, m_A, t) \to (pp, msk)$
  a) Generate an asymmetric bilinear group $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, e)$ with generators $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$.
  b) Generate a private hash function denoted by $\mathsf{H} : \mathbb{Z}_q \to \mathbb{Z}_q$ and another random number $\tau \in \mathbb{Z}_q$
  c) Choose a random scalar $\mu \in \mathbb{Z}_q^*$, two random vectors $\vec{r}_1, \vec{r}_2 \in \mathbb{Z}_q^{t \cdot m_A}$, and two random vectors $\vec{b}_1, \vec{b}_2 \in (\mathbb{Z}_q^*)^{t \cdot m_A}$, where all entries are non-zero.
  d) Output the public parameters $pp := (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, e, g_1, g_2)$ to the server, and retain the master secret key $msk := (\mu, \vec{r}_1, \vec{r}_2, \vec{b}_1, \vec{b}_2)$ locally.
  For notational convenience, for $p \in \{1, 2\}$, $i \in [m_A]$, and $j \in [t]$, let $r_{i,j}^p$ and $b_{i,j}^p$ denote the $((i-1) \cdot t + j)$-th entries of $\vec{r}_p$ and $\vec{b}_p$, respectively.
- $\mathsf{Encrypt}(msk, \vec{x}) \to \mathsf{ct}_{\vec{x}}$
  a) Sample a random scalar $\alpha \in \mathbb{Z}_q^*$.
  b) Construct ciphertexts $\mathsf{ct}_{\vec{x}} = (g_1^{\mathsf{H}(x_i)+\alpha\tau}, \mathsf{ct}_{\vec{x}}^1, \mathsf{ct}_{\vec{x}}^2, g_1^{\alpha\mu})$, where for each $p \in \{1, 2\}$, define:

$$\mathsf{ct}_{\vec{x}}^p := (\mathsf{ct}_1^p, \ldots, \mathsf{ct}_{m_A}^p), \quad \text{and} \quad \forall i \in [m_A], \quad \mathsf{ct}_i^p := \left(g_1^{\alpha b_{i,1}^p [(x_i)^1 + r_{i,1}^p]}, \ldots, g_1^{\alpha b_{i,t}^p [(x_i)^t + r_{i,t}^p]} \right).$$

- $\mathsf{KeyGen}(msk, \vec{y}) \to \mathsf{sk}_{\vec{y}}$
  a) For each attribute A_i, construct the filter polynomial $f_i(x)$, and decompose $f_i(x)$ into two polynomials $f_i^1(x)$ and $f_i^2(x)$ with non-zero coefficients.
  b) Sample a random scalar $\beta \in \mathbb{Z}_q^*$ and a random vector $\vec{\gamma} = (\gamma_1, \ldots, \gamma_{m_A}) \in \mathbb{Z}_q^{m_A}$ to mask the per-attribute evaluations.
  c) Let $c_{i,d}^1$ and $c_{i,d}^2$ denote the d-th coefficient of $f_i^1(x)$ and $f_i^2(x)$, respectively, for $d \in \{0, 1, \ldots, t\}$.
  d) Define the correction term:

$$u := \sum_{i=1}^{m_A} \gamma_i (c_{i,0}^1 + c_{i,0}^2) - \sum_{i=1}^{m_A} \sum_{j=1}^{t} \gamma_i (r_{i,j}^1 c_{i,j}^1 + r_{i,j}^2 c_{i,j}^2) - \eta\alpha\tau.$$

  e) Output the decryption key as $\mathsf{sk}_{\vec{y}} = (g_2^\eta, \mathsf{sk}_{\vec{y}}^1, \mathsf{sk}_{\vec{y}}^2, g_2^{\mu^{-1}\beta u})$, where for each $p \in \{1, 2\}$,

$$\mathsf{sk}_{\vec{y}}^p = (\mathsf{sk}_1^p, \ldots, \mathsf{sk}_{m_A}^p), \quad \text{and} \quad \forall i \in [m_A], \quad \mathsf{sk}_i^p := \left(g_2^{\beta\gamma_i (b_{i,1}^p)^{-1} c_{i,1}^p}, \ldots, g_2^{\beta\gamma_i (b_{i,t}^p)^{-1} c_{i,t}^p} \right).$$

Server-side routine:

- $\mathsf{Decrypt}(\mathsf{ct}_{\vec{x}}, \mathsf{sk}_{\vec{y}}) \to \{0, 1\}$
  a) Compute the pairing product:

$$\mathsf{res} = e(g_1^{\mathsf{H}(x_i)+\alpha\tau}, g_2^\eta) \cdot e(\mathsf{ct}_{\vec{x}}^1, \mathsf{sk}_{\vec{y}}^1) \cdot e(\mathsf{ct}_{\vec{x}}^2, \mathsf{sk}_{\vec{y}}^2) \cdot e(g_1^{\alpha\mu}, g_2^{\mu^{-1}\beta u}).$$

  b) Output 1 if $\mathsf{res} = e(g_1, g_2)^0$, i.e., the identity element in $\mathbb{G}_T$; otherwise, output 0.

Algorithm 5 The FE scheme for composable conjunctive queries towards table $\mathcal{T}_A$

Client-side routines:

- Combine all FE schemes to execute a query Q:
  a) Suppose the server stores ciphertexts $(\mathsf{ct}_{\vec{x}}^{mf}, g_1^{\alpha\mu_1})$, $(\mathsf{ct}_{\vec{x}}^{rq}, g_1^{\alpha\mu_2})$, $(\mathsf{ct}_{\vec{x}}^{et}, g_1^{\alpha\mu_3})$ for multi-value filter, range query, and equality test, respectively.
  b) WLOG, we assume the above ciphertexts aims to different attributes, e.g., they separately cover attributes

$$\{A_1, ..., A_{m_{mf}}\},\ \{A_{m_{mf}+1}, ..., A_{m_{rq}}\},\ \text{ and }\ \{A_{m_{rq}+1}, ..., A_{m_{et}}\}.$$

  Also, we additionally assume the attribute $A_{m_{et}+1}$ is encrypted as ciphertext $g_1^{\mathsf{H}(x_{m_{et}+1})+\alpha\tau}$ for equi-join
  c) Then the client firstly separately generate secret keys for these ciphertexts according to the functionalities required by query Q. Denote the corresponding secret keys as

$$(\mathsf{sk}_{\vec{y}}^{mf}, g_2^{\mu_1^{-1}\beta u_1}),\ (\mathsf{sk}_{\vec{y}}^{rq}, g_2^{\mu_2^{-1}\beta u_2}),\ \text{ and }\ (\mathsf{sk}_{\vec{y}}^{et}, g_2^{\mu_3^{-1}\beta u_3}).$$

  The secret key for equi-join should be g_2^{η}.
  d) Output the final secret key as

$$(\mathsf{sk}_{\vec{y}}^{mf}, g_2^{\mu_1^{-1}\beta(u_1+r_1)}, \mathsf{sk}_{\vec{y}}^{rq}, g_2^{\mu_2^{-1}\beta(u_2+r_2)}, \mathsf{sk}_{\vec{y}}^{et}, g_2^{\mu_3^{-1}\beta(u_3-r_1-r_2)}, g_2^{\eta}).$$

  Please note the randomness in equi-join can be canceled in the same way as shown in Alg. 4, which is implied by the value of u_3.

Server-side routine:

- $\mathsf{Decrypt}(\mathsf{ct}_{\vec{x}}, \mathsf{sk}_{\vec{y}}) \to \{0, 1\}$
  a) Compute the pairing product:

$$e(\mathsf{ct}_{\vec{x}}^{mf}, \mathsf{sk}_{\vec{y}}^{mf}) \cdot e(g_1^{\alpha\mu_1}, g_2^{\mu_1^{-1}\beta(u_1+r_1)}) \cdot e(\mathsf{ct}_{\vec{x}}^{rq}, \mathsf{sk}_{\vec{y}}^{rq}) \cdot e(g_1^{\alpha\mu_2}, g_2^{\mu_2^{-1}\beta(u_2+r_2)}) \cdot e(\mathsf{ct}_{\vec{x}}^{et}, \mathsf{sk}_{\vec{y}}^{et}) \cdot e(g_1^{\alpha\mu_3}, g_2^{\mu_3^{-1}\beta(u_3-r_1-r_2)}).$$

  And then compute the product between the above result and $e(g_1^{\mathsf{H}(x_{m_{et}+1})+\alpha\tau}, g_2^{\eta})$ as res.
  b) Output res as the final ciphertext. It is the deterministic encryption of $x_{m_{et}+1}$ if the record $\vec{x}$ satisfies all conditions of query Q.