HE^{3}DB: An Efficient and Elastic Encrypted Database Via Arithmetic-And-Logic Fully Homomorphic Encryption

Song Bian*
sbian@buaa.edu.cn
Beihang University
Beijing, China

Zhou Zhang*
zhouzhang@buaa.edu.cn
Beihang University
Beijing, China

Haowen Pan
panhaowen@buaa.edu.cn
Beihang University
Beijing, China

Ran Mao
maoran_44@buaa.edu.cn
Beihang University
Beijing, China

Zian Zhao
zhaozian@buaa.edu.cn
Beihang University
Beijing, China

Yier Jin
jinyier@gmail.com
University of Science and Technology of China
Hefei, Anhui, China

Zhenyu Guan†
guanzhenyu@buaa.edu.cn
Beihang University
Beijing, China

ABSTRACT

As concerns are increasingly raised about data privacy, encrypted database management system (DBMS) based on fully homomorphic encryption (FHE) attracts increasing research attention, as FHE permits DBMS to be directly outsourced to cloud servers without revealing any plaintext data. However, the real-world deployment of FHE-based DBMS faces two main challenges: i) high computational latency, and ii) lack of elastic query processing capability, both of which stem from the inherent limitations of the underlying FHE operators. Here, we introduce HE^{3}DB, a fully homomorphically encrypted, efficient and elastic DBMS framework based on a new FHE infrastructure. By proposing and integrating new arithmetic and logic homomorphic operators, we devise fast and high-precision homomorphic comparison and aggregation algorithms that enable a variety of SQL queries to be applied over FHE ciphertexts, e.g., compound filter-aggregation, sorting, grouping, and joining. In addition, in contrast to existing encrypted DBMS that only support aggregated information retrieval, our framework permits further server-side elastic analytical processing over the queried FHE ciphertexts, such as private decision tree evaluation. In the experiment, we rigorously study the efficiency and flexibility of HE^{3}DB. We show that, compared to the state-of-the-art techniques, HE^{3}DB can homomorphically evaluate end-to-end SQL queries as much as 41×–299× faster than the state-of-the-art solution, completing a TPC-H query over a 16-bit 10K-row database within 241 seconds.

CCS CONCEPTS

• **Security and privacy** → **Cryptography**; **Management and querying of encrypted data**.

KEYWORDS

Fully Homomorphic Encryption; Secure Database Outsourcing

*Both authors contributed equally to this research.
†Corresponding author.

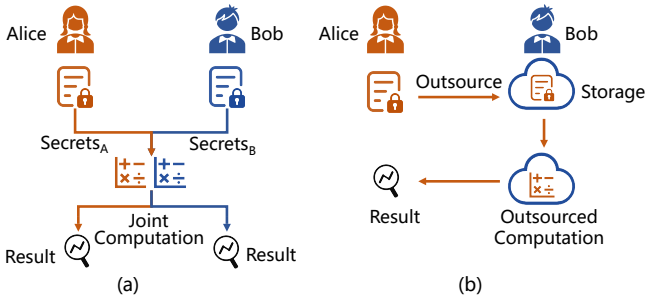


Figure 1: The conceptual illustrations of (a) a secure multi-party query evaluation (SQE) scheme and (b) a secure database outsourcing (SDO) scheme.

1 INTRODUCTION

Outsourcing the storage of and computation over private data to cloud service providers [8, 9, 34, 82] is a common practice for personal users, enterprises, and even government institutions to lift the burden of large-scale data management. However, in many cases, the outsourced data contain sensitive information, and data owners are concerned with their private data either directly stolen by the curious cloud service providers [76] or leaked through the endless data breaches [22, 99]. As a result, data owners and cloud service providers are interested in building encrypted database infrastructures [87], where the entire database is stored in ciphertext on the cloud. However, due to the inherent complexity of the design of multi-party secure protocols, the fundamental challenge facing encrypted databases is how to *efficiently* and *elastically* process queries on the encrypted data outsourced to the cloud while preventing various side-channel leakages [23, 50, 68]. In particular, being *elastic* means that the database engine can handle both diverse types of SQL queries and various data analysis algorithms.

To tackle the above challenges, existing works propose different designs of encrypted databases that utilize tools across multiple fields of research [17, 42, 45, 54, 74, 88, 90, 93, 106, 108]. We categorize existing designs of encrypted databases into two types of protocol models: secure multi-party query evaluation (SQE) and secure database outsourcing (SDO). As illustrated in Figure 1, we assume that no party in an SQE protocol can recover all of the secrets only by its own. In contrast, SDO refers to the case where one of the participating parties (e.g., the one who outsources the data) owns all of the private data. Thus, even though both SQE and SDO prefer lower protocol overheads, parties involved in SQE are willing to pay more computation and communication costs in the exchange of being able to compute over unknown secrets. Meanwhile, for SDO, client-side workloads need to be strictly smaller than that without the protocol, otherwise the client can simply perform the computations locally on its own.

With the optimal round complexity [83] and low online communication, fully homomorphic encryption (FHE) emerges as a powerful cryptographic primitive for SDO. While FHE was deemed too slow to DB applications [95], recent advances in FHE cryptography [10, 26, 53, 77] have significantly improved the efficiency and operator usability of FHE, leading to a series of recent works exploring the practical feasibility of FHE in DB-related tasks [36, 77, 93]. For instance, the very recent work HEDA [93] employs FHE schemes including the BFV [20, 46] and TFHE [32] to accelerate filter-aggregation queries over fully homomorphically encrypted database. Unfortunately, in addition to the relatively slow evaluation speed, HEDA suffers from limited query expressiveness as the exact algorithmic constructions for the evaluations of many common SQL statements, such as `MIN`, `MAX`, `JOIN` and `ORDER BY`, are left as open questions [93]. Therefore, the main challenge faced by FHE-based SDO becomes the following question: can we design an encrypted database scheme over FHE that achieves both fast query evaluation and elastic data processing at the same time?

1.1 Our Contribution

In this work, we propose HE^{3}DB, an FHE-based encrypted database management system (DBMS) for fast and elastic SDO. We observe that most existing FHE algorithms, whether specifically designed for SDO [93] or not [30, 77], do not fully capture the important properties of encrypted databases, and often fail to meet the key requirements of SDO. Contrarily, based on the idea of application-framework co-design, we propose a tailored FHE infrastructure consisting of DB-specific FHE operators that leverage the numerical and operational characteristics specific to encrypted databases. As it turns out, we are able to achieve higher accuracy, faster evaluation speed, richer query expressiveness and standard security all at once. In addition, to the best of our knowledge, HE^{3}DB is the first FHE-based DB framework that enables a client to execute outsourced data analysis algorithms on the queried ciphertext results without additional rounds of client-server interactions.

The main contributions of this work are summarized as follows.

- **An FHE Infrastructure for Encrypted DBMS**: We develop novel homomorphic filtering, aggregation and conversion algorithms that are highly composable and specifically designed for various types of DB operands and complex SQL operations. As a result, HE^{3}DB supports one of the most complete sets of SQL statements for encrypted databases, along with online data analysis capability.
- **High-Precision Homomorphic Comparison**: We observe that, the key FHE operator for encrypted database is the high-precision homomorphic comparison operator. However, the result of a homomorphic comparison is always binary. Based on the above insight, we devise a new shift-and-subtract strategy to homomorphically remove the lower-bit noises from the comparison results, and develop a high-precision homomorphic comparison operator `HomComp`. We can evaluate homomorphic comparison with up to 32-bit precision without using bit-by-bit encryption.
- **Homomorphic Aggregations over Filtered Results**: Based on the `HomComp` operator, we are able to propose fast algorithms to perform both arithmetic (e.g., `SUM`, `AVG`) and logic (e.g., `MIN`, `MAX`) aggregations on the filtered results. Our arithmetic and logic aggregation can be as much as 58× and 34× faster than existing works, respectively.
- **Thorough Experiments**: We show that HE^{3}DB achieves 7×–113× faster homomorphic filtering based on our homomorphic comparison operator `HomComp`, 24×–58× faster `COUNT`, 34× faster `ORDER BY`, and 12×–326× faster private decision tree evaluation using the proposed arithmetic and logic aggregation algorithms, all against the best-performing implementations. In addition, we show that HE^{3}DB is on average 41×–299× faster than the state-of-the-art (SOTA) FHE-based DBMS over end-to-end SQL benchmarks. Our code is publicly available[1].

1.2 Related Works

Here, we introduce existing works based on three main groups: trust-execution-environment-(TEE)-based SQE and SDO, multi-party-computation-(MPC)-based SQE, and MPC-based SDO. Note that, here, we consider FHE to be one particular type of general MPC technique.

1.2.1 TEE-based SDO and SQE. TEE provides a secure hardware enclave that divides the system into two entities: the trusted TEE enclave that can access and compute over all the sensitive data, and the outside rich execution environment (REE) that is exposed to a strong adversary capable of monitoring and manipulating the software and hardware systems. The objective is to protect the confidentiality and integrity of the data and algorithms within TEE from the external adversary. TEE-based solutions can easily realize both SDO and SQE simultaneously [5, 7, 12, 13, 45, 70, 81, 90, 101, 105, 109]. For example, although earlier techniques are constrained by the memory hierarchy of TEE [7, 13, 90], newer TEE architectures have largely abandoned memory integrity verification, thereby removing the limit on memory size [47, 65]. Subsequently, recent TEE-based techniques [5, 105] have demonstrated that advanced DBMS can be built over TEE with advanced access control mechanisms. Unfortunately, TEE-based encrypted databases face two fundamental challenges. First, the hardware architectural design of TEE can change significantly from one to the other, causing security vulnerabilities to be

[1] https://github.com/zhouzhangwalker/HE3DB

introduced. For example, as mentioned in [105], encrypted database based on Intel SGX [39] can leak access pattern [45], while that based on application-specific TEE enclaves [98] do not. Second, as a common critique for most privacy-preserving computing tasks based on TEE, it can be discouraging for applications that require a high level of data confidentiality to adopt a hardware root of trust.

1.2.2 MPC-based SQE. Most MPC-based encrypted databases focus on developing methods to evaluate queries over contents that are owned by different participating parties with security against *semi-honest* or *malicious* adversaries e.g., [15, 37, 42, 43, 48, 63, 74, 88, 103, 104, 106]. Due to the inherent complexity of SQE, MPC protocols vary significantly in terms of their security assumptions, security properties, efficiency, and usability. The main consequence of such a design complexity is that protocols tend to be incompatible with each other, and it can be very hard to integrate such protocols into a single database framework. For instance, the recent encrypted DB protocol Waldo [42] does not support certain SQL operators such as `GROUP BY` and `JOIN` [42]. Although many MPC-based protocols, such as Secure Yannakakis [106], Secrecy [74] and Senate [88], do support secure `GROUP BY` and `JOIN` operations, these protocols cannot be easily combined together because Secure Yannakakis [106] is a two-party protocol that utilizes standard secret sharing, while Waldo and Secrecy assume an honest-majority three-party setting built over functional secret sharing [19] and replicated secret sharing [6]. In addition, since secrets are defined to be distributed over the participating parties, MPC-based techniques generally require a relatively large amount of communication bandwidth and interaction rounds [42]. For example, with proper GPU acceleration [107], communication can be 6×−1000× slower than the computation in the state-of-the-art MPC protocols, becoming the main latency bottleneck.

1.2.3 MPC-based SDO. Different from SQE, methods for SDO explore how to securely and efficiently *outsource* a database to an untrusted server [54, 71, 85, 87, 89, 93], assuming that the server can be *semi-honest* or *malicious* . Existing MPC-based SDO schemes generally adopt more than one type of MPC primitives, including ORAM [45], SE [54, 85, 87], and homomorphic encryption [45, 54, 89]. However, as mentioned earlier, protocols based on SE [24, 41] introduces side-channel leakages [23, 50–52, 64]. Meanwhile, albeit ORAM [97] achieves poly-logarithmic complexity in oblivious storing and retrieving data items, executing multi-dimensional operations over ORAM, such as sorting, incurs extensive communications and interactions between the data owner and the cloud server that can be prohibitive in data outsourcing applications. Finally, most MPC-based SDO adopt some sort of homomorphic encryption, e.g., based on partially homomorphic encryption [54, 85, 87, 89] or leveled homomorphic encryption [71]. Unfortunately, without additional MPC primitives, both partially and leveled homomorphic encryption result in limited query expressiveness and data analysis capability, either due to complex query encoding schemes or *a-priori* bounds over the encryption parameters.

1.2.4 FHE-based SQE and SDO. Different from partially and leveled homomorphic encryption, FHE [20, 21, 29, 32, 44, 49, 96] is capable of evaluating complex queries over both outsourced and two-party secure databases[2] under a *semi-honest* adversary[3]. Since FHE was extremely computationally expensive in the earlier days [89, 95], protocols over FHE often only implement a small subset of database-related operations, e.g., supporting PIR [108] or keyword search [31, 69] only. Recently, motivated by the notable progresses made in the development of fast and flexible FHE primitives [73, 77], new FHE-based SDO schemes are proposed, e.g., [93]. Nonetheless, in addition to the slower performance, the query expressiveness of [93] can be rather restricted, as the scheme does not have support for common SQL statements such as `GROUP BY` and `ORDER BY`. Hence, the key objective of this work is to significantly boost the computational performance of FHE-based encrypted database while retaining the rich query expressiveness and unbounded data analysis capability of modern DBMS.

2 CRYPTOGRAPHIC PRELIMINARIES

In this section, we summarize some of the important notations for FHE ciphertext and FHE operators used throughout this work in Section 2.1, and Section 2.2, respectively.

2.1 Homomorphic Encryption

For FHE notations, we use λ to denote the security parameter and p for the plaintext modulus. We use $q/Q/Q'$ to indicate different sizes of ciphertext modulus (generally $Q > q$), and $n/N/N'$ specify lattice dimensions (generally $N > n$). $\mathbb{Z}_q$ refers to the set of integers modulo q and $\mathbb{Z}_q[x]$ depicts the set of polynomials with coefficients in $\mathbb{Z}_q$. We use R_N and $R_{N,Q}$ to denote $\mathbb{Z}[X]/(X^N+1)$ and $\mathbb{Z}_Q[X]/(X^N+1)$, respectively, for some ciphertext modulus Q and polynomial degree N. Throughout this paper, we use bold lowercase letters (e.g., $\mathbf{a}$) for vectors, tilde lowercase letters (e.g., $\tilde{a}$) for polynomials, and bold uppercase letters (e.g., $\mathbf{A}$) for matrices. For a complete list of notations, please refer to Table 1.

In this work, we mainly adopt the cryptographic constructions and techniques [16, 18, 28, 33, 55, 57, 58, 62, 73, 77] developed along both the CKKS [29] and TFHE [32] lanes of FHE schemes. Therefore, similar to [93], we use ciphertexts that are based on both the learning with error (LWE) and ring learning with error (RLWE) hardness problems [78, 91]. Moreover, we also make use of the ring variant of the GSW encryption scheme proposed in [49]. The concrete definitions are as follows.

- $\mathsf{LWE}_{\mathbf{s}}^{n,q}(m)$: The LWE ciphertext. Here, we use a symmetric version of the LWE encryption function encrypting a single integer message $m \in \mathbb{Z}_p$ under the secret key $\mathbf{s} \in \mathbb{Z}_q^n$ is given as:

$$\mathsf{LWE}_{\mathbf{s}}^{n,q}(m) = (b, \mathbf{a}) = (< -\mathbf{a}, \mathbf{s} > + \Delta m + e, \mathbf{a}).$$

where $\mathbf{a} \in \mathbb{Z}_q^n$ is chosen uniformly at random, the noise e is sampled from some distribution χ_{noise}, and $\Delta = \lfloor \frac{q}{p} \rceil$ is a scaling factor to protect the least significant bits of the message from the noises.

[2]We note that SQE over single-key FHE is limited to a two-party setting, while more complicated multi-party protocols require further advance in the development of multi-key FHE schemes [4, 83, 86].

[3]It is worth noting that achieving malicious security in an FHE-based approach requires supplementary cryptographic tools such as zero-knowledge proofs (ZKP) [100] or oblivious pseudorandom function (OPRF) [27]. .

Table 1: Summary of Notations

Notation	Description
$\mathcal{D}$	The database
$\mathcal{T}$	The data tables in the database
$\lvert\mathcal{T}\rvert_{\text{row}}/\lvert\mathcal{T}\rvert_{\text{col}}$	The number of rows/columns in table $\mathcal{T}$
$\mathcal{Q}$	The SQL query.
$\lvert\mathcal{Q}\rvert$	The number of predicates in query $\mathcal{Q}$.
$\mathbb{W}$	The parameters in data analysis model
λ	The security parameter
p	The plaintext modulus
$q/Q/Q'$	The ciphertext modulus for LWE/RLWE/RGSW
$n/N/N'$	The lattice dimension for LWE/RLWE/RGSW
l	The number of RLWE in RGSW
$\mathbb{Z}_q^n$	The set of n-vectors over $\mathbb{Z}_q$
$R_{N,Q}$	The cyclotomic ring $\mathbb{Z}_{\mathbb{Q}}[X]/(X^N+1)$
χ	The noise distribution
Δ	The scaling factor
$\mathbf{a}/\tilde{a}$	An element in vector domain/polynomial ring
$\mathbf{A}$	An element in matrix domain
$\mathsf{LWE}_{\mathbf{s}}^{n,q}(m)$	An LWE ciphertexts encrypting m with parameters (n,q) and secret $\mathbf{s}$
$\mathsf{RLWE}_{\tilde{s}}^{N,Q}(\tilde{m})$	An RLWE ciphertexts encrypting $\tilde{m}$ with parameters (N,Q) and secret $\tilde{s}$
$\mathsf{RGSW}_{\tilde{\mathbf{s}}}^{N',Q'}(\mathbf{m})$	An RGSW ciphertext encrypting $\mathbf{m}$ with parameters (N',Q') and secret $\tilde{\mathbf{s}}$
$\diamond$	Homomorphic matrix-vector multiplication
CMUX	Homomorphic selector [32]
PBS	Programmable bootstrapping [33]
HomGate	Homomorphic gate [32, 44]
RLWEtoLWEs	Converting RLWE to LWEs (e.g., sample extract index [32])
LWEstoRLWE	Converting LWEs to RLWE (e.g., repack [26, 77])
LWEtoRGSW	Converting LWE to RGSW (e.g., circuit bootstrapping [32])

• $\mathsf{RLWE}_{\tilde{s}}^{N,Q}(\tilde{m})$: The RLWE ciphertext. An RLWE ciphertext is defined as

$$\mathsf{RLWE}_{\tilde{s}}^{N,Q}(\tilde{m}) = (\tilde{b},\tilde{a}) = (-\tilde{a}\cdot\tilde{s}+\Delta\tilde{m}+\tilde{e},\tilde{a}).$$

for a messages $\tilde{m}\in R_{N,p}$ encrypted under the secret key $\tilde{s}\in R_{N,Q}$. Here, $\tilde{a}\in R_{N,Q}$ is chosen uniformly at random and $\Delta=\lfloor\frac{Q}{p}\rceil$ is the scaling factor.

• $\mathsf{RGSW}_{\tilde{s}}^{N',Q'}(\tilde{m})$: The RGSW ciphertext. First introduced in [49], given a gadget vector $\mathbf{g}=(g_0,g_1,...,g_{l-1})\in\mathbb{Z}_{Q'}^l$, the RGSW encryption of a message $\tilde{m}\in R_{N',p}$ under the secret key $\tilde{s}\in R_{N',Q'}$ is:

$$\mathsf{RGSW}_{\tilde{s}}^{N',Q'}(\tilde{m}) = \mathbf{Z}+\tilde{m}\mathbf{G} = (\mathbf{B},\mathbf{A})\in R_{N',Q'}^{2l\times 2}.$$

where $\mathbf{Z}\in R_{N',Q'}^{2l\times 2}$ contains $2l$ RLWE ciphertexts encrypting zeroes. $\mathbf{G}$ is defined as $\mathbf{G}=\mathbf{I}_2\otimes\mathbf{g}$, where $\mathbf{I}_2$ is the identity matrix of size 2 and $\otimes$ refers to the Kronecker product [60] between two matrices. Roughly speaking, an RGSW ciphertext is basically a collection of $2l$ RLWE ciphertexts.

We stress that all (R)LWE ciphertexts contain certain levels of noise that can be amplified by the homomorphic operators described below. Even if such noises may not cause decryption failures, they contaminate the least significant bits of the plaintext messages and reduce the precision of the ciphertext until it becomes indecipherable.

2.2 Homomorphic Operators

Here, we explain the fundamental homomorphic operators used throughout this work. Note that, we abbreviate the ciphertext notations to $\mathsf{LWE}(m)$, $\mathsf{RLWE}(m)$ and $\mathsf{RGSW}(m)$ when the parameters are not important to the discussion.

2.2.1 Homomorphic Arithmetic Operators. We primarily use homomorphic arithmetic operators to evaluate linear (i.e., polynomial) operations over RLWE ciphertexts. Specifically, operations over homomorphic arithmetic circuits are outlined as follows.

• $+$, $-$ and $\cdot$: Ciphertext addition, subtraction and multiplication. In this work, we consider an RLWE ciphertext to be a tuple of polynomials, where additions (+) and multiplications ($\cdot$) are supported between ciphertexts. For example, given two RLWE ciphertexts $\mathsf{RLWE}_0=(\tilde{b_0},\tilde{a_0})$ and $\mathsf{RLWE}_1=(\tilde{b_1},\tilde{a_1})$. The addition between the two ciphertexts is defined as $\mathsf{RLWE}(\tilde{m}_0+\tilde{m}_1)=\mathsf{RLWE}(\tilde{m_0})+\mathsf{RLWE}(\tilde{m_1})=(\tilde{b_0}+\tilde{b_1},\tilde{a_0}+\tilde{a_1})$. Similarly, one can define the homomorphic subtraction between two RLWE ciphertexts. Finally, given two RLWE ciphertexts $\mathsf{RLWE}(\tilde{m}_0)$and $\mathsf{RLWE}(\tilde{m}_1)$ which encrypt plaintexts $\tilde{m_0}$ and $\tilde{m_1}$, the homomorphic multiplication $\mathsf{RLWE}(\tilde{m_0})\cdot\mathsf{RLWE}(\tilde{m_1})$ results in $\mathsf{RLWE}(\tilde{m_0}\cdot\tilde{m_1})$. More details on ciphertext multiplication and the so-called relinearization process can be found in [20].

• $p(\mathsf{RLWE}(\tilde{m}))$: Polynomial evaluation over the input ciphertext. For any polynomial $p(x)$, note that $p(\mathsf{RLWE}(\tilde{m}))$ represents the homomorphic evaluation of p over the input ciphertext $\mathsf{RLWE}(\tilde{m})$ encrypting $\tilde{m}$, i.e., $\mathsf{RLWE}_{\text{out}}=\mathsf{RLWE}(p(\tilde{m}))=p(\mathsf{RLWE}(\tilde{m}))$, decrypts to $p(\tilde{m})$. Here, the evaluation of p can be realized by the homomorphic multiplication and addition operators. For example, let $ct=\mathsf{RLWE}(\tilde{m})$, if $p(x)=x^3+x$, then $p(ct)=ct\cdot ct\cdot ct+ct$. We note that there also exist more efficient algorithms for evaluating $p(x)$ over $\mathsf{RLWE}(m)$ [18, 25, 59, 73].

2.2.2 Homomorphic Logic Operators. Different from homomorphic arithmetic operators, homomorphic logic operators can be faster when evaluating a deep chain of non-polynomial functions. The main homomorphic logic operators are summarized as follows.

• $\mathsf{CMUX}(\mathsf{R/GSW}(\mathsf{t}),\mathsf{R/LWE}(a),\mathsf{R/LWE}(b))$: The homomorphic selector. Note that the homomorphic selector works for both RLWE and LWE ciphertext inputs, and we take the RLWE case as an example here. Given $\mathsf{RLWE}(a)$ and $\mathsf{RLWE}(b)$ along with a control signal $\mathsf{RGSW}(\mathsf{t})$ that encrypts a binary plaintext $\mathsf{t}\in\{0,1\}$, $\mathsf{CMUX}(\mathsf{RGSW}(\mathsf{t}),\mathsf{RLWE}(a),\mathsf{RLWE}(b))$ homomorphically computes

$$\mathsf{t}\ ?\ \mathsf{RLWE}(a)\ :\ \mathsf{RLWE}(b), \tag{1}$$

i.e., the function selects $\mathsf{RLWE}(a)$ if $\mathsf{t}=1$ and $\mathsf{RLWE}(b)$ if $\mathsf{t}=0$.

• PBS(LWE(m), **BK**, $T(x)$): The programmable bootstrapping operator. Given an LWE ciphertext ct = LWE(m) and a discrete function $T(x)$, and the bootstrapping key **BK**, PBS outputs LWE($T(m)$) with a constant (i.e., input-independent) noise level.

• HomGate(LWE(m_0), LWE(m_1), OP): The homomorphic logic gate. Given two LWE ciphertexts LWE(m_0) and LWE(m_1) along with a two-input logic gate OP, HomGate(LWE(m_0), LWE(m_1), OP) produces LWE(OP(m_0, m_1)). Note that, except for homomorphic NOT gate which is simply the negation of the input LWE ciphertext, other gates including XOR, AND, OR, NAND, etc., are evaluated based on PBS. As a result, HomGate always outputs a ciphertext with a constant level of noise and permits circuits of unbounded depths to be evaluated over the homomorphic gates.

2.2.3 Homomorphic Format Conversion Operators. Since homomorphic operators work on different types of homomorphic ciphertexts, homomorphic format conversion operators are required to convert between different ciphertext formats, e.g., from an LWE ciphertext to an RLWE ciphertext. Here, we briefly summarize the functionalities for each of the conversion operators.

• RLWEtoLWEs(RLWE($\tilde{m}$)): The conversion from one $\mathsf{RLWE}_{\tilde{s}}^{N,Q}$ ciphertext to a set of N $\mathsf{LWE}_{\mathbf{s}}^{N,Q}$ ciphertexts. Like [32], RLWEtoLWEs outputs N LWE ciphertexts $ct_0, ct_1, ..., ct_{N-1}$ where ct_i encrypts the i-th plaintext coefficient of the message polynomial $\tilde{m}$.

• LWEstoRLWE($\mathsf{LWE}_0, \cdots, \mathsf{LWE}_{N-1}$): The conversion from a set of N $\mathsf{LWE}_{\mathbf{s}}^{n,q}$ ciphertexts to one $\mathsf{RLWE}_{\tilde{s}}^{N,Q}$ ciphertext, which is basically the inverse of RLWEtoLWEs. However, we note that existing methods for LWEstoRLWE end up with significantly reduced precision in the resulting RLWE ciphertext. As later explained in Section 4.3, due to the underlying cryptographic limitations, existing methods devised for LWEstoRLWE cannot be directly applied to a DB setting which requires a large number of high-precision aggregation operations [26, 77].

• LWEtoRGSW(LWE, **BK**): The conversion from an $\mathsf{LWE}_{\mathbf{s}}^{n,q}$ ciphertext to an $\mathsf{RGSW}_{\tilde{\mathbf{s}}}^{N',Q'}$ ciphertext. LWEtoRGSW is generally used to convert a HomGate result in the LWE format to an RGSW switching signal for the CMUX operator.

Due to space limitation, more details about BlindRotate, PBS, HomGate, RLWEtoLWEs, LWEstoRLWE and LWEtoRGSW can be found in the related literature [26, 33].

3 FRAMEWORK OVERVIEW

In this section, we first provide a high-level workflow of HE^{3}DB in Section 3.1, and then analyze the threat model and security guarantees of HE^{3}DB in Section 3.2. Lastly, we summarize how to implement key SQL statements for the filter-aggregation process in Section 3.3.

3.1 HE^{3}DB Workflow

The HE^{3}DB framework is composed of three interlinked stages: (1) offline data encryption, (2) online query processing, and (3) online data analysis. In what follows, we detail the main objective of each stage and briefly sketch the lower-level FHE operations.

To begin with, in (1) the offline data encryption stage, the client first needs to encrypt all the data tables in the database using FHE and outsources the encrypted database to the server. Specifically, for every table $\mathcal{T} \in \mathcal{D}$, each column $t_i \in \mathcal{T}$ will first be divided into $J = \lceil |\mathcal{T}|_{\text{row}}/N \rceil$ data chunks, each of size N. Next, each data chunk is encoded into a degree-N plaintext polynomial $\tilde{m}_{i,j}$, and encrypted into an RLWE ciphertext RLWE($\tilde{m}_{i,j}$). The resulting encrypted database is thus a set of RLWE ciphertexts $\{\mathsf{RLWE}(\tilde{m}_{i,j})\}$. Along with the encryption database, the evaluation keys (e.g., **BK**) will also be sent to the server. We note that different from existing bit-by-bit [71] and row-by-row [93] schemes, we encrypt the entire database using only RLWE ciphertexts, significantly reducing the client-side encryption complexity and communication burdens.

Next, two main steps in (2), the online query processing stage, are homomorphic filtering and homomorphic aggregation. For filtering, we execute a number of homomorphic comparisons to realize the SELECT, JOIN, GROUP BY, and ORDER BY filtering operations on encrypted data items, and more details on the exact input-output behaviors of these SQL statements are provided in Section 3.3. Then, logic (MIN, MAX, Top-k) and arithmetic (SUM, AVG) aggregations are executed according to the statements in the queries. To better convey our idea, we first summarize how the SQL statements are built over the proposed FHE operators in Section 3.3. Later in Section 4, the proposed high-precision homomorphic comparison, aggregation, and conversion operators are presented.

If needed, the server performs further computations on the aggregated results in (3), the online data analysis stage. In most practical applications, the client also wishes to outsource some data analysis algorithms. Here, we assume that the data analysis algorithm Analysis contains a set of algorithm parameters W, which belongs to either the client or the server. Unfortunately, due to the limit of ciphertext noise level [71, 93] or incompatible plaintext encoding [54], queried results produced by existing FHE-based SDO often cannot be further processed without additional client-server interactions. In contrast, through the carefully designed homomorphic aggregation and conversion operators, HE^{3}DB permits arbitrary Analysis to be applied on the queried ciphertext results. A detailed presentation for the data analysis stage can be found in Section 5.

3.2 Threat Model and Security

Threat Model: Our security goal is to protect the outsourced database $\mathcal{D}$ owned by the client $\mathcal{C}$ from the semi-honest server $\mathcal{S}$, and our threat model is similar to that in previous works[35, 54, 71, 89, 93]. We assume that $\mathcal{S}$ honestly follows the protocol to execute the queries and data analysis algorithms, but acts as a probabilistic polynomial time adversary and perform computations to learn as much as possible from $\mathcal{D}$ owned by $\mathcal{C}$. The concrete public and private data from the perspective of the server is as follows.

Public Data:

- $|\mathcal{D}|$: the size of the database, i.e., the number of data tables in $\mathcal{D}$.
- $|\mathcal{T}|_{\text{row}}$, $|\mathcal{T}|_{\text{col}}$: the number of rows as well as the number of columns in some table $\mathcal{T} \in \mathcal{D}$.
- $|\mathcal{Q}|$: the number of filtering predicates in a SQL query $\mathcal{Q}$.
- Attr, |Attr|: The attribute label (e.g., gender, date) and the range of the attribute (e.g., |Gender| = 2).
- The logic connection (e.g., AND, OR) between the filtering predicates in a SQL query.
- The aggregation functions (e.g., SUM, MIN) in a SQL query.

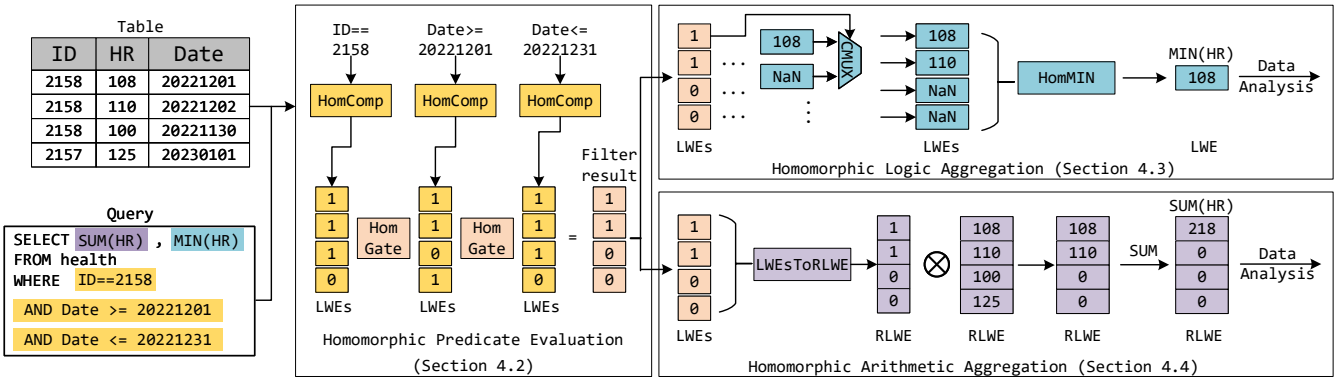


Figure 2: An example processing flow illustration in query evaluation. Here, we show how to perform logic aggregation (HomMIN) and arithmetic aggregations. Plaintext values in the figure are used only to demonstrate the range for each of the attributes.

Table 2: Summary of Operation Costs for Each Homomorphic SQL Statement.

SQL Statement	#HomComp	#HomGate	#CMUX	#LWEtoRGSW
SELECT	$\|\mathcal{T}\|_{\text{row}} \cdot \|\mathcal{Q}\|$	$\|\mathcal{T}\|_{\text{row}} \cdot (\|\mathcal{Q}\| - 1)$	0	0
JOIN	$\|\mathcal{T}_a\|_{\text{row}} \cdot \|\mathcal{T}_b\|_{\text{row}}$	0	0	0
GROUP BY	$\|\mathcal{T}\|_{\text{row}} \cdot \|\text{Attr}\|$	0	0	0
ORDER BY	$\|\mathcal{T}\|_{\text{row}} \log_2^2 \|\mathcal{T}\|_{\text{row}}$	0	$\|\mathcal{T}\|_{\text{row}}(\log_2^2 \|\mathcal{T}\|_{\text{row}} + 1)$	$\|\mathcal{T}\|_{\text{row}}(\log_2^2 \|\mathcal{T}\|_{\text{row}} + 1)$
MIN/MAX	$\|\mathcal{T}\|_{\text{row}} - 1$	0	$2\|\mathcal{T}\|_{\text{row}} - 1$	$2\|\mathcal{T}\|_{\text{row}} - 1$

$|\mathcal{Q}|$, $|\mathcal{T}|_{\text{row}}$ and $|\text{Attr}|$ are all public data known to the server.

- Analysis: the data analysis algorithm without parameters (i.e., only the procedures).

Private Data:

- $\mathcal{T}_{i,j}$, for $i \in |\mathcal{T}|_{\text{row}}$, $j \in |\mathcal{T}|_{\text{col}}$: exact values of the database items for all $\mathcal{T} \in \mathcal{D}$.
- $\mathcal{Q}_i \in \mathcal{Q}$, for $i \in |\mathcal{Q}|$: the predicate values in the SQL query.
- $w_i \in \mathcal{W}$, for $i \in |\mathcal{W}|$: the weight values of the data analysis algorithm Analysis.

Security of HE^{3}DB: Since the private data are all encrypted using FHE, the security of HE^{3}DB can be directly reduced to that of the underlying FHE scheme. Traditional FHE schemes, such as B/FV [20], CKKS [29], and TFHE [32], all guarantee security under chosen plaintext attacks, which translates to a semi-honest security on private client data. Assuming the circular security of FHE [21], switching between different ciphertext formats preserves the overall semi-honest security of the protocol. However, the parameters of such ciphertext formats need to be jointly adjusted to achieve the same level of security. Note that, since HE^{3}DB performs a linear scan of the entire database for each query and protects both the query values as well as the intermediate results throughout the whole process, HE^{3}DB is secure against access patterns and volume leakage attacks [68].

3.3 Key Operators in Filter-Aggregation

Since a SQL query can contain a variety of composed filtering and aggregation statements, HE^{3}DB breaks each statement down to atomic operators that can be directly implemented using FHE. Here, we use the simple plaintext SQL query shown in Figure 2 as an illustration for the statement evaluation process.

• SELECT. The homomorphic SELECT operator takes as inputs $|\mathcal{Q}|$ encrypted predicates with $|\mathcal{T}|_{\text{col}} \cdot J$ ($J = \lceil |\mathcal{T}|_{\text{row}}/N \rceil$) columns of RLWE ciphertexts, and outputs a set of LWE ciphertexts $\mathbf{sc} = (sc_0, sc_1, ..., sc_{|\mathcal{T}|_{\text{row}}-1})$. Here, each sc_i is an LWE ciphertext that encrypts either 1 (true) or 0 (false), indicating if the i-th row of $\mathcal{T}$ is selected. Functionally, the homomorphic SELECT operator consists two steps: i) individual predicate evaluation, and ii) logic connection between the predicates. In HE^{3}DB, encrypted predicate evaluations (as shown in Figure 2) are realized as homomorphic logical comparisons ($>, \geq, <, \leq, ==, \neq$) between ciphertexts based on the HomComp operator. Since HomComp is one of the key building blocks for our framework, more discussions on HomComp are delivered in Section 4.1. Besides, the unbounded-depth logic connection (AND, OR) between the individual comparison results can be implemented by the HomGate operator defined in Section 2.2. Since FHE-based filtering is inherently a linear scan, the cost of HomComp in SELECT is $|\mathcal{T}|_{\text{row}} \cdot |\mathcal{Q}|$ as each predicate requires the comparison between the queried predicate value and all items in the corresponding column of the encrypted table ($|\mathcal{T}|_{\text{row}}$). In addition, connecting $|\mathcal{Q}|$ predicates cost $|\mathcal{T}|_{\text{row}} \cdot (|\mathcal{Q}| - 1)$ HomGate operations, as the $|\mathcal{Q}|$ predicates contains $(|\mathcal{Q}| - 1)$ logic connections between the individual comparison results.

• JOIN. A homomorphic inner join is composed of the Cartesian product of the two encrypted tables and the evaluation of the join condition, which can simply be implemented by the homomorphic

comparison operator HomComp. Thus, for two data tables $\mathcal{T}_a$ and $\mathcal{T}_b$, the cost of HomComp in JOIN is $|\mathcal{T}_a|_{\text{row}} \cdot |\mathcal{T}_b|_{\text{row}}$.

• GROUP BY. To evaluate the homomorphic GROUP BY operator, we generate multiple copies of the original query Q, where each copy is equipped with an additional equality test over the group attribute. For example, a GROUP BY on the attribute Gender = {0, 1} can be implemented by the two queries Q AND Gender == 0 and Q AND Gender == 1. Let the range of the GROUP BY attribute Attr be |Attr|, the cost of GROUP BY is $|\mathcal{T}|_{\text{row}} \cdot |\text{Attr}|$ HomComp operations.

• **Aggregation functions**. As shown in Figure 2, after the homomorphic predicate evaluation, we get $|\mathcal{T}|_{\text{row}}$ LWE ciphertexts encrypting the filtering results. The homomorphic aggregation computes the functions on the filtered rows. HE^{3}DB supports both logic aggregation (such as MIN, MAX, and Top-k) and arithmetic aggregation (such as SUM, AVG, and COUNT) functions, and we detail the exact constructions in Section 4.2 and Section 4.3. Additionally, we note that, ORDER BY is essentially a logic aggregation, and can be implemented similar to Top-k. The main challenge for all types of homomorphic logic aggregations is how to apply numerical comparisons only on the filtered result. Since a logic mismatch is represented using a value 0 in most existing homomorphic comparison operators [35, 77], comparing logic mismatches and legitimate attribute values may produce incorrect aggregation results. We detail our method to overcome this challenge in Section 4.2.

4 CRYPTOGRAPHIC BUILDING BLOCKS

In this section, we introduce the main cryptographic tools we develop, namely, the homomorphic comparison operator HomComp for homomorphic predicate evaluation (Section 4.1), the homomorphic minimum and maximum operators HomMIN and HomMAX for homomorphic logic aggregation (Section 4.2), and a new LWEstoRLWE operator for homomorphic arithmetic aggregation (Section 4.3).

4.1 Homomorphic Predicate Evaluation

Since predicate evaluations are essentially concatenated comparisons, the core computation here is the comparison between two encrypted ciphertexts. In this section, we outline a series of homomorphic algorithms we devised that lead to the proposed homomorphic comparison algorithm HomComp, which supports unbounded-depth homomorphic predicate evaluation.

4.1.1 The Main Building Block: HMSB. We first discuss the homomorphic most significant bit (MSB) extraction algorithm, HMSB, which is the key building block in instantiating HomComp. We first point out that, as observed in the existing works [75, 77, 93], the comparison between two ciphertexts ct_a and ct_b encrypting integers a and b can be evaluated by extracting the MSB from $ct_a - ct_b$, since the MSB of the subtraction result encrypts a value of 0 when $a \geq b$ and a value of 1 when $a < b$ in the two's complement representation. However, we observe that existing homomorphic MSB extraction algorithms following the PBS procedure [77] proposed in [33] may not fit encrypted databases due to their low accuracy. In particular, such algorithms can only successfully extract the MSB when the plaintext message (which is the subtraction result $a - b$ here) is an integer that is less than 6 bits (the full PBS-based MSB extraction algorithm and the explanation on the precision constraints can be found in the Appendix).

Algorithm 1: $2k$-bit HMSB

Input : A LWE ciphertext $ct_I = \text{LWE}(m)$, where $m = 2^k \cdot m_1 + m_0$.
Input : A bootstrapping key **BK**.
Output: An LWE ciphertext $ct_O = \text{LWE}(\text{MSB}(m))$.

1 $ct_1 \leftarrow 2^{k-1} ct_I$; ▷ $ct_1 = \text{LWE}(2^{2k-1}\text{LSB}(m_1) + 2^{k-1}m_0)$
2 $ct_2 \leftarrow \text{HMSB}_{k+1}(ct_1)$; ▷ $ct_2 = \text{LWE}(2^{2k-1}\text{LSB}(m_1))$
3 $ct_3 \leftarrow ct_1 - ct_2$; ▷ $ct_3 = \text{LWE}(2^{k-1}m_0)$
4 $ct_4 \leftarrow \text{PBS}(ct_3, \mathbf{BK}, T(x) = \frac{x}{2^{k-1}})$; ▷ $ct_4 = \text{LWE}(m_0)$
5 $ct_5 \leftarrow ct_I - ct_4$; ▷ $ct_5 = \text{LWE}(2^k m_1)$
6 $ct_O \leftarrow \text{HMSB}_k(ct_5)$; ▷ $ct_O = \text{LWE}(\text{MSB}(m))$

Return: $ct_O = \text{LWE}(\text{MSB}(m))$

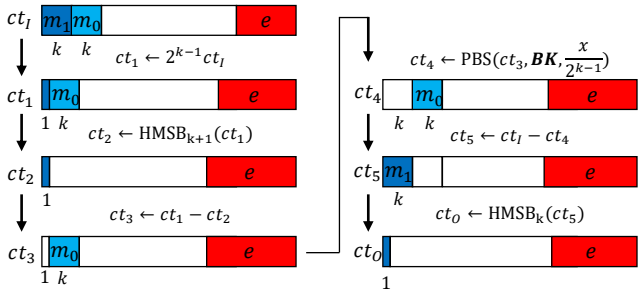


Figure 3: A line-by-line illustration of how Algorithm 1 works with the input ciphertext $ct_I = \text{RLWE}(2^k m_1 + m_0)$.

Algorithm 2: tk-bit HMSB

Input : A LWE ciphertext $ct_I = \text{LWE}(m)$, where $m = \sum_{i=0}^{t} 2^{ik} \cdot m_i$.
Input : A bootstrapping key **BK**.
Output: A LWE ciphertext $ct_O = \text{LWE}(\text{MSB}(m))$.

if $t == 1$ **then**
  $ct_O \leftarrow \text{HMSB}_k(ct_I)$
else
  $ct_1 \leftarrow 2^{tk-1} ct_I$
  $ct_2 \leftarrow \text{HMSB}_{k+1}(ct_1)$
  $ct_3 \leftarrow ct_1 - ct_2$
  $ct_4 \leftarrow \text{PBS}(ct_3, \mathbf{BK}, T(x) = \frac{x}{2^{tk-1}})$
  $ct_5 \leftarrow ct_I - ct_4$
  $ct_O \leftarrow \text{HMSB}_{(t-1)k}(ct_5)$

Return: $ct_O = \text{LWE}(\text{MSB}(m))$

To overcome the precision loss problem while retaining high performance, we propose a new homomorphic MSB extraction algorithm HMSB. The key idea of HMSB is to iterate the PBS algorithm a certain number of times over the same ciphertext, such that we can extract the MSB of arbitrary-precision messages. We do note that HMSB is not the first to adopt iterative bootstrapping for improving the accuracy of homomorphic computations. For instance, [11] also employs the same idea and devised a high-precision ciphertext bootstrapping algorithm.

Algorithm 3: HomComp

Input : Two ciphertexts $ct_a = \mathsf{LWE}(a), ct_b = \mathsf{LWE}(b)$.
Input : Comparison operator cmp.
Input : A bootstrapping key **BK**.
Output: A LWE ciphertext $ct_O = \mathsf{LWE}(c)$ where $c = 1$ if a cmp b is True, else $c = 0$.

1 **switch** cmp **do**
2 **case** $\geq$ **do** $ct_O \leftarrow \mathsf{HomGate}((\mathsf{HMSB}(c_a - c_b), \mathrm{NOT}))$;
3 **case** $>$ **do** $ct_O \leftarrow \mathsf{HMSB}(c_b - c_a)$;
4 **case** $\leq$ **do** $ct_O \leftarrow \mathsf{HomGate}(\mathsf{HMSB}(c_b - c_a), \mathrm{NOT})$;
5 **case** $<$ **do** $ct_O \leftarrow \mathsf{HMSB}(c_a - c_b)$;
6 **case** == **do**
7 $ct_O \leftarrow \mathsf{HomGate}(\mathsf{HMSB}(c_a - c_b), \mathsf{HMSB}(c_b - c_a), \mathrm{NOR})$
8 **case** $\neq$ **do**
9 $ct_O \leftarrow \mathsf{HomGate}(\mathsf{HomComp}(ct_a, ct_b, ==), \mathrm{NOT})$

Return: $ct_O = \mathrm{LWE}(c)$

We start with a PBS-based MSB extraction algorithm, named PBS-HMSB. We assume that PBS-HMSB can successfully extract the MSB of a κ-bit integer (e.g., as mentioned, $\kappa = 6$ using the original PBS method proposed in [33]). Let $k = \kappa - 1$ (e.g., $k = 5$ in our implementation), we show how to construct a $2k$-bit homomorphic MSB extraction algorithm based on PBS-HMSB in Algorithm 1. First, given a $2k$-bit input ciphertext $ct_I = \mathsf{LWE}(m)$, it is obvious that $m = 2^k \cdot m_1 + m_0$. Our critical insight is that the MSB of ct_I depends on the first bit of the plaintext, so we can homomorphically remove the remaining bits to construct a k-bit ciphertext. To achieve the bit removal, as depicted in Figure 3, we homomorphically shift the internal plaintext message towards the left and turns $\mathsf{LWE}(2^k \cdot m_1 + m_0)$ into $\mathsf{LWE}(m_0)$. Next, we can homomorphically subtract $\mathsf{LWE}(m_0)$ from $\mathsf{LWE}(m) = \mathsf{LWE}(2^k \cdot m_1 + m_0)$, and get $\mathsf{LWE}(2^k \cdot m_1)$. Then, we can apply PBS-HMSB to extract the MSB of the k-bit plaintext message m_1, which is also the MSB of $\mathsf{LWE}(m)$, i.e., ct_I. However, to correctly bootstrap the ciphertext ct_3 in Algorithm 1, the MSB of the encrypted plaintext must be set to zero [33, 77]. To address this problem, we leave the MSB as is in the shifting processing such that it can later be zeroed out (as indicated in lines 2-3 of Algorithm 1).

Based on a similar idea, we can extend the $2k$-bit HMSB to arbitrary precision tk for any positive integer t by repeatedly shifting and subtracting k-bit plaintext messages. As formalized in Algorithm 2, we first recursively discard the least significant k bits of the initial tk-bit ciphertext t times until only the most significant k bits remain. Then, we simply apply PBS-HMSB with κ-bit precision and accurately extract the MSB of the input ciphertext with tk-bit-precision plaintext message.

4.1.2 HomComp *Based on* HMSB. Based on the arbitrary-precision HMSB algorithm devised above, we can carry out arbitrary-precision comparisons between the queried attributes and the DB items. Let ct_a and ct_b be the two LWE ciphertexts that encrypt a and b, and cmp be the type of comparison to be performed ($\geq, >, \leq, <, ==, \neq$). $\mathsf{HomComp}(ct_a, ct_b, \mathrm{cmp})$ outputs an LWE ciphertext encrypting 1 if the predicate a cmp b is true and 0 if a cmp b is false. The HomComp algorithm can be easily constructed using HMSB with some additional homomorphic gates. We summarize the exact arithmetic procedure for HomComp in Algorithm 3. The complexity of our HomComp algorithm can be expressed by the following lemma, where the formal proof is given in the Appendix.

Algorithm 4: HomMIN

Input : Two LWE ciphertexts $ct_a = \mathsf{LWE}(a), ct_b = \mathsf{LWE}(b)$.
Input : A bootstrapping key **BK**.
Output: A LWE ciphertext $ct_O = \mathsf{LWE}(\mathsf{MIN}(a, b))$.

1 $ct \leftarrow \mathsf{HomComp}(ct_a, ct_b, \leq)$
2 $C \leftarrow \mathsf{LWEtoRGSW}(ct, \mathbf{BK})$
3 $ct_O \leftarrow \mathsf{CMUX}(C, ct_a, ct_b)$

Return: $ct_O = \mathrm{LWE}(\mathsf{MIN}(a, b))$

Algorithm 5: HomMINAgg

Input : An RLWE ciphertext of the aggregated column $ct = \mathrm{RLWE}(\tilde{m})$ where $\tilde{m}_i = m_i$ for $i \in \mathbb{Z}_{|\mathcal{T}|_{\mathrm{row}}}$.
Input : Filtered result $cf = (cf_0, cf_1, ..., cf_{|\mathcal{T}|_{\mathrm{row}}-1})$, where $cf_i = \mathrm{LWE}(f_i)$, $f_i = 0$ or 1.
Input : The pre-defined constant $\mathrm{LWE}_{\mathrm{NaN}}$.
Input : A bootstrapping key **BK**.
Output: A LWE ciphertext $ct_O = \mathsf{LWE}(\mathsf{MIN}(m_{t_0}, ..., m_{t_{f-1}}))$ where $f = \sum_{i=0}^{|\mathcal{T}|_{\mathrm{row}}-1} f_i$ and $f_{t_0}, f_{t_1}, ..., f_{t_{f-1}} = 1$.

1 **ct** $\leftarrow \mathsf{RLWEtoLWEs}(ct) = (ct_0, ct_1, ..., ct_{|\mathcal{T}|_{\mathrm{row}}-1})$
2 **for** $i = 0$ **to** $|\mathcal{T}|_{\mathrm{row}} - 1$ **do**
3 $C_i \leftarrow \mathsf{LWEtoRGSW}(cf_i, \mathbf{BK})$
4 $ct_i \leftarrow \mathsf{CMUX}(C, ct_i, \mathrm{LWE}_{\mathrm{NaN}})$
5 **for** $i = 0$ **to** $\log_2 |\mathcal{T}|_{\mathrm{row}} - 1$ **do**
6 **for** $j = 0$ **to** $2^{\log_2 |\mathcal{T}|_{\mathrm{row}} - i - 1} - 1$ **do**
7 $ct_{2^{i+1} \cdot j} \leftarrow \mathsf{HomMIN}(ct_{2^{i+1} \cdot j}, ct_{2^{i+1} \cdot j + 2^i}, \mathbf{BK})$

Return: $ct_O = ct_0$

Lemma 4.1. *Given the two tk-bit ciphertext ct_a, ct_b with a comparison operator* HomComp*, the Algorithm 3 outputs LWE ciphertext $ct_O = \mathsf{LWE}(c)$ where $c = 1$ if a cmp b is True, else $c = 0$. The number of* PBS *evaluated by Algorithm 3 is $2t - 1$ or $2t$.*

4.2 Homomorphic Logic Aggregation

Most existing homomorphically encrypted database techniques [71, 93] only support arithmetic aggregation (e.g., SUM or AVG). To support logic aggregation such as MIN, MAX, Top-k and SQL statement such as ORDER BY, we propose new two-input logic operators that can be used to construct arbitrary logic aggregation functions. Due to space limitation, we use HomMIN as an example in this section. Our methodology applies generally to other types of logic aggregations.

4.2.1 Homomorphic MIN *and* MIN *aggregation.* While there exists prior works [30, 77] for computing HomMIN and HomMAX operators, such techniques cannot be directly applied to a homomorphically encrypted database due to two major obstacles. The first obstacle is the low evaluation speed and unstable aggregation precision. For example, while [30] can achieve high-precision HomMIN evaluation, the method incurs a large number of latency overheads. On the other hand, although [77] can evaluate HomMIN and HomMAX

relatively fast (but still slower than our proposed technique), the technique suffers from the low precision (< 6-bit) problem. The second obstacle is how to evaluate logic aggregations only on filtered rows. For a set of homomorphically filtered results, the server cannot distinguish between an attribute value originally encrypted to be zero and a filtered attribute that is set to be zero. Therefore, when performing operations such as HomMIN, we can produce incorrect results if we aggregate on invalid rows that are marked as zeroes because they are filtered by the SQL statements.

To tackle the above two obstacles, we propose a new homomorphic logic aggregation procedure. For the first barrier, we introduce a new mechanism to homomorphically evaluate MIN (and MAX). We point out that, for two numbers a and b, prior works [30, 77] adopts the following formula to compute the MIN function.

$$\texttt{MIN}(a, b) = (a \le b)\ ?\ a\ :\ b = \frac{a+b}{2} - \frac{|a-b|}{2} \tag{2}$$

However, evaluating the absolute function in HE will either be too slow or too inaccurate. To avoid the absolute function, we propose to implement MIN using the homomorphic comparison HomComp proposed in Section 4.1 and the homomorphic selector CMUX defined in Section 2.2. Essentially, we compute

$$\texttt{HomMIN}(a, b) = \text{GSW}(a \le b)\ ?\ ct_a\ :\ ct_b, \tag{3}$$

where $\text{GSW}(a \le b) = \texttt{LWEtoRGSW}(\texttt{HomComp}(ct_a, ct_b, \le)$. Observe that Eq. (3) is precisely Eq. (1), and we sketch the full algorithm in Algorithm 4. That is, using HomComp, $a \le b$ can be homomorphically evaluated with high precision and high evaluation speed. In addition, since CMUX is also a fast operator (around 1000× faster than HomComp), our HomMIN operator can be both faster and more accurate than existing solutions [30, 77].

Next, we solve the second obstacle by defining a constant ciphertext LWE_{NaN} encrypting NaN, i.e., a 'not a number' ciphertext. As detailed in Algorithm 5, the key idea is simple: before comparing two LWE ciphertexts, we first use the encrypted filtered result $\text{GSW}(\mathfrak{t})$ to pre-filter the input ciphertext ct_i by computing

$$\text{GSW}(\mathfrak{t})\ ?\ ct_i\ :\ \text{LWE}_{\text{NaN}}, \tag{4}$$

which is again a CMUX operation. Note that, in practice, LWE_{NaN} can be initialized to be the largest value in the range of the plaintext message space (and the smallest value in the case of HomMAXAgg). Then, we can proceed to build a conventional min-tree of depth $\log_2 |\mathcal{T}|_{\text{row}}$ to calculate the minimum value over all rows in the attribute column. The complexity analysis of the algorithm is provided in Lemma 4.2 (the proof can be found in the Appendix).

Lemma 4.2. *In Algorithm 5 the number of homomorphic* CMUX *gates is* $2|\mathcal{T}|_{\text{row}} - 1$. *The number of homomorphic* LWEtoRGSW *operators is* $2|\mathcal{T}|_{\text{row}} - 1$, *and the number of homomorphic* HomComp *operators is* $|\mathcal{T}|_{\text{row}} - 1$.

4.2.2 Extending to Other Types of Aggregations. As mentioned, the HomMAX and HomMAXAgg can be implemented in a very similar way as HomMIN and HomMINAgg, except that LWE_{NaN} needs to be defined as the smallest value possible here. Meanwhile, to achieve homomorphic ORDER BY, we propose a simple swap function based on the HomMIN/HomMAX operators as

$$\texttt{HASwap}(ct_a, ct_b) = \texttt{HomMAX}(ct_a, ct_b), \texttt{HomMIN}(ct_a, ct_b) \tag{5}$$

$$\texttt{HDSwap}(ct_a, ct_b) = \texttt{HomMIN}(ct_a, ct_b), \texttt{HomMAX}(ct_a, ct_b), \tag{6}$$

depending the ascending (HASwap) and descending (HDSwap) orders requested by ORDER BY. Based on the homomorphic swap operators, it is easy to construct the sorting function by the existing sorting algorithms such as quick sort [61], and bitonic sorting [14]. Finally, Top-k can be implemented by first executing a descending ORDER BY, and then selecting the first k LWE ciphertexts.

Algorithm 6: LWEstoRLWE

Input : L LWE ciphertexts $(\text{LWE}_0, \text{LWE}_1, ..., \text{LWE}_{L-1})$ where $\text{LWE}_i = \text{LWE}_{\mathbf{s}}^{n,q}(m_i) = (b_i, \mathbf{a_i})$ and $m_i = 0/1$.

Input : A conversion key $\mathbf{CK} = \text{RLWE}_{\tilde{s}}^{N,Q_0}(\mathcal{E}(\mathbf{s}))$.

Input : A precision parameter δ, range parameter (u, v)

Output: An RLWE ciphertext $ct = \text{RLWE}_{\tilde{s}}^{N,Q}(\mathcal{E}(\mathbf{m}))$, where $\mathbf{m}_i = m_i \pm e$ for $i \in \mathbb{Z}_L$

Let $\mathbf{b} \leftarrow [b_0, b_1, ..., b_{L-1}]$, $\mathbf{A} \leftarrow [\mathbf{a_0}, \mathbf{a_1}, ..., \mathbf{a_{L-1}}]^T \in \mathbb{Z}_q^{L \times n}$

$ct_1 \leftarrow \mathbf{CK} \diamond \mathbf{A}$; ▷ $ct_1 = \text{RLWE}_{\tilde{s}}^{N,Q_1}(\mathcal{E}(\mathbf{As}))$

$ct_2 \leftarrow ct_1 + (\mathcal{E}(\mathbf{b}, 0))$; ▷ $ct_2 = \text{RLWE}_{\tilde{s}}^{N,Q_1}(\mathcal{E}(\mathbf{As} + \mathbf{b}))$

$ct_3 \leftarrow \texttt{HomMod}(ct_2, q)$; ▷ $ct_3 = \text{RLWE}_{\tilde{s}}^{N,Q_2}(\mathcal{E}(\mathbf{m}))$

Generate the approximate polynomial $p(x)$

$ct \leftarrow p_{\text{Round}}(ct_3)$; ▷ $ct = \text{RLWE}_{\tilde{s}}^{N,Q_3}(\mathcal{E}(p_{\text{Round}}(\mathbf{m})))$

Return: $ct = \text{RLWE}_{\tilde{s}}^{N,Q}(\mathcal{E}(p_{\text{Round}}(\mathbf{m})))$

4.3 Homomorphic Arithmetic Aggregation

To support homomorphic arithmetic aggregations (such as SUM, COUNT, and AVG), the $|\mathcal{T}|_{\text{row}}$ LWE ciphertexts which encrypt the filtered results have to be packed in an RLWE ciphertext as arithmetic aggregation over LWE ciphertexts are slow. In this section, we propose a new homomorphic ciphertext conversion algorithm LWEstoRLWE and the associated homomorphic arithmetic aggregations procedure.

4.3.1 Homomorphic ciphertext conversion LWEstoRLWE. Existing works in converting LWE ciphertexts into an RLWE ciphertext include PEGASUS [77] and the method of Chen *et al.* [26]. Taking advantage of SIMD packing arithmetic homomorphic circuits, PEGASUS [77] performs fast ciphertext conversion but suffers from the low-precision problem due to the noise growth in homomorphic arithmetic circuits. In contrast, Chen *et al.* [26] packs the LWE ciphertexts by evaluating a series of recursive homomorphic automorphisms [56]. We point out that Chen *et al.* [26] enjoys high-precision packing capability, but induces a higher amount of latency overheads.

Here, we design a homomorphic LWEstoRLWE algorithm that achieves both, a fast algorithm that can accurately pack a set of LWE ciphertexts into one RLWE ciphertext. Our critical insight is that, in a homomorphically encrypted database, the predicate results are in a very limited range, e.g., usually only 0's and 1's to express the logical validity of the database entries. Thus, we

can actually "correct" the errors on the low-precision ciphertext to obtain a high-precision ciphertext.

The detailed algorithm LWEstoRLWE involves the following steps. First of all, in Algorithm 6, we are given L LWE filtering results $\mathsf{LWE}_0, \mathsf{LWE}_1, ..., \mathsf{LWE}_{L-1}$ where $\mathsf{LWE}_i = \mathsf{LWE}_{\mathbf{s}}^{n,q}(m_i) = (b_i, \mathbf{a_i})$ with a conversion key $\mathbf{CK}$. The conversion key $\mathbf{CK}$ is an RLWE ciphertext $\mathsf{RLWE}_{\tilde{s}}^{N,Q_0}(\mathcal{E}(\mathbf{s}))$ that encrypts the secret $\mathbf{s}$, where $\mathcal{E}$ stands for the CKKS encoding function [29]. On Line 1, we follow [77] and rearrange the ciphertexts to construct a ciphertext vector $\mathbf{b} = [b_0, b_1, ..., b_{L-1}]$ and a ciphertext matrix $\mathbf{A} = [\mathbf{a_0}, \mathbf{a_1}, ..., \mathbf{a_{L-1}}]^T \in \mathbb{Z}_q^{L\times n}$. Next, on Line 2, we evaluate the homomorphic matrix multiplication [56, 67, 77] $\mathbf{CK} \diamond \mathbf{A}$ to get $ct_1 = \mathsf{RLWE}_{\tilde{s}}^{N,Q_1}(\mathcal{E}(\mathbf{As}))$. Then, on Line 3, we homomorphically add $\mathbf{b}$ to ct_1 and get $ct_2 = \mathsf{RLWE}_{\tilde{s}}^{N,Q_1}(\mathcal{E}(\mathbf{As}+\mathbf{b}))$. After the above transforms, we obtain the ciphertext ct_2 encrypting $\mathbf{As}+\mathbf{b} = \mathbf{m} + t\mathbf{q}$. To remove the $t\mathbf{q}$ term, on Line 4, we follow [18, 28, 66, 73] to apply HomMod to the mod q function homomorphically on ct_2, and get ct_3. At this stage, we obtain a packed but imprecise RLWE ciphertext $ct_3 = \mathsf{RLWE}(\mathcal{E}(\mathbf{m}))$ encrypting the vector of messages $\mathbf{m}$ packed from the input. Here, we observe that for each $\mathbf{m}_i \in \mathbf{m}$, $\mathbf{m}_i$ equals either $1 \pm e$ or $0 \pm e$, where the noise e in inevitably added via the above transforms.

To correct the errors, on Line 5 in Algorithm 6, we generate a polynomial $p_{\text{Round}}(x)$ to approximate the rounding function $\mathsf{Round}(x)$. Following [18], given an interval $[u, v]$ and a pre-defined precision parameter δ, we want to obtain a polynomial $p_{\text{Round}}(x) = \sum_{i=0}^{n-1} a_i x^i$ such that

$$\forall x \in [u, v], |p_{\text{Round}}(x) - \lceil x \rfloor| < 2^{-\delta}, \tag{7}$$

where $2^{-\delta}$ is negligible. Based on the equioscillation theorem [80], $p_{\text{Round}}(x)$ is the best possible approximation of the round function if and only if

$$\exists x_0 \le ...x_i... \le x_n \in [u, v], \tag{8}$$

$$p_{\text{Round}}(x_i) - \lceil x \rfloor = \epsilon(-1)^i |p_{\text{Round}}(x) - \lceil x \rfloor|_\infty \tag{9}$$

where $\epsilon = \pm 1$. Taking Equation (9) as the judgment condition, we can construct the Remez iteration algorithm [92] to obtain the polynomial $p(x)$ meets the requirement of Equation (7). Upon obtaining $p_{\text{Round}}(x)$, we can perform the approximate homomorphic rounding function on Line 6. As explained in Section 2.2, $p_{\text{Round}}(x)$ can be directly applied to the RLWE ct_3 produced on Line 4, and we get the final output $ct = \mathsf{RLWE}(\mathcal{E}(p_{\text{Round}}(\mathbf{m})))$. Here, the guarantee is that the noises in $p_{\text{Round}}(\mathbf{m})$ will strictly be smaller than that in $\mathbf{m}$.

4.3.2 Homomorphic arithmetic aggregation. Using the filtered results packed in an RLWE ciphertext ct_{pack} based on the proposed LWEstoRLWE operator described above, we can finally aggregate over the target attribute. Let the RLWE ciphertext encrypting items in the column be ct_a, we describe how to implement three types of conventional arithmetic aggregation functions, namely, COUNT, SUM and AVG. Note that the first two functions can be implemented using simple inner products between ciphertexts. Concretely, COUNT function can be implemented by the inner product between ct_{pack} and the vector $I = (1, 1, ..., 1)$, while SUM is also an inner product between ct_{pack} and ct_a. The homomorphic inner product of two RLWE ciphertexts can be implemented over ciphertexts in both the slot format [67, 77] or in the coefficient format [62, 93]. Furthermore,

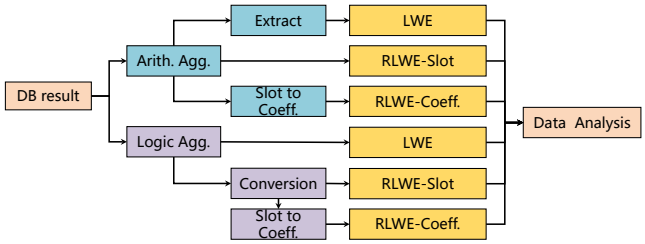


Figure 4: The conversion procedures to turn the aggregation results into data analysis inputs.

the AVG function can be evaluated by $\mathsf{AVG}() = \mathsf{SUM}() \times \mathsf{COUNT}()^{-1}$, where the inverse function can be implemented based on [30].

5 ANALYZING AGGREGATED RESULTS

Since prior works only focus on the design of SQL evaluated over encrypted data, the capability of further data processing on the queried result is mostly left unexplored. Here, we first discuss how to efficiently bridge the gap between the database query and online data analysis, and then show a case study using private decision tree evaluation.

5.1 Bridging DB Aggregation and Data Analysis

As shown in Figure 4, to support efficient data analysis algorithms using both arithmetic and logic operators, we need to convert the queried results into three different ciphertext formats: an LWE ciphertext (LWE), an RLWE ciphertext with slot encoding (RLWE-Slot), and an RLWE ciphertext with coefficient encoding (RLWE-Coeff).

We first discuss the conversion algorithms for logic aggregations. Note that, logic aggregation produces LWE ciphertexts as results. Therefore, as sketched in Figure 4, no conversion is necessary if the subsequent data analysis acts on LWE ciphertexts. For other formats of ciphertexts, the following operators can be applied.

- To RLWE-Slot: Converting a set of LWE ciphertexts into an RLWE ciphertext in slot representation is precisely the proposed LWEstoRLWE operator described in Section 4.3.
- To RLWE-Coeff: Conversion between the slot and coefficient forms of an RLWE ciphertext is well-studied in existing literature [25, 58, 59, 72]. Thus, converting multiple LWE ciphertexts into an RLWE-Coeff ciphertext can be constructed by applying the LWEstoRLWE operator followed by the slot-to-coefficient transform.

Unlike logic aggregations, the arithmetic aggregation results are RLWE ciphertexts in the slot representation. Hence, we can apply the following conversions to the resulting ciphertext for subsequent data processing.

- To LWE: We can simply apply RLWEtoLWEs to extract LWE ciphertexts from an RLWE ciphertext.
- To RLWE-Coeff: We can simply apply the slot-to-coefficient transform above to convert RLWE-Slot into RLWE-Coeff [58].

5.2 Case Study: Private Decision Tree

Here, we take private decision tree evaluation (PDTE) as an example to combine the evaluation flows of HE^{3}DB. A PDTE algorithm [35, 77] basically implements the following function over a set of LWE

Table 3: The Proposed Parameter Sets

Ciphertext	Parameters
LWE	$n = 672, \lceil \log_2 q \rceil = 32$
RLWE	$N = 65536, \lceil \log_2 Q \rceil = 653$
RGSW	$N' = 2048, \lceil \log_2 Q' \rceil = 64$

ciphertexts.

$$\{\mathsf{LWE}_{\text{output},i}\} = \mathtt{PDTEval}(\{\mathsf{LWE}_{\text{input},j}\}, \mathrm{W}), \tag{10}$$

where a set of ciphertext inputs $\{\mathsf{LWE}_{\text{input},i}\}$ are classified into ℓ output ciphertext classes $\{\mathsf{LWE}_{\text{output},j}\}$ based on the pre-trained parameters W. Similar to existing works [35], inputs and outputs of `PDTEval` are all LWE ciphertexts, since the evaluations of decision trees consist mostly of non-linear operations. As mentioned, we have two possible formats of ciphertext after the evaluation of an SQL query with aggregation: LWE and RLWE-Slot. Since logic aggregation naturally produces an LWE ciphertext, the output can directly be fed into a PDTE algorithm. Meanwhile, if we need to carry out PDTE over the arithmetically aggregated results, we can simply apply `RLWEtoLWEs`, as described in Section 5.1, and proceed with the evaluation of Equation (10).

Remark: Since most existing FHE-based PDTE algorithms are designed for two-party secure computing [35, 77], the model parameters W in Equation (10) are assumed to be known to the server. However, such an assumption does not always hold true in SDO, where the data analysis algorithm can also be outsourced [79, 95]. In particular, switching PDTE to a computation outsourcing setting actually requires a complete re-design of the evaluation protocol. Due to space limitation, we provide a complete description of our proposed SDO-orient PDTE algorithm based on the proposed `HomComp` operator in the Appendix.

6 EVALUATION

Throughout the experiments, we wish to answer the following two main research questions (RQs).

- **RQ1**: How efficient are the individual components of HE^{3}DB compared to (SOTA) methods? (Section 6.2, Section 6.3)
- **RQ2**: How does the performance of HE^{3}DB in specific SQL queries and end-to-end data analysis benchmark compare to other methods? (Section 6.4)

6.1 Experiment Setup

The entire HE^{3}DB is implemented using C++17 and compiled with GCC 10.3.0 based on Microsoft SEAL [94], OpenPEGASUS [3], and TFHEpp [102]. For single-core microbenchmarks, the experiments are carried out on an Intel Xeon Gold 6226R processor with 512GB of RAM. Meanwhile, the experiments for the TPC-H benchmarks [40] reported in Section 6.4 are performed on two servers with a total of four Intel Xeon Gold 5318Y processors with 96 cores and 1TB of RAM. The instantiated parameters are outlined in Table 3, which provide at least 128-bit of security level according to [2]. We fix the standard deviation σ of the noise distribution χ and set the security level λ to meet the Homomorphic Encryption Standard [1].

6.2 Qualitative Assessments for `HomComp`

Since `HomComp` is one of the key homomorphic operators proposed in this work, we first compare our `HomComp` with other related homomorphic comparison algorithms qualitatively, and defer quantitative results to Section 6.3. In the beginning, we note that approache like PEGASUS [77] focus on evaluating arbitrary logic functions over homomorphically encrypted ciphertext, while HE^{3}DB concentrates on the design of a homomorphic comparison algorithm. Therefore, our method achieves better comparison accuracy as well as efficiency than [77]. In contrast to general methods, the very recent work from Liu *et al.* [75] achieves large-precision homomorphic sign evaluation through the iterative use of the homomorphic floor function. While the method in [75] shares similarities with `HomComp`, we achieve better efficiency than [75], and a more detailed complexity analysis is included in the Appendix. SortingHat [35] proposes an efficient plaintext-ciphertext comparison algorithm, where one of the inputs to the comparison is known to the server. Nonetheless, under an SDO setting, SortingHat shows poor efficiency for ciphertext-ciphertext comparisons. The method of Antonio *et al.* [53] encrypts the input in a bit-wise manner, where binary circuits can be used to construct fast and high-precision homomorphic comparison. However, due to the encoding scheme, the algorithm in [53] produces a comparison result of $\{-1, 0, 1\}$, which cannot be used as filters for subsequent aggregations. Similarly, the polynomial approximation technique proposed by Cheon *et al.* [30] cannot support encrypted aggregations over results from equality tests ==, for the approximate polynomial evaluates to $1/2$ when the inputs are equal. On the other hand, some works also suggest to encrypt all database items bit-by-bit in a SIMD manner to accelerate the comparison process [69, 71]. However, in such approaches, the client encrypts the entire database using an *a-priori* fixed-set parameters that are dependent on the pre-defined maximum predicate depth. As a result, any query that requires more predicate evaluations than the pre-defined maximum depth cannot be directly evaluated. Here, the only solution is to re-encrypt entire database using a new set of encryption parameters, which can be too costly for SDO clients.

6.3 Evaluating Cryptographic Building Blocks

To answer **RQ1**, we benchmark the efficiency of each of the cryptographic building blocks proposed in Section 4, including homomorphic comparison in Section 4.1, homomorphic `MIX/MAX` evaluation in Section 4.2 and ciphertext conversion in Section 4.3.

First of all, we compare the proposed `HomComp` with the best known existing works that support unbounded-depth predicate evaluation, and the results are summarized in Table 5. We re-implement some of the existing works based on their open-source implementations [3, 38, 84] to fit our purpose (i.e., ciphertext-ciphertext comparison). As observed in Table 5, we are 7×–113× faster than existing methods. Meanwhile, we can reduce the ciphertext size by as much as 4.75× owing to our elastic ciphertext management policy. In addition, as we can see from Figure 5, our speedups are consistent across different comparison operators.

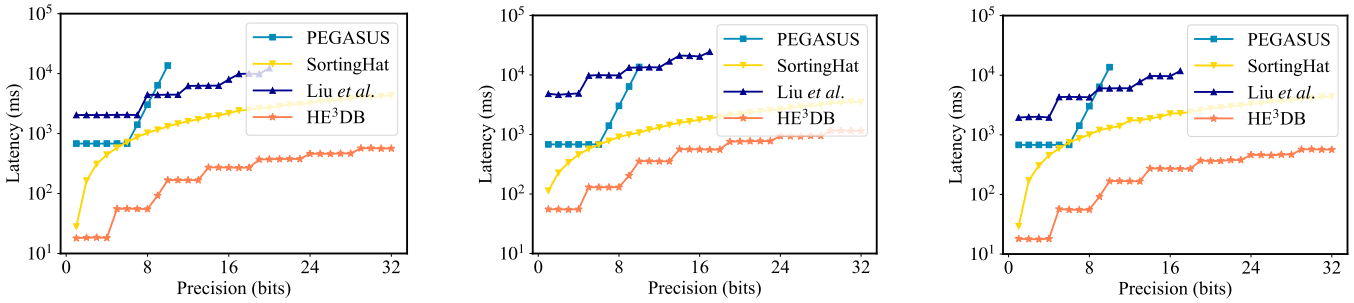
Next, we compare the latency performance of the proposed homomorphic `MIN/MAX` algorithms and SQL `ORDER BY` operator to PEGASUS [77] and Cheon *et al.* [30] with a varying number of

Table 4: Qualitative Comparisons Between Homomorphic Comparison Algorithms

	Kortekaas [71]	Antonio *et al.* [53]	Pegasus [77]	Liu *et al.* [75]	SortingHat [35]	Cheon *et al.* [30]	Ours
Cipher-cipher comp.$^{+}$	✓	✓	✓	✓	✗	✓	✓
Full comparison	✓	▲$^{++}$	✓	✓	✓	▲$^{++}$	✓
32-bit precision	✓	✓	✗	✓	✓	✓	✓
Unbounded depth	✗	✗	✓	✓	✓	✗	✓
Flexible parameter.	▲*	✗	✓	✗	✗	▲*	✓
Efficiency	High	High	Low	Low	Low	High	High
Storage	High	High	Low	Low	High	High	Low

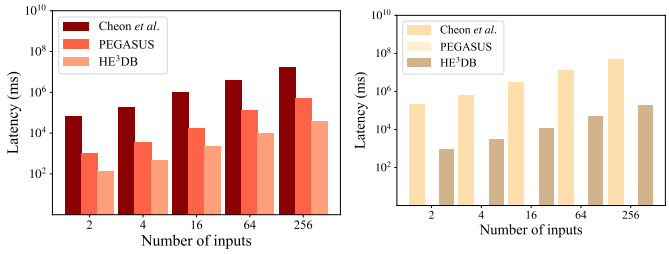
$^{+}$Cipher-cipher comparison refers to the case where both of the inputs to the comparison function are ciphertexts

*Changing encryption parameters requires the re-encryption the database $^{++}$Do not support encrypted aggregations over results from equality comparison



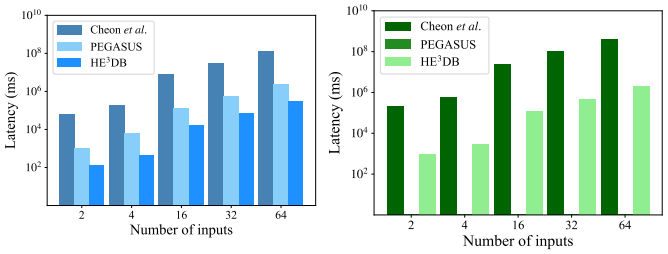
(a) Homomorphic ≥ **(b) Homomorphic ==** **(c) Homomorphic >**

Figure 5: Benchmark results for homomorphic comparison for the ≥, ==, > comparison operators with varying bit precision.



(a) 4-bit `MIN` **(b) 16-bit `MIN`**

Figure 6: Benchmark results for Homomorphic `MIN`.



(a) 4-bit `ORDER BY` **(b) 16-bit `ORDER BY`**

Figure 7: Benchmark results for Homomorphic `ORDER BY`.

inputs and bit precision for the inputs. As depicted in Figure 6 and Figure 7, our performance is on average around one order of magnitude faster than PEGASUS [77], and two orders of magnitude faster than [30]. Note in Figure 6 (b) and Figure 7 (b) the histograms of PEGASUS [77] are left blank, for PEGASUS cannot support homomorphic `MIN/MAX` and `ORDER BY` functions over 16-bit operands.

Lastly, to test the performance of the proposed `LWEstoRLWE` operator, we compare our algorithm to PEGASUS [77] and Chen *et al.* [26] over the task of converting 2^{15} LWE ciphertexts into one RLWE ciphertext. Although PEGASUS [77] achieves the lowest latency (56*s*), the method can only obtain 4 bits of precision, which is not practical in most DB applications. Although both the proposed `LWEstoRLWE` and Chen *et al.* [26] achieves higher precision (≥ 16 bits), HE^{3}DB can be as much as 2× faster than [26].

6.4 SQL Benchmarks

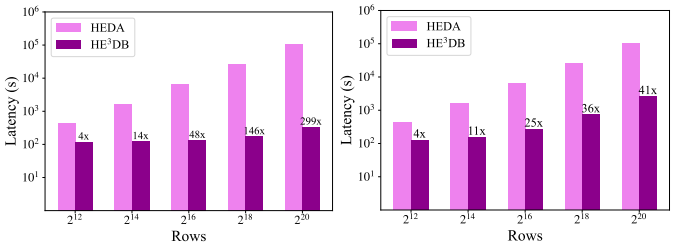
To answer the **RQ2**, we test the end-to-end performance of HE^{3}DB using the TPC-H benchmarks [40] and compare our results with the corresponding works [54, 93]. Since SAGMA [54] only uses HE for aggregation, the comparisons are placed in the Appendix.

We compare HE^{3}DB with HEDA, one of the most recent FHE-based encrypted DBMS that supports unbounded-depth filter [93]. As shown in Figure 8, on the same set of TPC-H queries, HE^{3}DB is on average 299× faster on TPC-H Query 1 and 41× faster on TPC-H Query 6. To closely examine the performance breakdown of HE^{3}DB,

Table 5: Benchmark Results for Homomorphic ≥

Precision	Methods	Latency (ms)		Ciphertext (kB)	
4	PEGASUS [77]	681	37.83×	8	2×
	SortingHat [35]	444	24.67×	10	2.5×
	Liu *et al.* [75]	2040	113×	16	4×
	Ours	18	1×	4	1×
16	PEGASUS [77]	—	—	—	—
	SortingHat [35]	2186	8.13×	38	2.38×
	Liu *et al.* [75]	7992	29.71×	16	1×
	Ours	269	1×	16	1×
32	PEGASUS [77]	—	—	—	—
	SortingHat [35]	4329	7.67×	76	4.75×
	Liu *et al.* [75]	—	—	—	—
	Ours	564	1×	16	1×

– means the method does not support or implement such bits of precision.



(a) Query 1 **(b) Query 6**

Figure 8: Comparison between HE^{3}DB and HEDA [54] on their modified TPC-H Query 1 and Query 6 with varying sizes of $|\mathcal{T}|_{\text{row}}$.

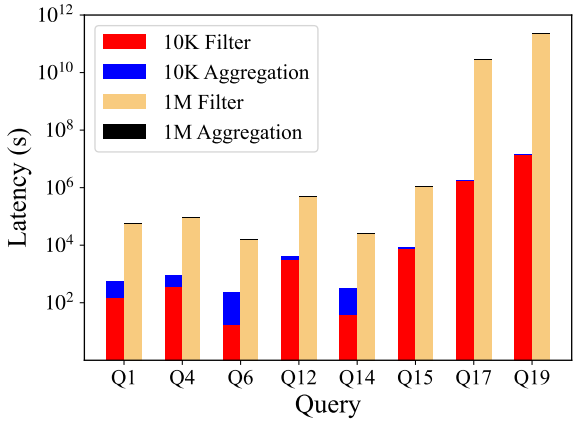


Figure 9: Latency performance of HE^{3}DB on various TPC-H benchmark over 10K rows and estimated latency on 1M rows.

we test a variety of the TPC-H benchmark queries on our HE^{3}DB framework, and the results are outlined in Figure 9. As the figure shows, while the filtering and aggregation latency is somewhat balanced on smaller databases, the filtering latency becomes the main performance bottleneck as the size of the database gets larger. Overall, we can evaluate an end-to-end SQL query over a 10K-row database within 241 seconds.

For the evaluation of the private decision tree, we provide a brief summary of the evaluation results obtained over multiple decision tree models under a computation outsourcing setting, where both the inputs and the model weights of the decision trees are ciphertexts. To enable fair comparisons, we use the same datasets as in [35], and reworked the implementation of [35] such that the solution in [35] can also operate over encrypted inputs and model weights. While detailed comparison results can be found in the appendix, we achieve 12x–326x faster evaluation speed than [35] over a wide range of datasets and input sizes. Thus, by adding the powerful online data analysis capability to the expressive query evaluation framework, HE^{3}DB is able to significantly improve the efficiency, usability and practical applicability of outsourcing databases securely.

7 CONCLUSIONS

In this work, we propose HE^{3}DB, an FHE-based encrypted database framework that enables expressive SQL statements to be efficiently queried on large-scale encrypted DBMS. We observe that a new FHE infrastructure needs to be developed to meet the usability, security, and efficiency demands from the encrypted DBMS. By designing comparison, aggregation, and conversion operators specifically for encrypted databases, we can outperform the best-known FHE algorithms on all DB-related task benchmarks. In particular, we are able to achieve 41×–299× reduction in query latency against the state-of-the-art FHE-based encrypted database solution on a 1M-record encrypted database.

ACKNOWLEDGMENTS

We thank the anonymous reviewers and shepherds for their valuable comments and helpful feedback. This work was partially supported by National Natural Science Foundation of China through projects 62002006, 62202028, 62172025, U21B2021, 61972019, U2241213, and the CCF-Huawei Populus euphratica project.

REFERENCES

[1] Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, Satya Lokam, Daniele Micciancio, Dustin Moody, Travis Morrison, Amit Sahai, and Vinod Vaikuntanathan. 2018. *Homomorphic Encryption Security Standard.* Technical Report. HomomorphicEncryption.org.

[2] Martin R. Albrecht, Rachel Player, and Sam Scott. 2015. On the concrete hardness of Learning with Errors. *J. Math. Cryptol.* 9, 3 (2015), 169–203. http://www.degruyter.com/view/j/jmc.2015.9.issue-3/jmc-2015-0016/jmc-2015-0016.xml

[3] Alibaba-Gemini-Lab. [n. d.]. Pegasus: Bridging Polynomial and Non-polynomial Evaluations in Homomorphic Encryption. Retrieved October 19, 2022 from https://github.com/Alibaba-Gemini-Lab/OpenPEGASUS

[4] Asma Aloufi, Peizhao Hu, Yongsoo Song, and Kristin Lauter. 2021. Computing Blindfolded on Data Homomorphically Encrypted under Multiple Keys: A Survey. *ACM Computing Surveys (CSUR)* 54, 9 (2021), 1–37.

[5] Panagiotis Antonopoulos, Arvind Arasu, Kunal D Singh, Ken Eguro, Nitish Gupta, Rajat Jain, Raghav Kaushik, Hanuma Kodavalla, Donald Kossmann, Nikolas Ogg, et al. 2020. Azure SQL database always encrypted. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data.* 1511–1525.

[6] Toshinori Araki, Assi Barak, Jun Furukawa, Tamar Lichter, Yehuda Lindell, Ariel Nof, Kazuma Ohara, Adi Watzman, and Or Weinstein. 2017. Optimized honest-majority MPC for malicious adversaries—breaking the 1 billion-gate

per second barrier. In *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 843–862.

[7] Arvind Arasu, Ken Eguro, Manas Joglekar, Raghav Kaushik, Donald Kossmann, and Ravi Ramamurthy. 2015. Transaction processing on confidential data using cipherbase. In *2015 IEEE 31st International Conference on Data Engineering*. IEEE, 435–446.

[8] AWS. 2023. Machine Learning on AWS. https://aws.amazon.com/machine-learning/?nc2=h_ql_sol_use_ml. Accessed: 2023-01-01.

[9] Azure. 2023. Azure Machine Learning. https://azure.microsoft.com/en-us/products/machine-learning/. Accessed: 2023-01-01.

[10] Youngjin Bae, Jung Hee Cheon, Wonhee Cho, Jaehyung Kim, and Taekyung Kim. 2022. META-BTS: Bootstrapping Precision Beyond the Limit. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi (Eds.). ACM, 223–234. https://doi.org/10.1145/3548606.3560696

[11] Youngjin Bae, Jung Hee Cheon, Wonhee Cho, Jaehyung Kim, and Taekyung Kim. 2022. META-BTS: Bootstrapping Precision Beyond the Limit. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi (Eds.). ACM, 223–234. https://doi.org/10.1145/3548606.3560696

[12] Maurice Bailleu, Jörg Thalheim, Pramod Bhatotia, Christof Fetzer, Michio Honda, and Kapil Vaswani. 2019. SPEICHER: Securing LSM-based Key-Value Stores using Shielded Execution. In *17th USENIX Conference on File and Storage Technologies (FAST 19)*. 173–190.

[13] Sumeet Bajaj and Radu Sion. 2011. TrustedDB: a trusted hardware based database with privacy and data confidentiality. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. 205–216.

[14] Kenneth E. Batcher. 1968. Sorting Networks and Their Applications. In *American Federation of Information Processing Societies: AFIPS Conference Proceedings: 1968 Spring Joint Computer Conference, Atlantic City, NJ, USA, 30 April - 2 May 1968*. Thomson Book Company, Washington D.C., 307–314. https://doi.org/10.1145/1468075.1468121

[15] Johes Bater, Gregory Elliott, Craig Eggen, Satyender Goel, Abel Kho, and Jennie Rogers. 2016. SMCQL: Secure querying for federated databases. *arXiv preprint arXiv:1606.06808* (2016).

[16] Song Bian, Dur-e-Shahwar Kundi, Kazuma Hirozawa, Weiqiang Liu, and Takashi Sato. 2021. APAS: Application-Specific Accelerators for RLWE-Based Homomorphic Linear Transformations. *IEEE Trans. Inf. Forensics Secur.* 16 (2021), 4663–4678.

[17] Dmytro Bogatov, Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O'Neill. 2021. ϵpsolute: Efficiently Querying Databases While Providing Differential Privacy. In *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*. ACM, 2262–2276. https://doi.org/10.1145/3460120.3484786

[18] Jean-Philippe Bossuat, Christian Mouchet, Juan Ramón Troncoso-Pastoriza, and Jean-Pierre Hubaux. 2021. Efficient Bootstrapping for Approximate Homomorphic Encryption with Non-sparse Keys. In *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 12696)*. Springer, 587–617. https://doi.org/10.1007/978-3-030-77870-5_21

[19] Elette Boyle, Nishanth Chandran, Niv Gilboa, Divya Gupta, Yuval Ishai, Nishant Kumar, and Mayank Rathee. 2021. Function secret sharing for mixed-mode and fixed-point secure computation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 871–900.

[20] Zvika Brakerski. 2012. Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP. In *CRYPTO*. 868–886.

[21] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. 2012. (Leveled) fully homomorphic encryption without bootstrapping. In *ITCS*. 309–325.

[22] Carole Cadwalladr and Emma Graham-Harrison. 2018. Revealed: 50 million Facebook profiles harvested for Cambridge Analytica in major data breach. *The guardian* 17 (2018), 22.

[23] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. 2015. Leakage-abuse attacks against searchable encryption. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*. 668–679.

[24] David Cash, Stanislaw Jarecki, Charanjit Jutla, Hugo Krawczyk, Marcel-Cătălin Roşu, and Michael Steiner. 2013. Highly-scalable searchable symmetric encryption with support for boolean queries. In *Annual cryptology conference*. Springer, 353–373.

[25] Hao Chen, Ilaria Chillotti, and Yongsoo Song. 2019. Improved Bootstrapping for Approximate Homomorphic 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 (Lecture Notes in Computer Science, Vol. 11477)*. Springer, 34–54. https://doi.org/10.1007/978-3-030-17656-3_2

[26] Hao Chen, Wei Dai, Miran Kim, and Yongsoo Song. 2021. Efficient Homomorphic Conversion Between (Ring) LWE Ciphertexts. In *Applied Cryptography and Network Security (Lecture Notes in Computer Science, Vol. 12726)*. Springer, 460–479.

[27] Hao Chen, Zhicong Huang, Kim Laine, and Peter Rindal. 2018. Labeled PSI from Fully Homomorphic Encryption with Malicious Security. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*. ACM, 1223–1237.

[28] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. 2018. Bootstrapping for Approximate Homomorphic Encryption. In *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part I (Lecture Notes in Computer Science, Vol. 10820)*. Springer, 360–384. https://doi.org/10.1007/978-3-319-78381-9_14

[29] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. 2017. Homomorphic Encryption for Arithmetic of Approximate Numbers. In *ASIACRYPT*. 409–437.

[30] Jung Hee Cheon, Dongwoo Kim, Duhyeong Kim, Hun-Hee Lee, and Keewoo Lee. 2019. Numerical Method for Comparison on Homomorphically Encrypted Numbers. In *ASIACRYPT*, Vol. 11922. Springer, 415–445.

[31] Jung Hee Cheon, Miran Kim, and Myungsun Kim. 2016. Optimized Search-and-Compute Circuits and Their Application to Query Evaluation on Encrypted Data. *IEEE Trans. Inf. Forensics Secur.* 11, 1 (2016), 188–199. https://doi.org/10.1109/TIFS.2015.2483486

[32] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. 2020. TFHE: Fast Fully Homomorphic Encryption Over the Torus. *J. Cryptol.* 33, 1 (2020), 34–91.

[33] Ilaria Chillotti, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap. 2021. Improved Programmable Bootstrapping with Larger Precision and Efficient Arithmetic Circuits for TFHE. In *ASIACRYPT*. 670–699.

[34] Google Cloud. 2023. Cloud SQL. https://cloud.google.com/sql/. Accessed: 2023-01-01.

[35] Kelong Cong, Debajyoti Das, Jeongeun Park, and Hilder V. L. Pereira. 2022. SortingHat: Efficient Private Decision Tree Evaluation via Homomorphic Encryption and Transciphering. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*. ACM, 563–577. https://doi.org/10.1145/3548606.3560702

[36] Kelong Cong, Radames Cruz Moreno, Mariana Botelho da Gama, Wei Dai, Ilia Iliashenko, Kim Laine, and Michael Rosenberg. 2021. Labeled psi from homomorphic encryption with reduced computation and communication. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 1135–1150.

[37] Henry Corrigan-Gibbs and Dan Boneh. 2017. Prio: Private, Robust, and Scalable Computation of Aggregate Statistics. In *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation* (Boston, MA, USA). 259–282.

[38] KU Leuven COSIC. [n. d.]. Private decision tree evaluation via Homomorphic Encryption and Transciphering. Retrieved November 30, 2022 from https://github.com/KULeuven-COSIC/SortingHat

[39] Victor Costan and Srinivas Devadas. 2016. Intel SGX explained. *Cryptology ePrint Archive* (2016).

[40] Transaction Processing Performance Council. 2022. *TPC BENCHMARK^{TM} H Standard Specification*. Technical Report. San Francisco,CA.

[41] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. 2006. Searchable symmetric encryption: improved definitions and efficient constructions. In *Proceedings of the 13th ACM conference on Computer and communications security*. 79–88.

[42] Emma Dauterman, Mayank Rathee, Raluca Ada Popa, and Ion Stoica. 2022. Waldo: A Private Time-Series Database from Function Secret Sharing. In *43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022*. IEEE, 2450–2468. https://doi.org/10.1109/SP46214.2022.9833611

[43] Ioannis Demertzis, Dimitrios Papadopoulos, Charalampos Papamanthou, and Saurabh Shintre. 2020. SEAL: Attack mitigation for encrypted databases via adjustable leakage. In *29th USENIX Security Symposium (USENIX Security 20)*. 2433–2450.

[44] Léo Ducas and Daniele Micciancio. 2015. FHEW: Bootstrapping Homomorphic Encryption in Less Than a Second. In *EUROCRYPT*. 617–640.

[45] Saba Eskandarian and Matei Zaharia. 2019. ObliDB: Oblivious Query Processing for Secure Databases. *Proc. VLDB Endow.* 13, 2 (2019), 169–183. https://doi.org/10.14778/3364324.3364331

[46] Junfeng Fan and Frederik Vercauteren. 2012. Somewhat Practical Fully Homomorphic Encryption. *IACR Cryptol. ePrint Arch.* (2012), 144. http://eprint.iacr.org/2012/144

[47] Erhu Feng, Xu Lu, Dong Du, Bicheng Yang, Xueqiang Jiang, Yubin Xia, Binyu Zang, and Haibo Chen. 2021. Scalable Memory Protection in the PENGLAI Enclave. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*. 275–294.

[48] Benjamin Fuller, Mayank Varia, Arkady Yerukhimovich, Emily Shen, Ariel Hamlin, Vijay Gadepally, Richard Shay, John Darby Mitchell, and Robert K Cunningham. 2017. Sok: Cryptographically protected database search. In *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 172–191.

[49] Craig Gentry, Amit Sahai, and Brent Waters. 2013. Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based. In *CRYPTO*. 75–92.

[50] Paul Grubbs, Marie-Sarah Lacharité, Brice Minaud, and Kenneth G Paterson. 2018. 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*. 315–331.

[51] Paul Grubbs, Marie-Sarah Lacharité, Brice Minaud, and Kenneth G Paterson. 2019. Learning to reconstruct: Statistical learning theory and encrypted database attacks. In *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1067–1083.

[52] Zichen Gui, Oliver Johnson, and Bogdan Warinschi. 2019. Encrypted Databases: New Volume Attacks against Range Queries. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 361–378.

[53] Antonio Guimarães, Edson Borin, and Diego F. Aranha. 2021. Revisiting the functional bootstrap in TFHE. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2021, 2 (2021), 229–253. https://doi.org/10.46586/tches.v2021.i2.229-253

[54] Timon Hackenjos, Florian Hahn, and Florian Kerschbaum. 2020. SAGMA: Secure Aggregation Grouped by Multiple Attributes. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*. ACM, 587–601. https://doi.org/10.1145/3318464.3380569

[55] Shai Halevi, Yuriy Polyakov, and Victor Shoup. 2019. An Improved RNS Variant of the BFV Homomorphic Encryption Scheme. In *CT-RSA*. 83–105.

[56] Shai Halevi and Victor Shoup. 2014. Algorithms in HElib. In *CRYPTO*. 554–571.

[57] Shai Halevi and Victor Shoup. 2020. Design and implementation of HElib: a homomorphic encryption library. *IACR Cryptol. ePrint Arch.* (2020).

[58] Kyoohyung Han, Minki Hhan, and Jung Hee Cheon. 2019. Improved Homomorphic Discrete Fourier Transforms and FHE Bootstrapping. *IEEE Access* 7 (2019), 57361–57370.

[59] Kyoohyung Han and Dohyeong Ki. 2020. Better Bootstrapping for Approximate Homomorphic Encryption. In *Topics in Cryptology - CT-RSA 2020 - The Cryptographers' Track at the RSA Conference 2020, San Francisco, CA, USA, February 24-28, 2020, Proceedings (Lecture Notes in Computer Science, Vol. 12006)*. Springer, 364–390. https://doi.org/10.1007/978-3-030-40186-3_16

[60] Harold V. Henderson, Friedrich Pukelsheim, and Shayle R. Searle. 1983. On the history of the kronecker product. *Linear and Multilinear Algebra* 14, 2 (1983), 113–120. https://doi.org/10.1080/03081088308817548

[61] C. A. R. Hoare. 1962. Quicksort. *Comput. J.* 5, 1 (01 1962), 10–16. https://doi.org/10.1093/comjnl/5.1.10 arXiv:https://academic.oup.com/comjnl/article-pdf/5/1/10/1111445/050010.pdf

[62] Zhicong Huang, Wen-jie Lu, Cheng Hong, and Jiansheng Ding. 2022. Cheetah: Lean and Fast Secure Two-Party Deep Neural Network Inference. In *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022*. USENIX Association, 809–826. https://www.usenix.org/conference/usenixsecurity22/presentation/huang-zhicong

[63] Yuval Ishai, Eyal Kushilevitz, Steve Lu, and Rafail Ostrovsky. 2016. Private large-scale databases with distributed searchable symmetric encryption. In *Cryptographers' Track at the RSA Conference*. Springer, 90–107.

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

[65] Simon Johnson, Raghunandan Makaram, Amy Santoni, and Vinnie Scarlata. 2021. Supporting intel sgx on multi-socket platforms. https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/supporting-intel-sgx-on-mulit-socket-platforms.pdf.

[66] Charanjit S. Jutla and Nathan Manohar. 2022. Sine Series Approximation of the Mod Function for Bootstrapping of Approximate HE. In *Advances in Cryptology - EUROCRYPT 2022 - 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 - June 3, 2022, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 13275)*. Springer, 491–520. https://doi.org/10.1007/978-3-031-06944-4_17

[67] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha P. Chandrakasan. 2018. GAZELLE: A Low Latency Framework for Secure Neural Network Inference. In *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*. USENIX Association, 1651–1669.

[68] Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O'Neill. 2016. Generic Attacks on Secure Outsourced Databases. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*. ACM, 1329–1340. https://doi.org/10.1145/2976749.2978386

[69] Alhassan Khedr, Glenn Gulak, and Vinod Vaikuntanathan. 2015. SHIELD: scalable homomorphic implementation of encrypted data-classifiers. *IEEE Trans. Comput.* 65, 9 (2015), 2848–2858.

[70] Taehoon Kim, Joongun Park, Jaewook Woo, Seungheun Jeon, and Jaehyuk Huh. 2019. Shieldstore: Shielded in-memory key-value storage with sgx. In *Proceedings of the Fourteenth EuroSys Conference 2019*. 1–15.

[71] Y.A.M. Kortekaas. 2020. Access Pattern Hiding Aggregation over Encrypted Databases. http://essay.utwente.nl/83874/

[72] Joon-Woo Lee, Eunsang Lee, Yongwoo Lee, Young-Sik Kim, and Jong-Seon No. 2021. High-Precision Bootstrapping of RNS-CKKS Homomorphic Encryption Using Optimal Minimax Polynomial Approximation and Inverse Sine Function. In *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 12696)*. Springer, 618–647. https://doi.org/10.1007/978-3-030-77870-5_22

[73] Yongwoo Lee, Joon-Woo Lee, Young-Sik Kim, Yongjune Kim, Jong-Seon No, and HyungChul Kang. 2022. High-Precision Bootstrapping for Approximate Homomorphic Encryption by Error Variance Minimization. In *Advances in Cryptology - EUROCRYPT 2022 - 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 - June 3, 2022, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 13275)*. Springer, 551–580. https://doi.org/10.1007/978-3-031-06944-4_19

[74] John Liagouris, Vasiliki Kalavri, Muhammad Faisal, and Mayank Varia. 2021. Secrecy: Secure collaborative analytics on secret-shared data. *arXiv preprint arXiv:2102.01048* (2021).

[75] Zeyu Liu, Daniele Micciancio, and Yuriy Polyakov. 2021. Large-Precision Homomorphic Sign Evaluation using FHEW/TFHE Bootstrapping. *IACR Cryptol. ePrint Arch.* (2021), 1337. https://eprint.iacr.org/2021/1337

[76] Jack Lu. 2019. Assessing the cost, legal fallout of Capital One data breach. *Law360 Expert Analysis* (2019).

[77] Wen-jie Lu, Zhicong Huang, Cheng Hong, Yiping Ma, and Hunter Qu. 2021. PEGASUS: Bridging Polynomial and Non-polynomial Evaluations in Homomorphic Encryption. In *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*. IEEE, 1057–1073. https://doi.org/10.1109/SP40001.2021.00043

[78] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. 2010. On ideal lattices and learning with errors over rings. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 1–23.

[79] Kotaro Matsuoka, Ryotaro Banno, Naoki Matsumoto, Takashi Sato, and Song Bian. 2021. Virtual Secure Platform: A Five-Stage Pipeline Processor over TFHE. In *30th USENIX Security Symposium (USENIX Security 21)*. 4007–4024.

[80] Robert Mayans. 2006. The chebyshev equioscillation theorem. *Journal of Online Mathematics and Its Applications* 6 (2006).

[81] Pratyush Mishra, Rishabh Poddar, Jerry Chen, Alessandro Chiesa, and Raluca Ada Popa. 2018. Oblix: An Efficient Oblivious Search Index. In *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*. IEEE Computer Society, 279–296. https://doi.org/10.1109/SP.2018.00045

[82] MonogoDB. 2023. Application-Driven Analytics. https://www.mongodb.com/use-cases/analytics. Accessed: 2023-01-01.

[83] Pratyay Mukherjee and Daniel Wichs. 2016. Two round multiparty computation via multi-key FHE. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 735–763.

[84] openfheorg. [n. d.]. OpenFHE - Open-Source Fully Homomorphic Encryption Library. Retrieved December 31, 2022 from https://github.com/openfheorg/openfhe-development

[85] Antonis Papadimitriou, Ranjita Bhagwan, Nishanth Chandran, Ramachandran Ramjee, Andreas Haeberlen, Harmeet Singh, Abhishek Modi, and Saikrishna Badrinarayanan. 2016. Big Data Analytics over Encrypted Datasets with Seabed. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation* (Savannah, GA, USA) *(OSDI'16)*. USENIX Association, USA, 587–602.

[86] Chris Peikert and Sina Shiehian. 2016. Multi-key FHE from LWE, revisited. In *Theory of cryptography conference*. Springer, 217–238.

[87] Rishabh Poddar, Tobias Boelter, and Raluca Ada Popa. [n. d.]. Arx: An Encrypted Database using Semantically Secure Encryption. *Proceedings of the VLDB Endowment* 12, 11 ([n. d.]).

[88] Rishabh Poddar, Sukrit Kalra, Avishay Yanai, Ryan Deng, Raluca Ada Popa, and Joseph M Hellerstein. 2021. Senate: A Maliciously-Secure MPC Platform for Collaborative Analytics. In *30th USENIX Security Symposium (USENIX Security 21)*. 2129–2146.

[89] Raluca A. Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. 2011. CryptDB: protecting confidentiality with encrypted query processing. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles 2011, SOSP 2011, Cascais, Portugal, October 23-26, 2011*. ACM, 85–100. https://doi.org/10.1145/2043556.2043566

[90] Christian Priebe, Kapil Vaswani, and Manuel Costa. 2018. EnclaveDB: A secure database using SGX. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 264–278.

[91] Oded Regev. 2009. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM* 56, 6 (2009), 34.

[92] Eugene Y Remez. 1934. Sur la détermination des polynômes d'approximation de degré donnée. *Comm. Soc. Math. Kharkov* 10, 196 (1934), 41–63.

Algorithm 7: `BlindRotate`

Input : A Test polynomial $\tilde{v} = v_0 + v_1 x + ... + v_{N-1}x^{N-1}$.
Input : An LWE ciphertext $ct = \mathsf{LWE}_{\mathbf{s}}^{n,q}(m) = (b, \mathbf{a})$.
Input : A bootstrapping key **BK**, where
$\mathbf{BK}[i] = \mathsf{RGSW}_{\tilde{s}}^{N,Q}(\mathbf{s}_i)$, for $i \in [1, n]$
Output: A LWE ciphertext $ct_O = \mathsf{LWE}_{\tilde{s}}^{N,Q}(v_m)$.

1 $b \leftarrow \left\lceil \frac{2N}{q} b \right\rfloor$ and $\mathbf{a} \leftarrow \left\lceil \frac{2N}{q} \mathbf{a}_i \right\rfloor$.
2 Initialize $ACC \leftarrow X^{-b} \cdot (\tilde{v}, 0) \in \mathsf{RLWE}_{\tilde{s}}^{N,Q}(.)$.
3 **for** $i = 0$ **to** $n - 1$ **do**
4 $\quad ACC \leftarrow \mathtt{CMUX}(\mathbf{BK}_i, X^{-\mathbf{a}[i]} \cdot ACC, ACC)$
5 $ct_O \leftarrow \mathtt{RLWEtoLWEs}(ACC)[0]$
Return: $ct_O = \mathsf{LWE}_{\tilde{\mathbf{s}}}^{N,Q}(v_m)$

Algorithm 8: κ-bit `PBS-HMSB`

Input : A LWE ciphertext $ct_I = \mathsf{LWE}_{\mathbf{s}}^{n,q}(m) = (b, \mathbf{a})$,
where $m \; in [0, 2^k - 1]$.
Input : A constant $\mu = q/4$.
Input : A bootstrapping key **BK**, where
$\mathbf{BK}[i] = \mathsf{RGSW}_{\mathbf{s}}^{n,q}(\mathbf{s}_i)$, for $i \in [1, n]$
Output: A LWE ciphertext $ct_O = \mathsf{LWE}_{\mathbf{s}}^{n,q}(\mathtt{MSB}(m))$.

1 $\tilde{v} \leftarrow \mu \cdot (1 + x^2 + x^3 + ... + x^{N-1})$
2 $ACC \leftarrow \mathtt{BlindRotate}(\tilde{v}, ct_I, \mathbf{BK})$
3 $ct_O = (b', \mathbf{a}') \leftarrow \mathtt{RLWEtoLWEs}(ACC)[0]$
Return: $ct_O = (b' + \mu, \mathbf{a}') = \mathsf{LWE}_{\mathbf{s}}^{n,q}(\mathtt{MSB}(m))$

A PROOF FOR LEMMA 4.1 AND LEMMA 4.2

Proof. **Lemma 4.1** First, we prove the correctness of the Algorithm 3. If the comparison operator is $\geq$, the line 2 in Algorithm 3 results $\mathsf{LWE}(\mathtt{NOT}(\mathtt{MSB}(a-b))$. the predicate $a \geq b$ is equal to $\mathtt{MSB}(a-b) == 0$. If $a \geq b$, the predicate is True, and $\mathsf{LWE}(\mathtt{NOT}(\mathtt{MSB}(a-b)) = \mathsf{LWE}(1)$. If $a < b$, the predicate is False, and $\mathsf{LWE}(\mathtt{NOT}(\mathtt{MSB}(a-b)) = \mathsf{LWE}(0)$. Other comparison operators are the same. The number of `PBS` in Algorithm 3 includes the `PBS` in `HMSB` and `HomGate`. From Algorithm 2, the number of `PBS` evaluated in a tk-bit ciphertext MSB extraction is $2 \times (t-1) + 1 = 2t - 1$. Meanwhile, one `HomGate` includes one `PBS` as we discussed in Section 2.2. Thus, the number of `PBS` in Algorithm 3 is $2t - 1$ when the comparison operator is $>, <$ and $2t$ when the comparison operator is $\geq, \leq, ==, \neq$. □

Proof. **Lemma 4.2** In Algorithm 5, the first for loop (line 2-4) includes $|\mathcal{T}|_{\text{row}}$ `LWEtoRGSW` and `CMUX`, the second for loop (line 5-7) contains $2^{\log_2 |\mathcal{T}|_{\text{row}} - 1} + 2^{\log_2 |\mathcal{T}|_{\text{row}} - 2} + ... + 2^0 = |\mathcal{T}|_{\text{row}} - 1$ `HomMIN`. As shown in Algorithm 4, each `HomMIN` has one `HomComp`, one `LWEtoRGSW` and one `CMUX`. Therefore, the total number of `CMUX` gates is $|\mathcal{T}|_{\text{row}} + |\mathcal{T}|_{\text{row}} - 1 = 2|\mathcal{T}|_{\text{row}} - 1$, the whole number of `LWEtoRGSW` operators is $|\mathcal{T}|_{\text{row}} + |\mathcal{T}|_{\text{row}} - 1 = 2|\mathcal{T}|_{\text{row}} - 1$, the number of `HomComp` is $|\mathcal{T}|_{\text{row}} - 1$. □

B COMPLEXITY COMPARISON BETWEEN HE^{3}DB AND [75]

Here, we take a closer look at the detailed complexity analysis between HE^{3}DB and [75]. Although both the methods of HE^{3}DB and [75] achieve high-precision MSB extraction by removing the lower bits of the ciphertext, there is a fundamental difference between the two approaches. For a message m encrypted in a ciphertext $\mathbf{c} = (\mathbf{a}, b)$, [75] clear the the lower bits of m through the HomFloor function, which set the least $k = \log(q/\alpha)$ bits of m to be zeroes. Then, HomSign repeatedly calls HomFloor to clear all the lower bits of m. In contrast, in our method, we multiply $(\mathbf{a}, b)$ by a factor of 2^{k-1}, and take out a k-bit segment of m using TFHE PBS. The difference here is essential: [75] takes out the least k-bit segment of m by taking $\mathbf{c} = (\mathbf{a}, b) \bmod q$. while HE^{3}DB take out a k-bit segment of m by multiplying $(\mathbf{a}, b)$ with 2^{k-1} and then reduce the result by Q (e.g., the real ciphertext modulus). In practice, the parameter q in [75] can only be as large as $2N = 4096$ in most AP/GINX type bootstrapping procedures for efficiency reasons. Meanwhile, $\alpha = 2^8$ is required to separate m from the LWE errors

[93] Xuanle Ren, Le Su, Zhen Gu, Sheng Wang, Feifei Li, Yuan Xie, Song Bian, Chao Li, and Fan Zhang. 2022. HEDA: Multi-Attribute Unbounded Aggregation over Homomorphically Encrypted Database. *Proc. VLDB Endow.* 16, 4 (2022), 601–614. https://www.vldb.org/pvldb/vol16/p601-gu.pdf

[94] SEAL 2021. Microsoft SEAL (release 3.7). https://github.com/Microsoft/SEAL. Microsoft Research, Redmond, WA..

[95] Zihao Shan, Kui Ren, Marina Blanton, and Cong Wang. 2018. Practical secure computation outsourcing: A survey. *ACM Computing Surveys (CSUR)* 51, 2 (2018), 1–40.

[96] Nigel P. Smart and Frederik Vercauteren. 2014. Fully homomorphic SIMD operations. *Des. Codes Cryptogr.* 71, 1 (2014), 57–81.

[97] Emil Stefanov, Marten Van Dijk, Elaine Shi, T-H Hubert Chan, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. 2018. Path ORAM: an extremely simple oblivious RAM protocol. *Journal of the ACM (JACM)* 65, 4 (2018), 1–26.

[98] G Edward Suh, Charles W O'Donnell, and Srinivas Devadas. 2007. Aegis: A single-chip secure processor. *IEEE Design & Test of Computers* 24, 6 (2007), 570–580.

[99] Lawrence J Trautman and Peter C Ormerod. 2017. Corporate Directors' and Officers' Cybersecurity Standard of Care: The Yahoo Data Breach. *American University Law Review* 66, 5 (2017), 3.

[100] Alexander Viand, Christian Knabenhans, and Anwar Hithnawi. 2023. Verifiable Fully Homomorphic Encryption. *CoRR* abs/2301.07041 (2023). https://doi.org/10.48550/arXiv.2301.07041 arXiv:2301.07041

[101] Dhinakaran Vinayagamurthy, Alexey Gribov, and Sergey Gorbunov. 2019. StealthDB: a Scalable Encrypted Database with Full SQL Query Support. *Proc. Priv. Enhancing Technol.* 2019, 3 (2019), 370–388.

[102] virtualsecureplatform. [n. d.]. TFHEpp. Retrieved October 19, 2022 from https://github.com/virtualsecureplatform/TFHEpp

[103] Nikolaj Volgushev, Malte Schwarzkopf, Ben Getchell, Mayank Varia, Andrei Lapets, and Azer Bestavros. 2019. Conclave: secure multi-party computation on big data. In *Proceedings of the Fourteenth EuroSys Conference 2019*. 1–18.

[104] Chenghong Wang, Johes Bater, Kartik Nayak, and Ashwin Machanavajjhala. 2022. IncShrink: Architecting Efficient Outsourced Databases using Incremental MPC and Differential Privacy. In *SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*. ACM, 818–832. https://doi.org/10.1145/3514221.3526151

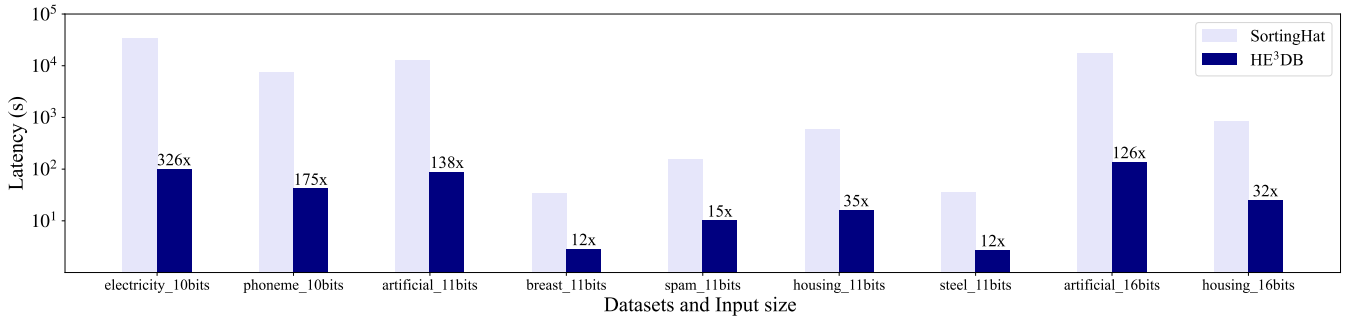
[105] Sheng Wang, Yiran Li, Huorong Li, Feifei Li, Chengjin Tian, Le Su, Yanshan Zhang, Yubing Ma, Lie Yan, Yuanyuan Sun, et al. 2022. Operon: An encrypted database for ownership-preserving data management. *Proceedings of the VLDB Endowment* 15, 12 (2022), 3332–3345.

[106] Yilei Wang and Ke Yi. 2021. Secure Yannakakis: Join-Aggregate Queries over Private Data. In *Proceedings of the 2021 International Conference on Management of Data*. 1969–1981.

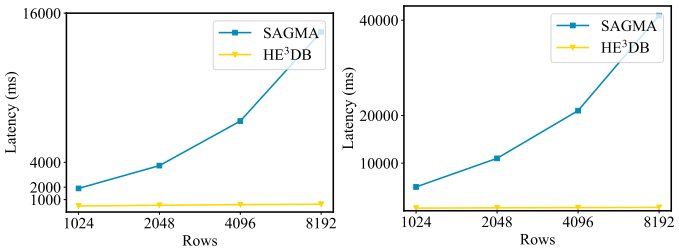
[107] Jean-Luc Watson, Sameer Wagh, and Raluca Ada Popa. 2022. Piranha: A GPU Platform for Secure Computation. In *31st USENIX Security Symposium (USENIX Security 22)*. 827–844.

[108] Xun Yi, Mohammed Golam Kaosar, Russell Paulet, and Elisa Bertino. 2012. Single-database private information retrieval from fully homomorphic encryption. *IEEE Transactions on Knowledge and Data Engineering* 25, 5 (2012), 1125–1134.

[109] Wenting Zheng, Ankur Dave, Jethro G Beekman, Raluca Ada Popa, Joseph E Gonzalez, and Ion Stoica. 2017. Opaque: An oblivious and encrypted distributed analytics platform. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. 283–298.



Appendix Figure A1: Running time of our PTDE algorithm compared to SortingHat [35]



Appendix Figure A2: Comparison with SAGMA [54] on aggregation function COUNT and SUM.

e, resulting in a maximum $k = 4$. Moreover, without the multiplication between $(\mathbf{a}, b)$ with 2^{k-1}, [75] has to scale down to a smaller ciphertext modulus in each iteration to keep the pre-removed k bits in the least $\log q$ bits, which results in more complicated noise management. Consequently, HE^{3}DB has fewer parameter requirements and achieves larger k (e.g., $k = 5$) than [75] in each iteration. When N and Q become large, we may see further differences as the methods of determining k between HE^{3}DB and [75].

C ADDITIONAL COMPARISONS AGAINST SAGMA

As SAGMA [54] only uses HE in its aggregation process, we only compare HE^{3}DB with SAGMA on the arithmetic aggregation tasks. As shown in Figure A2, we can obtain 23× and 58× faster in COUNT and SUM operations on a 8192-row database, respectively.

D EVALUATING PRIVATE OUTSOURCED DECISION TREE

A private decision tree model $\mathcal{CM}$ includes the encryption of threshold ct_i, the attribute index a_i and the encryption of label cw_i. Algorithm 9 shows our PDTE algorithm, which involves the following steps:

• **Line 1-2, Initialization**: Server initializes the value of each node as the encryption of 0, denoted by v_j for all $j \in \{0, ..., \mathfrak{d} - 2\}$, except the root, denoted by v_0, which contains the encryption of 1.

• **Line 3-5, Comparison**: For each decision node i, server computes the Homomorphic Comparison function $cb_i \leftarrow [cr_{a_v} \geq ct_i]$,

Algorithm 9: PDTEval based on HomComp

Input : Private decision tree Model $\mathcal{CM}$=$\{(ct_0, ..., ct_{2\mathfrak{d}-2}), (a_0, ..., a_{2\mathfrak{d}-2}), (cw_0, ..., cw_{\mathfrak{d}-1})\}$
Input : Query results $\mathcal{CR}$= $(cr_0, cr_1, ..., cr_{|G|})$
Output: $\mathfrak{d}$ values of leaves: $z_0, ..., z_{\mathfrak{d}-1}$

1 Initialize root value v_0 as $\mathrm{LWE}_{\mathbf{s}}^{n,q}(1)$
2 Initialize other node value v_j as $\mathrm{LWE}_{\mathbf{s}}^{n,q}(0), j \in \{1, ..., 2\mathfrak{d} - 2\}$
3 **for** $i = 0$ **to** $2\mathfrak{d} - 2$ **do**
4 $\quad cb_i \leftarrow \mathtt{HomComp}(cr_{a_i}, ct_i, \geq)$
5 **for** $i = 1$ **to** $\log \mathfrak{d}$ **do**
6 $\quad$ **for** $j = 0$ **to** $2^{i-1} - 1$ **do**
7 $\quad\quad v_{2^i+2*j} \leftarrow \mathtt{HomGate}(cb_{2^{i-1}-1+j}, v_{2^{i-1}-1+j}, \mathrm{AND})$
$\quad\quad v_{2^i+2*j-1} \leftarrow v_{2^{i-1}-1+j} - v_{2^i+2*j}$
8 **for** $i = 0$ **to** $\mathfrak{d} - 1$ **do**
9 $\quad z_i \leftarrow v_{2^{\log \mathfrak{d}}-1+i}$
Return: $z_0, ..., z_{\mathfrak{d}-1}$

we denote the controller bit of each node by cb_i corresponding to the node i for $i \in \{0, ..., \mathfrak{d} - 2\}$.

• **Line 6-10, Traversal**: Then the server moves root value v_0 from the root to a desired leaf. The path depends on the controller bits on each node. Take node i as an example, if node i has value v_i and the controller bit of i is cb_i, the value of the left child node (denoted by v_{Lci})and the right child node (denoted by v_{Rci}) are compute as $v_{Rci} = \mathtt{HomAND}(cb_i, v_i)$, $v_{Lci} = v_i - v_{Rci}$. After traversal, the value of the root (which is the encryption of 1) is copied to the desired place.

Finally, only one leaf has a value which is the encryption of 1, we can easily get this unique label value.

Evaluation Results: As shown in Figure A1, since SortingHat [35] does not target on evaluating private decision under a secure computation outsourcing setting, we can achieve up to 300× speedup using Algorithm 9.