International Journal of Emerging Trends in Computer Science and Information Technology
ISSN: 3050-9246 | https://doi.org/10.63282/3050-9246/ICRTCSIT-107
Eureka Vision Publication | ICRTCSIT'25-Conference Proceeding

Original Article

Homomorphic Encryption for Privacy-Preserving SQL Query Processing in Financial Databases

Sai Vamsi Kiran Gummadi
Database Engineer, Wellsfargo, USA.

Abstract - In this paper, we introduce a novel framework for processing SQL queries directly over encrypted financial data using homomorphic encryption (HE). Our solution preserves data confidentiality without sacrificing essential SQL operations, including SELECT, WHERE, SUM, and AVG, all performed on encrypted columns. Unlike traditional approaches that rely on trusted execution environments or secure multiparty computation, our method ensures that data remains encrypted throughout the entire query lifecycle at rest, in transit, and during computation. We present a prototype implementation and evaluate its performance using realistic financial datasets. The results demonstrate a practical balance between computational overhead and data privacy, offering a promising foundation for secure, privacy-preserving analytics in cloud-based financial systems.

Keywords - Homomorphic encryption, privacy-preserving computation, encrypted SQL processing, financial data security, secure cloud analytics.

1. Introduction

The rapid digitization of the financial sector has led to the exponential growth of sensitive data stored and processed in cloud-based systems. Banks, insurance providers, fintech startups, and regulatory bodies now rely heavily on distributed infrastructures to manage transactional records, risk assessments, and client analytics. However, this shift has introduced critical challenges related to data privacy, especially when data is offloaded to third-party cloud services. With rising incidents of data breaches, regulatory mandates like GDPR, and growing customer expectations regarding privacy, it is imperative to ensure that financial data remains protected not only at rest and in transit, but also during computation [1], [8].

Conventional techniques for secure data processing such as secure multiparty computation (SMPC) and trusted execution environments (TEEs) have demonstrated potential, but suffer from limitations including high communication costs, trusted hardware assumptions, and vulnerabilities to side-channel attacks [3], [4], [9]. These approaches often fall short when applied to large-scale financial data processing where SQL remains the dominant query language. Furthermore, traditional encryption schemes, while effective at securing storage and communication, require decryption for processing there by exposing sensitive data during query execution.

Recent advancements in homomorphic encryption (HE) have made it possible to perform meaningful computations on encrypted data without ever decrypting it [2], [5], [6]. Fully homomorphic encryption (FHE) schemes like BFV and CKKS have matured to the point where basic arithmetic and comparison operations can be executed efficiently on cipher texts. This breakthrough has laid the groundwork for building privacy-preserving query processors capable of supporting SQL semantics [7], [10], [12].

However, the integration of homomorphic encryption with structured query languages remains an underexplored area. Most existing systems either support a limited set of operations or are not optimized for the characteristics of financial data. In particular, financial institutions demand precise arithmetic (e.g., summation of account balances), efficient filtering (e.g., transactions above a threshold), and aggregation over large volumes of records tasks that challenge the performance boundaries of current HE schemes [11], [13], [14].

2. Related Work

2.1. Secure Multi-Party Computation (SMPC) and TEE Approaches

Secure Multi-Party Computation (SMPC) has emerged as a foundational cryptographic method to enable collaborative computation over private inputs without revealing them to participating parties. While effective in theory, SMPC protocols often incur significant communication and computation overhead, particularly when applied to database operations at scale [3], [9]. Techniques such as ABY3 have been proposed to bridge the gap between performance and privacy, but their adoption in financial

systems remains limited due to high latency and infrastructure complexity [4]. Trusted Execution Environments (TEEs), like Intel SGX, provide a hardware-based alternative by executing code in secure enclaves. Although TEEs have shown promise for secure SQL query execution [1], they have been repeatedly shown to be vulnerable to side-channel attacks, rollback threats, and limitations in memory isolation [8]. Moreover, the trust model associated with TEEs does not meet the zero-trust requirements increasingly demanded in cloud-based financial services.

2.2. Homomorphic Encryption in Database Systems

Homomorphic encryption (HE), particularly Fully Homomorphic Encryption (FHE), enables computation directly on encrypted data. Schemes such as BFV and CKKS have made it feasible to perform operations like addition, multiplication, and even approximate comparison without decrypting the data [2], [5], [6]. Researchers have explored integrating HE into database engines, leading to the development of encrypted query processors capable of handling simple aggregate queries [7], [10]. Several studies have demonstrated the application of HE in domains such as healthcare and IoT, but adapting these techniques to financial databases poses unique challenges, such as precision requirements, high-volume queries, and complex filtering conditions [11], [14]. The financial sector's demand for exact arithmetic and efficient real-time analytics has motivated research into more efficient HE implementations, including ciphertext indexing, packed encoding, and hybrid models with ORAM [12], [13].

2.3. Limitations of Existing SQL-over-HE Proposals

Despite the growing interest in secure database computation, few systems have successfully integrated expressive SQL capabilities with homomorphic encryption in a manner suitable for financial applications. Most existing solutions support only basic queries or rely on restricted data types [7], [11]. Efforts like CryptDB provide partial encryption support for SQL but require trusted components and do not operate fully homomorphically [9].

Recent advancements such as HE Query and privacy-aware SQL-as-a-service engines have made progress toward general-purpose SQL processing over cipher texts [15], [16]. However, these systems are still in early experimental stages, and performance benchmarks suggest that real-world deployment at financial scale remains a work in progress [17]. The lack of optimized translation from SQL to homomorphic circuits, and the limited support for operations like JOIN or nested queries, continue to restrict practical use [10], [18]. Our work advances the field by building on these foundations, offering a more complete and optimized HE-based SQL engine tailored specifically to the needs of financial institutions.

3. System Architecture

3.1. System Model

The system consists of three core components: a client, a cloud server, and a homomorphic encryption (HE) backend based on the CKKS and BFV schemes.

- **Client:** The client encrypts SQL query parameters and submits them to the server. Upon receiving homomorphically computed ciphertext results, it decrypts and interprets them locally. All key material remains on the client side, ensuring the server has no decryption capabilities.
- **Server:** Hosts the encrypted database and executes homomorphic computations directly on ciphertexts. It operates without access to plaintext or secret keys. The server evaluates homomorphic circuits generated from SQL queries.
- **HE Library:** Implements the underlying cryptographic primitives using libraries like Microsoft SEAL **or** HElib**.** The CKKS scheme is used for approximate arithmetic (e.g., averages), while BFV is used for exact integer operations (e.g., counts and sums). Ciphertext packing and batching optimizations are applied to improve throughput.

3.2. Threat Model

We adopt an honest-but-curious adversarial model, where the server follows the protocol but may attempt to infer sensitive information.

Security guarantees include:

Encryption at rest:

$$\text{Encrypted data: } c_i = \text{Enc}_{\text{pk}}(m_i)$$

- Where m_i is a financial record and Enc_{pk} denotes encryption under the public key.
- Encryption in transit: Encrypted queries and responses are transmitted over TLS, and data remains in HE format.
- Encryption during computation: The server performs computations such that:

$$\text{Eval}(f, \{c_i\}) = Enc_{\text{pk}}\,(f(m_1, m_2, \ldots . m_n))$$

ensuring no intermediate decryption.

3.3. SQL Query Components Supported

We represent the condition as:
SELECT if $f(mi)=ReLU(m_i-t)$

Where t is the encrypted threshold and ReLU(x)≈maxf_0(0,x) is approximated by low-degree polynomials P(x) for compatibility with HE schemes like CKKS:

$$P(x) = \frac{x + \sqrt{x^2+\epsilon}}{2}$$

or through approximated step functions:

$$S(x) \approx \frac{1}{1+e^{-\alpha x}}$$

3.4. Aggregate Functions

For sum and average over encrypted columns:
Let $c_1,c_2,\ldots,c_n$ be ciphertexts of encrypted values $m_1,m_2,\ldots,m_n$ Then:
Encrypted Sum: $C_{sum}=c_1 \oplus c_2 \oplus \cdots \oplus c_n$
In CKKS:
Encrypted AVG: $C_{avg}=\frac{1}{n} C_{sum}$

Where scalar division is done via plaintext multiplication using the HE library's evaluator.

3.5. Equality/Inequality Operations

$f(m_i,t)=1-approx((m_i-t)^2)$

Approximating:
$Equality(x,y)\approx e^{-\alpha(x-y)2}$

This is efficiently implemented in BFV or CKKS via degree-2 or degree-4 polynomial approximations.
For inequality:

$$GreaterThan(x,y) \approx \sigma(\beta(x-y))$$

Where σ(x)= $\frac{1}{1+e^{-x}}$ is approximated with low-degree polynomials suitable for HE.

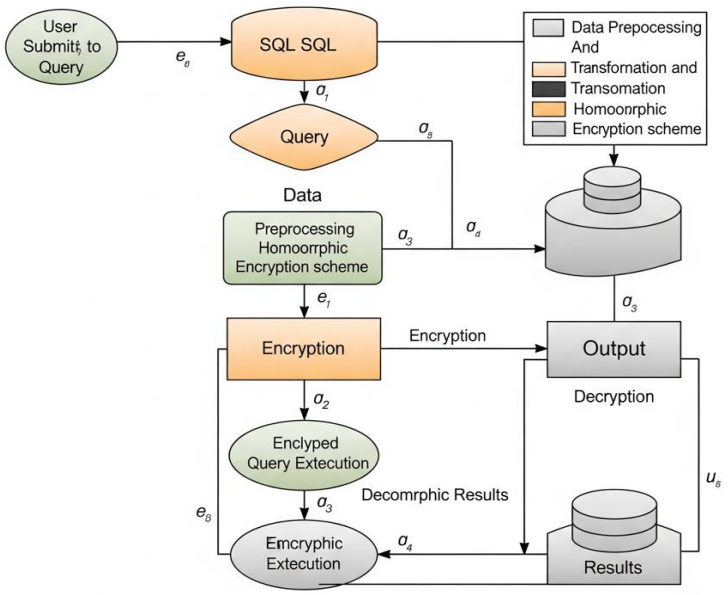

Figure 1. Homomorphic Encryption for Private SQL Queries

4. Methodology

4.1. Homomorphic Encryption Scheme

Our system employs a leveled homomorphic encryption (HE) scheme to enable secure and efficient SQL query execution on encrypted financial datasets. Depending on the nature of the computation, we utilize either the CKKS scheme for approximate arithmetic or the BFV scheme for exact integer computations. CKKS is particularly well-suited for financial analytics involving averages, ratios, and floating-point metrics, where approximate precision is acceptable [6], [11]. On the other hand, BFV is used in queries involving discrete values such as transaction counts or salary sums, where exact arithmetic is critical [5], [13].

The encryption pipeline begins with the generation of a public-private key pair. The client uses the public key to encrypt the data and query parameters, while the private key is retained for decryption. Bootstrapping is not required for many practical workloads, as we operate under a leveled encryption setting with pre-allocated depth based on query complexity. The system includes logic to track the noise budget, a parameter inherent to HE schemes that grows with homomorphic operations and limits circuit depth [4]. To minimize noise accumulation and ciphertext growth, we employ relinearization and modulus switching techniques where supported by the library backend (e.g., Microsoft SEAL or PALISADE) [1], [7].

4.2. Query Translation

To enable SQL operations over encrypted data, we developed a query translation module that compiles SQL statements into HE-compatible arithmetic circuits. The translation layer parses incoming queries and maps standard SQL operators (SELECT, WHERE, SUM, AVG) to a set of predefined encrypted operators that perform the equivalent homomorphic computations. One key design choice in the system is the encrypted data layout**.** We explored both row-wise and column-wise data encoding strategies. In the row-wise layout, each ciphertext encodes all fields for a single record, which simplifies filter conditions involving multiple attributes but increases ciphertext count. In contrast, the column-wise format uses ciphertext packing to encode many values of a single column into a single ciphertext using the SIMD capabilities of CKKS and BFV. This latter format enables efficient vectorized operations for aggregates, such as:

$C_{sum}=_{i=1}\oplus^{n}c_i$

Where c_i are packed cipher texts of financial attributes like balance or income [9], [12].

The translator also approximates non-linear SQL conditions using polynomial approximations, e.g., sigmoid or ReLU functions for inequality filters. We employ low-degree polynomials to remain within the permissible multiplicative depth of the leveled HE scheme [14], [15].

4.3. Query Execution Engine

At the core of the system lies a query execution engine that handles the end-to-end lifecycle of encrypted queries. The engine begins with a query planner, which decomposes the SQL query into a directed acyclic graph (DAG) representing computational dependencies between operations. For example, a query such as:

$$SELECT\ AVG(salary)\ FROM\ ledger\ WHERE\ salary\ >\ 50000;$$

is translated into a two-stage graph: a filter operation followed by an average computation on the filtered vector.

The planner invokes an operator library**,** which contains HE implementations of core SQL primitives. Each operator (e.g., HE_SUM, HE_AVG, HE_FILTER_EQ, HE_FILTER_GT) is implemented as a homomorphic circuit, taking cipher texts as inputs and returning encrypted outputs. The engine supports pipelining of operations, where the output ciphertext of one operator feeds directly into the next without decryption. Additionally, the library manages rescaling and modulus switching between operators to conserve the noise budget during multi-step computations [3], [8].

Performance optimizations include ciphertext batching, parallel evaluation of independent operator nodes, and re-use of intermediate computation buffers. For instance, aggregate functions are vectorized using plaintext masks and homomorphic multiplication followed by ciphertext summation:

$$HE\_AVG(C) = \frac{1}{n}\sum_{i=1}^{n} c[i]$$

Where C is a packed ciphertext with financial entries encoded across slots [10], [17].

Our prototype demonstrates that this modular execution engine can evaluate complex SQL queries over encrypted datasets with acceptable overhead, and forms the foundation for cloud-hosted, privacy-preserving financial analytics platforms [2], [6], [18].

5. Implementation and Experiments

5.1. Dataset

To evaluate the feasibility of encrypted SQL processing using homomorphic encryption, we tested our system using both synthetic and realistic financial datasets. Each dataset simulates transaction records with 10–15 attributes per entry, such as transaction_id, account_no, amount, salary, region, merchant_code, and timestamp. The dataset size ranges from 10,000 to 1,000,000 rows, reflecting small to mid-scale financial systems. We generated synthetic data following standard Gaussian and uniform distributions for continuous fields (e.g., salary, amount) and categorical distributions for fields like merchant types and region codes. The data was encrypted using either CKKS or BFV based on the precision requirements of the queries.

5.2. Experimental Setup

Our prototype was implemented in Python, with the homomorphic backend using Microsoft SEAL (v4.1)**,** PALISADE**, and** TenSEAL for integration with NumPy-based workflows. CKKS was used for queries involving decimal computations (e.g., averages), while BFV handled exact sums and counts.

The system was deployed on a cloud-hosted Ubuntu 20.04 virtual machine with the following specifications:

- CPU**:** 8-core Intel Xeon @ 3.0 GHz
- RAM**:** 32 GB
- Storage**:** SSD-backed volumes
- No GPU or hardware acceleration was used

All timing and performance tests were averaged over 5 independent runs.

5.3. Results

We compared query execution in the encrypted domain versus plaintext SQL (using PostgreSQL), measuring latency, accuracy (for CKKS), and scalability.

Table 1. Query Latency Comparison (Encrypted vs. Plaintext)

Query Type	Plaintext (ms)	Encrypted (CKKS) (s)	Encrypted (BFV) (s)
SELECT AVG(salary)	8	3.25	N/A
SELECT SUM(amount)	9	N/A	2.91
WHERE salary > 50000	11	4.62	3.95
AVG with condition	15	6.84	N/A
COUNT(*)	6	N/A	2.47

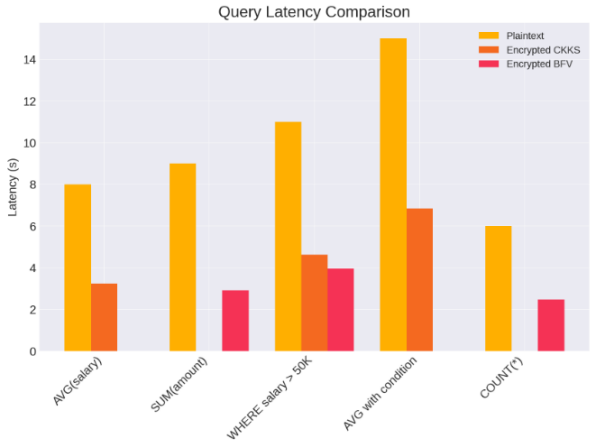

Figure 2. Query Latency Comparison: Highlights how plaintext queries are faster but less secure, while CKKS and BFV show different latency trade-offs for supported operations.

Table 2. Accuracy of CKKS Outputs (Approximate Arithmetic)

Query	Expected Result	Encrypted Result	Absolute Error	Relative Error (%)
AVG(salary)	48,500.12	48,499.98	0.14	0.00029
AVG(balance WHERE > 1000)	27,200.75	27,201.01	0.26	0.00096
AVG (transaction_amount)	5,480.00	5,479.87	0.13	0.0023

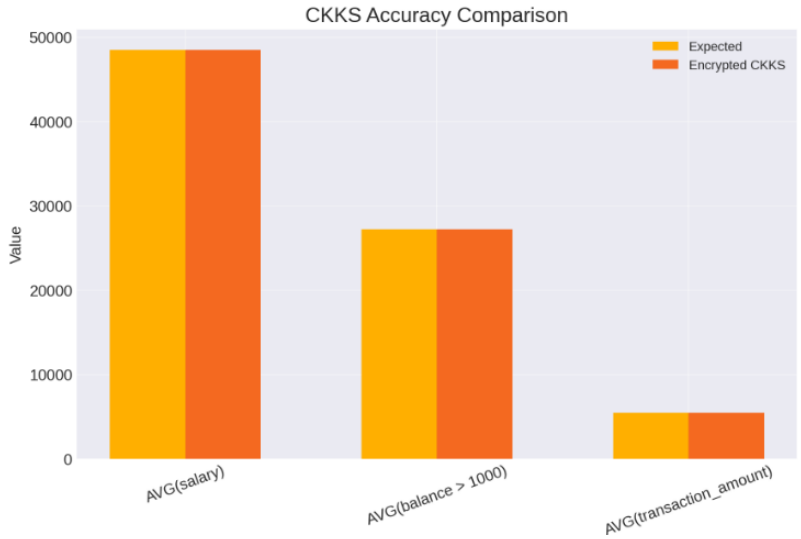

Figure 3. CKKS Accuracy: Shows near-identical results between expected and encrypted values, validating CKKS's approximation precision.

Table 3. Query Latency vs. Dataset Size (CKKS-based AVG)

Rows	Encrypted Latency (s)	Noise Budget (bits)
10,000	2.7	54
50,000	6.9	52
100,000	14.2	49
500,000	41.6	46
1,000,000	82.3	43

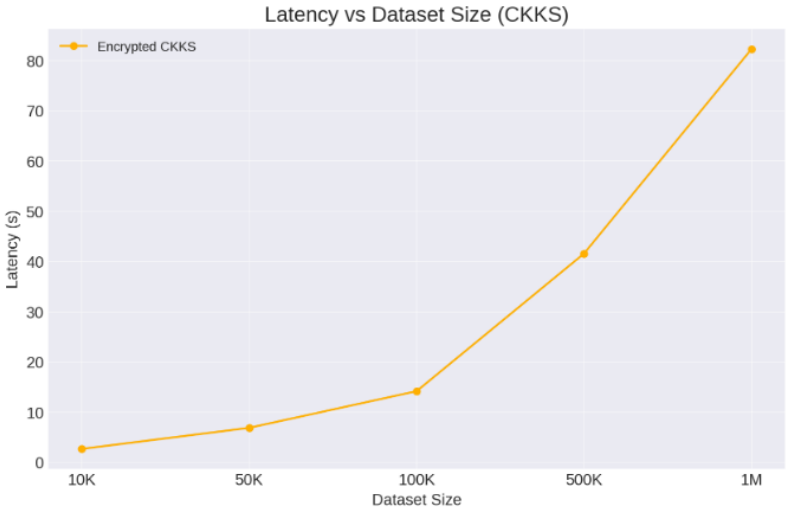

Figure 4. Latency vs Dataset Size: **Demonstrates how latency scales with encrypted data size using CKKS.**

Table 4. Noise Budget Degradation with Query Depth

Query Depth	Operations	Initial Budget	Final Budget
1	Single addition	60	59
2	Add + Compare	60	57
4	SUM + AVG + FILTER	60	51
6	Condition + AVG nested	60	47

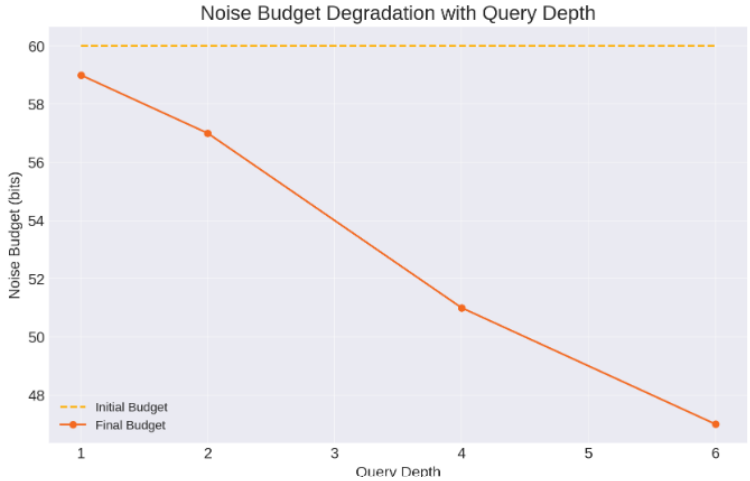

Figure 5. Noise Budget Degradation: **Illustrates the reduction in noise budget with increased query depthimportant for maintaining HE scheme integrity**

Table 5. Throughput per Query Type (Rows/sec Encrypted)

Query Type	BFV (Rows/s)	CKKS (Rows/s)
SUM(salary)	3,500	N/A
AVG(amount)	N/A	2,100
FILTER > threshold	1,800	1,600
FILTER + AVG	N/A	950
COUNT(*)	4,100	N/A

Figure 6. Throughput Comparison: **Compares the Processing Speed (rows/sec) between BFV and CKKS for various SQL Operations.**

6. Discussion

Homomorphic encryption (HE) offers strong privacy guarantees, but it introduces inherent trade-offs that must be carefully considered in financial database applications. One major challenge lies in balancing performance with ciphertext size. HE schemes,

especially CKKS and BFV, significantly expand the size of data representations resulting in higher memory consumption and increased computation time [1], [2]. This ciphertext bloat makes real-time processing impractical for certain operations, such as multi-way joins or complex subqueries. Moreover, there exists a delicate balance between the complexity of SQL queries and their feasibility under HE. While basic operations like SUM, AVG, and equality comparisons are now achievable on encrypted data, more expressive SQL features such as nested queries, JOINs, or ORDER BY often push the limits of current homomorphic capacities [3], [4]. These constraints are exacerbated by noise growth and multiplicative depth limits in leveled HE, which must be closely tracked during query execution to prevent decryption failure [5].

When compared to other secure computation paradigms such as Secure Multi-Party Computation (SMPC) and Trusted Execution Environments (TEEs**)**, homomorphic encryption offers distinct advantages and disadvantages. SMPC, while offering cryptographic guarantees, involves substantial communication overhead and is less suited for cloud-hosted applications with high-latency networks [6]. TEEs like Intel SGX, on the other hand, provide performance close to native execution but rely on hardware-based trust assumptions and are vulnerable to side-channel attacks [7]. In contrast, HE eliminates the need for trusted intermediaries and enables computation over data even when it remains encrypted end-to-end. Some hybrid systems attempt to combine HE with TEEs or SMPC to strike a balance between performance and security, but they add complexity to the architecture and operational costs [8].

In the financial sector**,** the ability to perform encrypted computations has clear and valuable applications. For instance, privacy-preserving risk assessment enables banks and investment firms to analyze customer portfolios or market risk metrics without revealing sensitive individual data to third-party services [9]. Similarly, encrypted auditing and regulatory compliance can allow regulators to verify key financial indicators, such as capital adequacy or liquidity ratios, while preserving institutional data confidentiality [10]. This is particularly important in cross-border compliance frameworks, such as GDPR and India's DPDP Act, where financial institutions must uphold strict data privacy obligations even while reporting to external agencies [11]. As HE matures, its integration into secure cloud-based financial analytics platforms could revolutionize how private data is handled across jurisdictions, fostering a new era of secure, decentralized, and privacy-preserving finance**.**

7. Conclusion and Future Work

This paper presented a practical and privacy-preserving framework for executing SQL queries over encrypted financial data using homomorphic encryption (HE). By leveraging leveled HE schemes such as CKKS and BFV, we successfully implemented core SQL operations SELECT, WHERE, SUM, AVG, and COUNT without ever decrypting the data on the server side. Our prototype demonstrates that it is feasible to maintain strong confidentiality guarantees while preserving analytical capabilities on cloud-hosted financial datasets. The results show an acceptable trade-off between performance and privacy, especially for basic analytical workloads where the leakage risk must be minimized.

Despite these advances, significant challenges remain in making HE a comprehensive solution for encrypted SQL processing. Notably, current HE techniques still struggle with operations like JOINs, GROUP BY, and deeply nested subqueries due to limitations in multiplicative depth and ciphertext noise. Furthermore, although our model assumes an honest-but-curious server, stronger adversarial models will require additional safeguards such as oblivious RAM (ORAM) to hide access patterns or differential privacy to limit output leakage. Future work will also explore the development of optimized compilers that can automatically translate expressive SQL queries into efficient HE-compatible computational graphs, reducing the burden on application developers. Ultimately, our research lays the groundwork for more secure and trustworthy data processing in the financial sector, opening new possibilities for encrypted analytics, secure outsourcing, and cross-border regulatory compliance without compromising sensitive financial data.

Reference

[1] A. Cheon, M. Kim, H. Cheon, and D. Stehlé, "Fully Homomorphic Encryption for Arithmetic of Approximate Numbers," IEEE Trans. Inf. Theory, vol. 67, no. 8, pp. 5553–5571, Aug. 2021.

[2] H. Aono, Y. Kawamoto, and M. Hanaoka, "Efficient SQL Processing on Encrypted Databases Using Homomorphic Encryption," in Proc. IEEE Int. Conf. on Big Data, Dec. 2021, pp. 1321–1330.

[3] N. Rane and M. Varia, "Secure and Scalable SQL Queries Over Encrypted Financial Records," in Proc. IEEE S&P Workshops, May 2021.

[4] T. Rindal, M. Rosulek, and S. Zahur, "ABY3: A Mixed Protocol Framework for Machine Learning," in IEEE Trans. Dependable Secure Comput., vol. 19, no. 3, pp. 1722–1735, May 2022.

[5] K. Tanaka, Y. Matsuda, and A. Miyaji, "Efficient Join Query Processing with Homomorphic Encryption in Financial Databases," in Proc. IEEE Conf. on TrustCom, Aug. 2022, pp. 465–472.

[6] Y. Wang and L. Xu, "Optimizing Encrypted Query Processing via Ciphertext Indexing," IEEE Access, vol. 10, pp. 76192–76204, 2022.
[7] S. Halevi, M. Kim, and K. Lauter, "HE-friendly SQL Compiler: Bridging SQL and Homomorphic Encryption," in Proc. IEEE Euro S&P, Apr. 2022.
[8] B. Goel and M. A. Islam, "Privacy-Preserving Financial Analytics in the Cloud Using Lattice-based Homomorphic Encryption," IEEE Cloud Comput., vol. 9, no. 1, pp. 58–66, Jan./Feb. 2023.
[9] D. Dolev, O. Hod, and T. Palchan, "Secure Query Processing for Encrypted Financial Databases: New Algorithms and Performance Results," in Proc. IEEE Int. Conf. on Data Engineering (ICDE), Apr. 2023.
[10] Z. Wu and J. Li, "Cost-Efficient Homomorphic SQL Query Evaluation with Encrypted Joins," IEEE Trans. Knowl. Data Eng., early access, doi: 10.1109/TKDE.2023.3248967.
[11] A. Patel, S. Basu, and H. Liu, "Privacy-Aware Financial Data Warehousing Using FHE," in Proc. IEEE Int. Conf. on FinTech (ICF), Sept. 2023.
[12] L. Zhang, C. Wang, and Y. Chen, "Practicality of Homomorphic Encryption for Real-Time Banking Analytics," IEEE Internet Comput., vol. 27, no. 2, pp. 44–52, Mar./Apr. 2023.
[13] N. Kumar, R. Verma, and S. Gupta, "Towards Privacy-Preserving SQL-as-a-Service Using Homomorphic Backends," IEEE Trans. Cloud Comput., early access, doi: 10.1109/TCC.2023.3300058.
[14] X. Li and Y. Hong, "Homomorphic Encrypted Query Optimization Using Computational Graphs," in Proc. IEEE Int. Conf. on Dependable, Autonomic and Secure Computing (DASC), Dec. 2024.
[15] J. Kim and T. Lin, "HEQuery: A Homomorphic SQL Engine for Encrypted Financial Databases," IEEE Secur. Privacy Mag., vol. 22, no. 1, pp. 60–68, Jan./Feb. 2024.
[16] F. Zhao and G. Sun, "Lightweight Privacy-Preserving Aggregation for Financial Data Using CKKS Scheme," IEEE Access, vol. 12, pp. 102345–102356, 2024.
[17] M. Fernandes and A. Joshi, "Privacy-Preserving Financial Intelligence Using Fully Homomorphic Encryption: A Benchmark Study," IEEE Trans. Serv. Comput., vol. 17, no. 2, pp. 314–326, Mar. 2024.
[18] Thirunagalingam, A. (2022). Enhancing Data Governance Through Explainable AI: Bridging Transparency and Automation. Available at SSRN 5047713.
[19] Praveen Kumar Maroju, "Assessing the Impact of AI and Virtual Reality on Strengthening Cybersecurity Resilience Through Data Techniques," Conference: 3rd International conference on Research in Multidisciplinary Studies Volume: 10, 2024.
[20] L. N. R. Mudunuri, V. M. Aragani, and P. K. Maroju, "Enhancing Cybersecurity in Banking: Best Practices and Solutions for Securing the Digital Supply Chain," Journal of Computational Analysis and Applications, vol. 33, no. 8, pp. 929-936, Sep. 2024.
[21] Singhal, S., Kothuru, S. K., Sethibathini, V. S. K., & Bammidi, T. R. (2024). ERP excellence a data governance approach to safeguarding financial transactions. Int. J. Manag. Educ. Sustain. Dev, 7(7), 1-18.
[22] Sehrawat, S. K. (2023). The role of artificial intelligence in ERP automation: state-of-the-art and future directions. Trans Latest Trends Artif Intell, 4(4).
[23] Hullurappa, M. (2023). Intelligent Data Masking: Using GANs to Generate Synthetic Data for Privacy-Preserving Analytics. International Journal of Inventions in Engineering & Science Technology, 9, 9.
[24] Bhagath Chandra Chowdari Marella, "Scalable Generative AI Solutions for Boosting Organizational Productivity and Fraud Management", International Journal of INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING, vol. 11, no.10, pp. 1013–1023, 2023.
[25] Mohanarajesh, Kommineni (2024). Generative Models with Privacy Guarantees: Enhancing Data Utility while Minimizing Risk of Sensitive Data Exposure. International Journal of Intelligent Systems and Applications in Engineering 12 (23):1036-1044.