Security-Performance Optimization in Cloud-Based Bank Data Processing Using HE-ZKP-ORAM

Tuan Nguyen Kim
Phenikaa School of Computing, Phenikaa University, Duong Noi, Hanoi, Vietnam
tuan.nguyenkim@phenikaa-uni.edu.vn (corresponding author)

Nguyen Minh Nhut Pham
Vietnam-Korea University of Information Technology, Da Nang University, Vietnam
pnmnhut@vku.udn.vn

Received: 2 December 2025 | Revised: 30 December 2025, 20 January 2026, and 10 February 2026 | Accepted: 13 February 2026

 DOI: https://doi.org/10.48084/etasr.16654

ABSTRACT

In the context of banking systems increasingly relying on cloud computing platforms, protecting sensitive data while maintaining processing performance is a major challenge. This paper presents and evaluates a cloud banking data processing model that integrates Homomorphic Encryption (HE), Zero-Knowledge Proof (ZKP), and the ORAM protocol to achieve a balance between security and performance. Experiments were conducted on a real Bank Marketing (UCI) dataset with 5000 records, using DSL query operations to calculate the average balance, count high-balance customers, total call duration, and savings deposit acceptance rate. The results show that the combination of HE, ZKP, and ORAM significantly improves security but increases computational cost; however, a suitable configuration can significantly reduce latency while still meeting security requirements. A detailed analysis of the security-performance trade-off provides an important empirical basis for implementing banking data security solutions in the cloud.

Keywords-homomorphic encryption; privacy-preserving computation; secure cloud computing architecture; data security; computation on encrypted data

I. INTRODUCTION

Moving banking infrastructure and applications to the cloud brings many benefits in terms of scalability, cost, and flexibility. However, the nature of financial data, including transaction information, account balances, and customer behavior, requires a high level of security while maintaining processing performance to meet real-time query and analysis needs. Although moving banking to the cloud offers many benefits, it also creates challenges. Traditional encryption solutions, for example, protect data only during storage or transmission. Processing still requires decryption, exposing information to potential leaks. Homomorphic Encryption (HE) enables direct computation on encrypted data, eliminating this risk. However, pure HE cannot hide the entire query pattern or guarantee calculation result integrity.

To address these limitations, this study proposes a cloud banking data processing model that combines three technologies: (i) HE [1], which maintains data security throughout the entire processing lifecycle; (ii) Zero-Knowledge Proofs (ZKP) [2], which verify the correctness of the computation results without revealing sensitive data or the computation itself; and (iii) Oblivious RAM (ORAM) [3], which obscures the query patterns and data access locations to prevent inference from access behavior. Unlike previous studies, this one focuses on quantitatively analyzing the trade-off between security and performance in the context of banking data and identifying which configuration is more appropriate under different security and performance requirements. Experiments are conducted on a real Bank Marketing dataset [4] with a subset of 5000 records and DSL query operations [5] specific to banking analytics.

II. RELATED WORKS

Cloud-based data security solutions can be divided into three main groups:

1. Homomorphic Encryption (HE): Since the introduction of Fully Homomorphic Encryption (FHE) [6], many optimized variants have been proposed. Among them, CKKS [7] supports approximate arithmetic over real numbers, making it well-suited for financial statistical computations. In [8], HE was applied to protect banking data in a cloud environment, but this work was limited to encryption and basic computation without mechanisms for verifying correctness or hiding query patterns.

2. Zero-Knowledge Proof (ZKP): ZKPs, especially zk-SNARKs [9], enable proving the correctness of computation results without revealing the underlying data. Pinocchio [10] is a representative construction in this direction. In [11], ZKP was combined with HE for processing medical data, but without analyzing system performance or applying this approach to banking scenarios involving large tabular datasets.
3. Query-pattern hiding with ORAM/PIR: Oblivious RAM (ORAM) [12] and Private Information Retrieval (PIR) techniques both aim to conceal access locations and query patterns. Path ORAM [13] significantly reduced latency compared to classical ORAM constructions, and in [14], ORAM was used to secure query execution in cloud environments. However, these studies focus primarily on single-query settings and do not evaluate the integration of ORAM with HE and ZKP for batch data processing or large-scale analytical workloads.

In [15], a cloud data encryption mechanism was proposed based on CuLOA combined with SqueezeNet to generate fine-tuned keys, aiming to optimize the security-performance trade-off. Unlike this work, which focuses on processing encrypted data using HE, ZKP, and ORAM to protect data content, computational correctness, and query behavior, the approach in [15] mainly improves the key-generation process and does not address secure computation over encrypted data.

Most existing works present technical solutions or experiments on general datasets, but lack a quantitative analysis of the security-performance trade-off when combining HE, ZKPs, and ORAM in a single application scenario. In particular, no previous study has evaluated how to determine the optimal configuration of these three technologies specifically for banking data workloads. This study aims to contribute to filling that gap by deploying a combined model of HE, ZKP, and ORAM on the Bank Marketing dataset [4], performing banking-specific DSL operations, and analyzing in detail the security and performance trade-off to propose the optimal configuration.

III. PROPOSED SYSTEM

A. Objective of the Proposed System

The objective was to build a banking data processing solution on a cloud computing platform, maintaining computational performance while ensuring high information security. Specifically, the model needs to secure the data content even during processing, verify the correctness of results without revealing data or processing procedures, and hide query patterns to prevent behavioral analysis. To meet these requirements, the model simultaneously combines three technologies: HE, ZKP, and ORAM. Although the cloud server is assumed to be semi-honest, ZKPs are included to ensure result verifiability in practical deployments, where incorrect outputs may arise from implementation, configuration, or system-level faults. This verification layer improves reliability without changing the underlying threat model.

B. Overall Architecture

The system consists of four main components, each accompanied by its respective inputs and outputs:

1. Client (Bank/User): The client prepares DSL queries, encrypts data using HE, keeps the secret key, and later receives and decrypts the results. The input involves original plaintext data, DSL queries, and the HE public/secret keys. The output includes encrypted data and the encoded query, which are sent to the HE-DSL Compiler, and the final plaintext results after decryption and verification.
2. HE-DSL Compiler: This component translates the DSL query into a computation circuit compatible with HE, while optimizing packing strategies and circuit depth to reduce computational overhead. It takes as input DSL queries, HE parameters (*poly_modulus_degree*, *coeff_modulus*, *scale*), and the required security configuration, and outputs an optimized HE computation graph and data-packing information, which are forwarded to the Cloud Server.
3. Cloud Server: The server performs the HE operations on encrypted data, applies ORAM when record retrieval is needed, and generates a ZKP to prove the correctness of the computation. It takes as input encrypted data, the HE computation graph from the HE-DSL Compiler, and the ORAM/ZKP configuration, and outputs the encrypted result together with a ZKP proof, which are sent to the Verifier.
4. Verifier (Client): The verifier checks the ZKP proof, decrypts the result, and evaluates the numerical error against the expected value. It takes as input the encrypted result, ZKP proof, and the HE secret key, and outputs the plaintext result, the verification status (pass/fail), and an error report.

The system's performance benefits from the HE-DSL Compiler, which optimizes the computation circuit and takes advantage of CKKS packing to reduce the number of homomorphic operations. ORAM is applied only when needed, and ZKP is used at key steps, helping balance latency and security. From a security standpoint, the data remains encrypted throughout processing, ZKP ensures the correctness of the result without revealing information, and ORAM hides query patterns, preventing leakage through access behavior. Together, these mechanisms provide comprehensive protection for banking data in a cloud environment.

Figure 1 presents the architecture of the proposed system. Its main distinctive feature is the HE-DSL Compiler, which optimizes DSL queries into HE circuits before cloud execution. This study adopted the standard honest-but-curious (semi-honest) server model, in which the cloud follows the protocol correctly but may still attempt to infer information from encrypted data or query patterns. This assumption reflects common cloud-computing scenarios and motivates the integration of HE, ZKP, and ORAM in the proposed architecture.

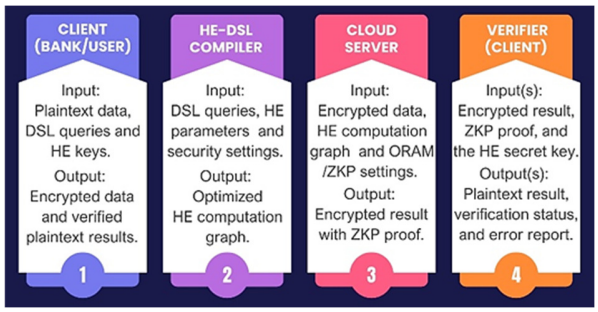

Fig. 1. Overall architecture of the proposed model.

C. Processing Flow

The proposed model handles data through four main stages, each involving a specific component and producing the output required for the next one.

- Step 1 - Query compilation: The HE-DSL Compiler (running on the client side) receives the DSL query, for example, mean(balance), and converts it into a sequence of homomorphic operations, including additions, multiplications, slot rotations, and rescaling. At the same time, the compiler determines an appropriate packing strategy and estimates the required circuit depth to improve performance. The output of this stage is an optimized HE computation circuit together with packing information, which is forwarded to Step 2.
- Step 2 - Encryption and data transmission: The client encrypts the original data using the CKKS scheme according to the parameters provided in the previous stage and attaches the description of the computation circuit. The resulting ciphertext and circuit information are then uploaded to the Cloud Server for further processing.
- Step 3 - Server-side processing: When record-level access is needed, the ORAM module protects the access pattern. The HE engine evaluates the homomorphic operations defined in the computation circuit, and the ZKP module produces a zero-knowledge proof to confirm the correctness of the computation without revealing sensitive information. The outputs are the encrypted result and the accompanying ZKP proof, both returned to the client.
- Step 4 - Verification and decryption: In the final stage, the Verifier (running on the client side) checks the ZKP proof. If verification succeeds, the client decrypts the ciphertext and obtains the validated plaintext result. This output may then be used for analysis, storage, or decision-making.

Overall, the processing flow ensures that the data remains encrypted throughout its entire lifecycle, while still providing mechanisms for correctness verification and access-pattern protection. Each stage is designed to operate independently while supplying the required output to the next stage, forming a coherent and closed processing pipeline. Figure 2 illustrates the four main stages of the encrypted data processing workflow. A key feature of this design is the HE-DSL Compiler, which optimizes queries before encryption, enabling efficient cloud-side execution using HE, ZKP, and ORAM to ensure end-to-end security.

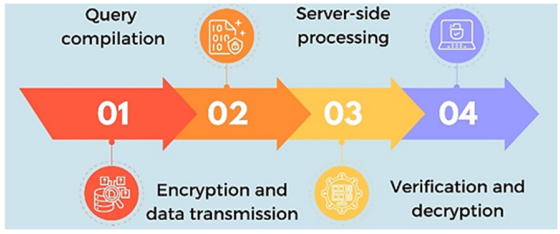

Fig. 2. Encrypted data processing workflow in the proposed system

1) Interaction Between ORAM and CKKS Packing

ORAM is applied at the data access layer to hide record access patterns, while CKKS packing is used at the computation layer for efficient homomorphic evaluation. Records are stored as individual encrypted entries within ORAM, ensuring oblivious access independent of query identifiers. After retrieval, encrypted values are packed into CKKS ciphertext slots for computation. Since ORAM is only involved in data access and not in homomorphic evaluation, its overhead, mainly due to additional data movement, remains manageable for batch-oriented analytics workloads.

D. ZKP Target and Integration with HE

In the proposed framework, ZKPs are used to verify that the encrypted result returned by the cloud corresponds to the correct evaluation of a predefined function $f(\cdot)$ over the original plaintext inputs, without revealing sensitive data. The proof is constructed over an arithmetic circuit derived from the DSL-level query, capturing plaintext-equivalent computation semantics rather than internal ciphertext operations. A standard SNARK-style representation (e.g., R1CS) is employed. As a result, the proof verifies the logical correctness of the computation while remaining independent of HE internals. Accordingly, the ZKP mechanism verifies the correctness of the intended plaintext-level computation and does not cover low-level ciphertext operations or fully malicious adversarial behaviors.

IV. EXPERIMENTAL IMPLEMENTATION

A. Data and Preprocessing

The experiments used the Bank Marketing Dataset from the UCI Machine Learning Repository [4], which contains information from telephone marketing campaigns carried out by a Portuguese bank. This dataset was selected because it has a clear tabular structure, belongs to the financial domain, where strict security requirements are common, and includes several numerical attributes that are suitable for homomorphic processing. This study used the first 5,000 records of the file bank-additional-full.csv as the sample dataset. Five key attributes were kept: *age* (customer age), *balance* (average yearly account balance), *duration* (length of the marketing call in seconds), *campaign* (number of contacts during the current campaign), and *CamR*, derived from the original *y* attribute and encoded in binary form (yes = 1, no = 0) to make it compatible with HE operations. The numerical attributes (*age*, *balance*, *duration*, *campaign*) were normalized to a common value range to reduce approximation errors when computation is performed under the CKKS scheme. After preprocessing, the data is ready for HE and for transmission to the cloud environment.

B. Experimental Scenario

The problem considered is as follows: The client (the bank) needs to perform several basic statistical operations on customer data that is stored and processed in the cloud, while ensuring that no information is revealed to the cloud server. The selected operations include Mean(balance), Sum(duration), Count(CamR = 1), which represents the number of customers who accepted the service, and Correlation(age, balance). These computations represent common statistical queries used in financial data analysis.

The queries are written in DSL and translated into optimized homomorphic computation circuits by the HE-DSL Compiler. The execution is then carried out under three different security configurations to examine the trade-off between performance and protection: Configuration A (HE only): HE is used solely to protect data content; Configuration B (HE+ZKP): ZKPs are added to ensure the correctness of the computed results; Configuration C (HE+ZKP+ORAM): All three mechanisms, HE, ZKP, and ORAM, are combined to protect data content, verify correctness, and hide access patterns, providing the highest level of security.

C. Deployment Environment

The experiments were implemented in Python using the following main libraries and tools: Microsoft SEAL for homomorphic encryption with the CKKS scheme [7], libsnark for constructing and verifying ZKPs [16], and a Python-based implementation of Path ORAM to hide access patterns during record retrieval [13]. The system was deployed on a PC equipped with an Intel Core i7 CPU, 16 GB of RAM, and running Ubuntu 22.04 LTS. The key CKKS parameters used consistently across all experimental configurations were as follows: *poly_modulus_degree* = 8192; *coeff_modulus* = [60, 40, 40, 60] bits; *scale* = 240. These environment settings ensure consistent evaluation across all experimental configurations, allowing the performance differences observed to reflect the impact of the security mechanisms rather than variations in hardware or parameter settings.

D. Experimental Procedure

The experimental workflow was carried out through the following steps:

- Step 1 - Data preparation: The client preprocesses the dataset, normalizing numerical attributes and encoding the CamR label into binary form.
- Step 2 - Query compilation: The DSL queries are translated by the HE-DSL Compiler into an optimized HE computation circuit.
- Step 3 - Encryption and data transmission: The client encrypts the processed data using the CKKS scheme and uploads the ciphertext and computation circuit to the Cloud Server.
- Step 4 - Cloud-side execution: The Cloud Server performs the homomorphic computation on the ciphertext under three different security configurations: Configuration A: HE only, Configuration B: HE+ZKP, and Configuration C: HE+ZKP+ORAM;
- Step 5 - Result return and verification: The Cloud Server sends back the result ciphertext and, when applicable, the ZKP proof. The client verifies the proof, decrypts the ciphertext, and compares the output with the plaintext result to measure error and processing time.

This workflow ensures that all computations are performed directly over encrypted data while providing correctness verification and query-pattern protection, depending on the chosen configuration. The results from each configuration form the basis for evaluating the performance-security trade-offs.

E. Experimental Settings, Measurement Criteria, and Reproducibility

All experiments were conducted under controlled and repeatable settings. Each configuration was executed for 10 runs under identical parameters, and reported results correspond to average values, with runtime variation typically below 5%. The system was evaluated in single-thread mode, with warm-up runs performed to mitigate initialization and cache effects.

Runtime was measured as the end-to-end execution time, including HE, ZKP generation and verification, and ORAM access when enabled. Communication cost accounts for the total data exchanged between the client and the cloud server during query execution. Numerical errors arise from CKKS approximate arithmetic; through data normalization and appropriate parameter selection, the observed relative error remains small (below 0.1%) and stable across runs.

For statistical operations such as correlation, the computation is expressed using additions, multiplications, and multiplications by constants. Division and normalization are handled by multiplying with pre-computed approximate inverses, which is supported by the CKKS scheme. Due to approximate arithmetic, small numerical errors may occur; however, these errors are controlled through data normalization and appropriate parameter selection and remain stable across repeated runs.

V. EXPERIMENTAL EVALUATION

A. Experimental Results

All reported results are averaged over 10 independent runs under identical experimental settings. Table I reports the results obtained from running four statistical operations, mean(balance), sum(duration), count(CamR=1), and correlation(age, balance), on 5,000 records from the Bank Marketing Dataset under three different security configurations: A (HE only), B (HE+ZKP), and C (HE+ZKP+ORAM). This table summarizes the average processing time, the amount of transmitted data, the size of the ZKP proof (when applicable), and the error compared with plaintext computation.

The results indicate a clear trend. Configuration A achieves the lowest processing time and communication overhead because it relies solely on HE. Configuration B introduces an additional cost due to the construction and transmission of ZKP proofs. Configuration C incurs the highest overhead but also provides the most comprehensive security guarantees. This

pattern is consistent with the theoretical analysis discussed earlier. The increased runtime and communication costs are mainly due to the additional security mechanisms enabled in each configuration. In particular, homomorphic computation dominates the execution time, while ZKP and ORAM introduce supplementary overhead consistent with their respective roles in correctness verification and access-pattern protection.

Figure 3 illustrates the trade-off between processing time and security level: Configuration A is at the performance extreme, configuration C is at the security extreme, and configuration B is balanced between them. From a system perspective, the dominant performance cost comes from homomorphic computation, since encrypted arithmetic operations largely determine execution time. ZKP generation and verification introduce additional but secondary overhead, while ORAM mainly impacts latency during data access due to oblivious retrieval. The network transmission overhead is relatively smaller.

TABLE I. EXPERIMENTAL RESULTS ON 5,000 RECORDS

Computation	Configu-ration	Time (ms)	Data Transmitted (KB)	ZKP proof (KB)	Error (%)
Mean(balance)	A	120	85	–	0.05
	B	210	95	25	0.05
	C	350	130	25	0.05
Sum(duration)	A	140	90	–	0.04
	B	230	100	27	0.04
	C	380	140	27	0.04
Count(CamR=1)	A	100	80	–	0
	B	180	90	20	0
	C	300	125	20	0
Correlation (age, balance)	A	250	110	–	0.08
	B	380	125	35	0.08
	C	520	160	35	0.08

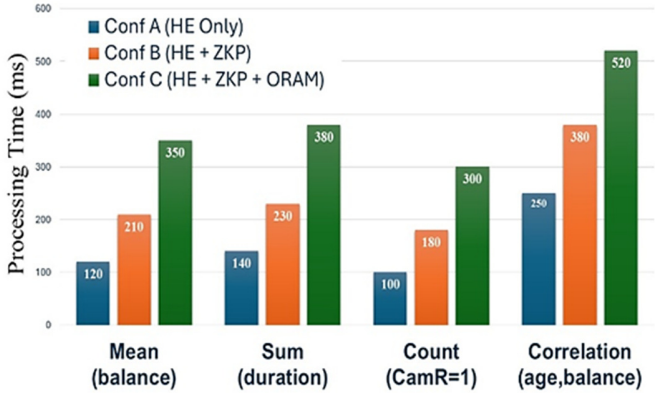

Fig. 3. Illustration of the trade-off between processing time and security level.

Overall, this analysis explains the observed performance trends and confirms that the results are consistent with the system design, where different security mechanisms contribute overhead at distinct processing stages.

B. Performance Analysis

The results in Table I highlight clear performance differences among the three experimental configurations. Configuration A (HE only) shows the lowest processing time and communication overhead because it relies solely on HE and, therefore, avoids the extra cost of generating ZKP proofs or using ORAM. However, this configuration does not include any mechanism for verifying correctness or hiding query behavior, so it is suitable only in highly trusted environments. Configuration B (HE+ZKP) introduces higher processing and communication costs compared to A, mainly due to the generation and transmission of ZKP proofs (20-35 KB depending on the query). In return, it provides stronger assurance of result correctness even when the server is not fully trusted. Configuration C (HE+ZKP+ORAM) offers the most comprehensive security, but also incurs the highest computational and communication overhead. The integration of ORAM increases latency and the amount of exchanged data due to its shuffling and access-indirection mechanisms. Its advantage is that it completely hides the query pattern.

C. Security Assessment

Configuration C (HE+ZKP+ORAM) provides the highest level of security. The use of HE protects the data content even during the computation, ZKPs ensure the correctness of the results, and ORAM completely hides the query pattern, preventing information from being inferred from the access behavior. However, this level of security comes at a significant cost in terms of processing time, data transmission volume, and implementation complexity.

In a practical context, configuration C is a suitable choice for applications that require absolute confidentiality, such as processing banking data, financial transactions, or sensitive medical information. Configuration B (HE+ZKP) can be a balance when correctness is still required, but performance is prioritized. Configuration A (HE only), although the fastest, is only suitable in trusted computing environments where the risk of attack and data leakage is low.

VI. DISCUSSION

As the size of data increases, runtime is mainly affected by homomorphic computation and ORAM access overhead. For CKKS-based encryption, circuit depth and ciphertext operations dominate runtime, while packing enables vectorized processing that mitigates growth for batch statistical queries. ORAM introduces additional overhead due to oblivious access and reshuffling, but Path ORAM's logarithmic complexity leads to a gradual increase as the dataset grows. In the proposed design, ORAM is applied selectively during data retrieval, keeping the overall overhead acceptable for moderate-scale banking analytics, where security is prioritized over real-time performance.

Adding ZKP increases both the processing time and the communication overhead due to the cost of generating and transmitting proofs. Incorporating ORAM adds further overhead, as it must shuffle data and rely on indirect access to conceal query patterns. Although Configuration C offers the strongest level of protection, not every application requires security at this level. Depending on how sensitive the data is and how the system will be operated, designers should select a configuration that gives a reasonable balance between

efficiency and security instead of defaulting to the strictest option.

The model assumes an honest-but-curious server [17], where the server follows the protocol but may try to infer information from the data or queries. Under this assumption, HE protects the data content even during computation. ZKP prevents the server from returning incorrect results, and ORAM hides access patterns, limiting behavioral analysis. However, the model does not fully address active attacks (for example, malicious code injection) or situations in which an adversary gains physical access to the client. Such cases would require additional hardware-level protections and stronger authentication mechanisms.

However, several limitations remain; the computational cost increases rapidly as the dataset grows, ORAM introduces noticeable latency because of its reshuffling mechanism, and developing DSL programs for HE still requires optimization to reduce circuit depth. Potential improvements include parallelizing HE operations to make better use of multi-core CPUs or GPUs, tuning CKKS parameters to reach a suitable balance between accuracy and performance, and applying lightweight ORAM variants to reduce query overhead. It is also important to extend the experiments to larger datasets and a broader set of query types to more accurately assess performance and security in realistic deployment scenarios.

VII. CONCLUSION

This study developed a cloud-based model for processing banking data using a combination of HE, ZKP, and ORAM, supporting statistical computation directly over encrypted data while preserving result correctness and concealing query behavior. Experiments on the Bank Marketing dataset indicate that the approach functions reliably and highlights the performance-security trade-offs observed across the three tested configurations. The main contributions of this work can be summarized as follows: (i) An integrated architecture combines HE, ZKP, and ORAM for secure financial data processing in a cloud setting, (ii) Concrete experimental scenarios using a domain-specific language to represent typical queries, (iii) Quantitative results that help identify which configuration is more suitable under different security and performance requirements. The findings suggest that Configuration C is more appropriate for environments demanding strong security guarantees, whereas Configurations B or A may be preferred when computational efficiency is the priority. Future work involves improving HE performance through parallel processing, exploring lightweight ORAM variants to reduce latency, and extending the experiments to larger datasets and more complex query patterns to better assess the applicability of the model in practical settings.

REFERENCES

[1] Y. Xie, R. Huang, and J. Qiu, "Verifiable Threshold Multi-Party Fully Homomorphic Encryption from Share Resharing," *Applied Sciences*, vol. 15, no. 9, Apr. 2025, https://doi.org/10.3390/app15094745.

[2] G. K. Mahato and S. K. Chakraborty, "A fast verifiable fully homomorphic encryption technique for secret computation on cloud data," *International Journal of Information Technology*, vol. 17, no. 4, pp. 2193–2207, May 2025, https://doi.org/10.1007/s41870-024-01994-9.

[3] D. Zhao, "Silca: Singular Caching of Homomorphic Encryption for Outsourced Databases in Cloud Computing." arXiv, June 26, 2023, https://doi.org/10.48550/arXiv.2306.14436.

[4] P. R. S. Moro, "Bank Marketing." UCI Machine Learning Repository, 2014, https://doi.org/10.24432/C5K306.

[5] M. Iezzi, "Practical Privacy-Preserving Data Science With Homomorphic Encryption: An Overview," in *2020 IEEE International Conference on Big Data (Big Data)*, Atlanta, GA, USA, Sept. 2020, pp. 3979–3988, https://doi.org/10.1109/BigData50022.2020.9377989.

[6] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, Feb. 2009, pp. 169–178, https://doi.org/10.1145/1536414.1536440.

[7] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic Encryption for Arithmetic of Approximate Numbers," in *Advances in Cryptology – ASIACRYPT 2017*, 2017, pp. 409–437, https://doi.org/10.1007/978-3-319-70694-8_15.

[8] M. A. Junior, P. Appiahene, O. Appiah, and K. Adu, "Cloud data privacy protection with homomorphic algorithm: a systematic literature review," *Journal of Cloud Computing*, vol. 14, no. 1, Nov. 2025, Art. no. 84, https://doi.org/10.1186/s13677-025-00774-5.

[9] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof-systems," in *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, Association for Computing Machinery, 2019, pp. 203–225.

[10] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza, "SNARKs for C: Verifying Program Executions Succinctly and in Zero Knowledge," in *Advances in Cryptology – CRYPTO 2013*, Santa Barbara, CA, USA, 2013, pp. 90–108, https://doi.org/10.1007/978-3-642-40084-1_6.

[11] B. Parno, J. Howell, C. Gentry, and M. Raykova, "Pinocchio: Nearly Practical Verifiable Computation," in *2013 IEEE Symposium on Security and Privacy*, May 2013, pp. 238–252, https://doi.org/10.1109/SP.2013.47.

[12] O. Goldreich and R. Ostrovsky, "Software protection and simulation on oblivious RAMs," *Journal of the ACM*, vol. 43, no. 3, pp. 431–473, Feb. 1996, https://doi.org/10.1145/233551.233553.

[13] E. Stefanov *et al.*, "Path ORAM: An Extremely Simple Oblivious RAM Protocol," *Journal of the ACM*, vol. 65, no. 4, Dec. 2018, Art. no. 18, https://doi.org/10.1145/3177872.

[14] H. Dai, Y. Ji, G. Yang, H. Huang, and X. Yi, "A Privacy-Preserving Multi-Keyword Ranked Search Over Encrypted Data in Hybrid Clouds," *IEEE Access*, vol. 8, pp. 4895–4907, 2020, https://doi.org/10.1109/ACCESS.2019.2963096.

[15] R. Patil and G. HimaBindu, "CuLOA-based Data Encryption with Tuned Key for Privacy Preservation in the Cloud," *Engineering, Technology & Applied Science Research*, vol. 15, no. 1, pp. 19546–19552, Feb. 2025, https://doi.org/10.48084/etasr.8523.

[16] B. Parno, J. Howell, C. Gentry, and M. Raykova, "Pinocchio: nearly practical verifiable computation," *Communications of the ACM*, vol. 59, no. 2, pp. 103–112, Jan. 2016, https://doi.org/10.1145/2856449.

[17] R. Buhren, S. Gueron, J. Nordholz, J. P. Seifert, and J. Vetter, "Fault Attacks on Encrypted General Purpose Compute Platforms," in *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, Nov. 2017, pp. 197–204, https://doi.org/10.1145/3029806.3029836.