

Article

Evaluation of Homomorphic Encryption Integration Strategies in Database Management Systems

Henrique Jorge¹, Cristina Wanzeller^{1,2}  and João Henriques^{1,2,*} ¹ Higher School of Technology and Management, Informatics Department, Polytechnic University of Viseu, 3504-510 Viseu, Portugal; pv21027@alunos.estgv.ipv.pt (H.J.); cwanzeller@estgv.ipv.pt (C.W.)² CISED—Research Centre in Digital Services, Polytechnic of Viseu, 3504-510 Viseu, Portugal

* Correspondence: joaohenriques@estgv.ipv.pt

Abstract

Homomorphic Encryption (HE) has emerged as a promising approach for data processing without exposing sensitive information. Despite significant advances, the practical strategies for the integration of HE into widely used database management systems (DBMSs) remain limited due to performance constraints and architectural challenges. This paper explores HE integration strategies within DBMS, focusing on SQL Server, PostgreSQL, and MariaDB. A methodology is proposed to assess the feasibility and performance of multiple HE schemes, including BFV, CKKS, BGV, TFHE, Paillier, and RSA (without padding). The evaluation considers different integration strategies, namely Python-based execution and native C++ extensions, across Windows and Debian environments. Experimental results obtained from four configurations demonstrate that the choice of HE scheme and integration strategy significantly impacts performance. Lattice-based schemes (BFV, CKKS, BGV) provide a balanced trade-off between functionality and efficiency, while TFHE incurs high computational costs due to its bit-level design. Native C++ integrations consistently outperform Python-based approaches, although the latter offer greater flexibility and ease of development. The findings highlight the feasibility of integrating HE into DBMS while emphasizing the importance of selecting appropriate schemes and integration mechanisms to meet application-specific requirements. The proposed evaluation framework provides preliminary insights into the relative behavior of different HE schemes and integration strategies under controlled experimental conditions, supporting future work on privacy-preserving DBMS design.

Keywords: homomorphic encryption; relational database management systems; database integration; healthcare data protection; cryptographic efficiency; data privacy

Academic Editors: Jun Feng and
Changqing Luo

Received: 4 May 2026

Revised: 18 June 2026

Accepted: 23 June 2026

Published: 27 June 2026

Copyright: © 2026 by the authors.
Licensee MDPI, Basel, Switzerland.
This article is an open access article
distributed under the terms and
conditions of the [Creative Commons
Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

1. Introduction

Homomorphic Encryption (HE) represents a major advancement in modern cryptography by running mathematical operations on encrypted data without requiring prior decryption [1]. This capability addresses one of the most critical challenges in contemporary data security, as it allows sensitive information to be processed while preserving confidentiality throughout its entire lifecycle [1]. By eliminating the need to expose plaintext data during computation, HE significantly reduces the risks associated with data breaches, unauthorized access, and insecure processing environments.

The concept of performing computations on encrypted data was first introduced in 1978 by Rivest, Adleman, and Dertouzos, under the notion of privacy homomorphism

[2], shortly after the development of the RSA cryptosystem [3]. Although this early work was mainly theoretical and did not provide practical constructions capable of supporting arbitrary computations, it established the foundation for later developments in homomorphic cryptography.

For several decades, the realization of Fully Homomorphic Encryption (FHE) remained an open problem, with research limited to Partially Homomorphic Encryption (PHE) schemes supporting only restricted operations, such as additions or multiplications [4]. This limitation was overcome in 2009 with the seminal work of Gentry, who demonstrated the first feasible FHE scheme capable of evaluating arbitrary circuits on encrypted data through the use of bootstrapping techniques [5]. Since then, HE has evolved significantly, leading to the development of practical schemes and increasing interest from both academia and industry, particularly in domains requiring strong data protection guarantees.

The design of HE schemes relies on advanced algebraic structures, including rings, fields, and lattices, which enable the preservation of algebraic relationships during encryption [6]. This property ensures that operations performed on ciphertexts produce, after decryption, results consistent with those obtained from plaintext computations [7]. Based on their capabilities, HE schemes are generally classified into three categories: FHE, which supports arbitrary computations; Somewhat Homomorphic Encryption (SWHE), which allows limited-depth computations; and PHE, which supports only one type of operation [8]. While FHE provides maximum flexibility, it incurs significant computational overhead, motivating the use of leveled or approximate schemes in practical applications.

The adoption of HE has grown rapidly in sectors dealing with sensitive data, such as healthcare, where confidentiality is a fundamental requirement [9]. HE enables secure processing of medical data in cloud and distributed environments while ensuring compliance with strict regulatory frameworks, such as the General Data Protection Regulation (GDPR) [10,11]. It also supports advanced applications, including statistical analysis and Machine Learning over encrypted data, facilitating collaboration between institutions without exposing sensitive information [12,13].

Recent advances have also explored the combination of HE with federated cloud architectures and advanced access control mechanisms in order to strengthen privacy and security in distributed and multi-user environments [14]. These approaches demonstrate how HE can support secure data processing while contributing to challenges such as access control enforcement and key management in complex infrastructures. Such developments further reinforce the growing relevance of HE as a complementary technology for privacy-preserving computation in modern distributed systems.

In parallel, there has been growing interest in integrating HE into Database Management Systems (DBMS), such as SQL Server, PostgreSQL, and MariaDB, which are widely used for managing large-scale structured data [15]. Such integration aims to enable query execution and data processing over encrypted datasets, extending the security guarantees of HE to database environments. However, despite promising advances, the practical adoption of HE in DBMS remains limited due to significant computational overhead, integration complexity, and performance constraints [15,16]. Accordingly, the present study focuses on comparative feasibility rather than full SQL-engine integration, query optimization, or production-scale benchmarking.

Motivated by these challenges, this paper presents a systematic evaluation of HE schemes integrated into relational DBMS, focusing on both feasibility and performance aspects. The study considers multiple schemes, integration strategies, and execution environments, providing a comprehensive comparison across SQL Server, PostgreSQL, and MariaDB platforms. The objective is not to demonstrate production readiness or large-scale scalability but rather to provide a controlled experimental evaluation of different

HE schemes and DBMS integration mechanisms, with particular emphasis on latency and operational feasibility under constrained experimental conditions.

The remainder of this paper is organized as follows. Section 2 reviews the state of the art. Section 3 describes the proposed methodology. Section 4 details the experiments. Section 5 presents the evaluation work. Section 6 discusses the results obtained. Section 7 analyzes the limitations of the study. Finally, Section 8 presents the main conclusions.

2. State of the Art

HE has emerged as a key technology for the protection of sensitive data in distributed and cloud computing environments, as it enables operations to be performed on encrypted data without the need for decryption [17]. This approach reduces the risks associated with data exposure during processing, since servers can operate without accessing plaintext data. Recent literature highlights its applicability in the healthcare sector, where the confidentiality of clinical data is critical, enabling secure and collaborative analysis without compromising patient privacy [1]. More recently, HE-based architectures have been proposed to support secure neural inference in cloud environments, particularly in intelligent healthcare applications [18].

In this domain, HE supports the secure processing of sensitive data, such as electronic health records and medical imaging, while ensuring compliance with regulatory requirements such as the GDPR [19]. It also enables collaboration between entities through protected data analysis. In this context, [11] propose an architectural model for the secure sharing of Personal Health Records (PHRs), combining FHE with distributed technologies such as Blockchain (BC) and Distributed Hash Tables (DHTs), highlighting its role in ensuring data integrity, privacy and separation of responsibilities.

Different HE schemes exhibit large fluctuations both in the range of supported operations and their computational complexity [4]. In [20], the application of FHE is analyzed to ensure the privacy of healthcare data in the context of Digital Twin systems, guaranteeing that information remains protected even during processing. Similarly, the authors of [21] provide a comprehensive review of the use of different HE types for the protection and analysis of sensitive healthcare data, discussing key techniques, applications in the medical domain, and potential vulnerabilities and mitigation mechanisms.

The integration of HE into DBMS has attracted increasing interest within the scientific community, with proposals aimed at enabling query execution over encrypted data without the need for decryption, alongside progressive improvements in performance and scalability [22]. Recent architectures, such as HE3DB, introduce efficient and elastic frameworks based on FHE infrastructures, allowing secure queries and operations to be performed over encrypted databases while mitigating some of the main challenges associated with latency and computational complexity inherent to HE [23]. This line of research is particularly relevant for overcoming performance limitations and compatibility issues with traditional DBMS, especially in cloud computing environments and distributed systems. Complementary, in [16], the application of HE in distributed databases is analyzed, highlighting the need for specific optimizations to achieve acceptable performance in query execution over encrypted data.

Although database research has made significant progress in areas such as internal optimization and the integration of artificial intelligence (AI) techniques, studies focusing on the practical application of HE in widely used relational systems, such as SQL Server, PostgreSQL, and MariaDB, remain scarce or are still at an early stage. In much of the literature, HE is primarily presented as a theoretical approach to enhancing privacy in AI workloads, often being regarded as constrained by the high computational costs involved [24].

In the context of MariaDB, there is established support for data-at-rest encryption mechanisms with minimal performance impact, highlighting its potential as a candidate for future integration with homomorphic schemes [15]. However, studies addressing such integration remain limited. Similarly, PostgreSQL and SQL Server provide encryption functionalities through extensions and complementary mechanisms, but native support for HE remains an open area, representing a relevant opportunity for future research [24].

Recent advances have explored the integration of HE with other security technologies, particularly BC, with the aim of ensuring integrity, traceability, and transparency during the processing and sharing of clinical data [11,25]. This approach strengthens security in highly regulated environments by facilitating audit mechanisms and the segregation of responsibilities, which are essential for user trust and regulatory compliance. As evidenced in [11,25], the combination of HE with BC not only protects data throughout the entire processing lifecycle but also ensures secure and verifiable auditing of performed operations, proving particularly relevant in the healthcare sector.

Performance limitations associated with HE in DBMS remain one of the main obstacles to its widespread adoption, motivating a growing body of research into optimization techniques. Among the proposed approaches, parallelization of operations, caching of intermediate results, and optimization of bootstrapping mechanisms stand out, with the aim of reducing latency and accelerating the execution of homomorphic operations [16,26]. These improvements are crucial to enable the use of HE in environments that require high availability and low response times, such as hospitals and research centers.

From a legal and ethical security perspective, the authors of [27] conducted a qualitative evaluation with experts on the use of HE and distributed technologies in the healthcare sector for the protection of clinical data, identifying relevant ethical, legal, and consent management challenges. The results indicate that, although these technologies can contribute to compliance with legal and ethical requirements and enhance patient trust, their effectiveness depends on the existence of appropriate governance models, transparency mechanisms, and robust consent management processes. In this context, HE is regarded as a promising solution to address the privacy requirements associated with clinical data, provided that it is integrated within solid regulatory and organizational frameworks capable of mitigating risks of misuse or data breaches.

The literature on the integration of HE into widely used DBMS indicates that, despite technical challenges related to performance and high computational complexity, prototypes and extensions for database engines have been developed using open-source libraries for running queries over encrypted data [22,23]. Although the work in [24] does not specifically address the integration of HE into DBMS, it provides a comprehensive framework on the evolution of database engines towards more intelligent, modular, and extensible systems. The author argues that the convergence between AI and database engines is transforming these systems into active platforms capable of incorporating new functionalities [24]. Within this evolutionary context, the recent literature identifies privacy-preserving computation techniques, such as HE, as natural candidates for future integration into DBMS, enabling secure query execution over encrypted data in sensitive domains, particularly in the healthcare sector [22].

Finally, the convergence of HE, BC, and AI opens new opportunities for the secure and efficient processing of sensitive data, as well as for the development of collaborative healthcare models that strictly preserve patient privacy [11,13]. Despite these advances, the literature dedicated to the integration of HE into traditional DBMS continues to reveal significant gaps, particularly with regard to the lack of consistent methodologies for the comparative evaluation of homomorphic schemes [28,29]. In particular, limitations persist

in terms of performance, computational efficiency, and the accuracy of encrypted operations when applied to real-world database scenarios.

3. Methodology

This section describes the proposed methodology for evaluating the integration of HE schemes in DBMS. The experimental methodology is formalized by defining the evaluation model, the processing stages, and the performance metrics for comparing HE schemes integrated into DBMS, capturing

- The multidimensional configuration space;
- The cryptographic processing pipeline;
- The latency-based performance metrics;
- The constraints influencing practical deployment.

Let

- $\mathcal{S} = \{s_1, s_2, \dots, s_m\}$ denote the set of HE schemes under evaluation;
- $\mathcal{D} = \{d_1, d_2, \dots, d_n\}$ denote the set of DBMS platforms;
- $\mathcal{I} = \{i_1, i_2, \dots, i_k\}$ denote the set of integration mechanisms;
- $\mathcal{E} = \{e_1, e_2\}$ denote the execution environments (Windows, Debian);
- $\mathcal{X} = \{x_1, x_2, \dots, x_N\}$ denote the dataset containing N records.

Each experimental configuration is defined as a tuple:

$$c = (s, d, i, e) \in \mathcal{S} \times \mathcal{D} \times \mathcal{I} \times \mathcal{E} \quad (1)$$

Let $x_j \in \mathcal{X}$ represent a plaintext value (e.g., Age). For a given scheme s , the encryption function is defined as

$$\text{Enc}_s : \mathcal{M} \rightarrow \mathcal{C}, \quad c_j = \text{Enc}_s(x_j) \quad (2)$$

where $\mathcal{M}$ and $\mathcal{C}$ denote the plaintext and ciphertext spaces, respectively.

The corresponding decryption function is

$$\text{Dec}_s : \mathcal{C} \rightarrow \mathcal{M}, \quad \hat{x}_j = \text{Dec}_s(c_j) \quad (3)$$

Correctness requires

$$\hat{x}_j = x_j \quad (\text{exact or approximate equality depending on the scheme}) \quad (4)$$

Let $\mathcal{O}_s$ denote the set of supported operations for scheme s . For an operation $f \in \mathcal{O}_s$, the homomorphic property is defined as

$$\text{Dec}_s(f(c_1, c_2, \dots)) = f(x_1, x_2, \dots) \quad (5)$$

In the experiments, operations are restricted to

$$\mathcal{O}_s \subseteq \{+, -, \times, \div\}$$

subject to the constraints defined by each scheme (cf. Table 4).

For each configuration c and each record x_j , three primary stages are considered:

- Encryption latency: $T_{\text{enc}}^{(c)}(j)$.
- Homomorphic operation latency: $T_{\text{op}}^{(c)}(j)$.
- Decryption latency: $T_{\text{dec}}^{(c)}(j)$.

The total processing latency for a record is

$$T_{\text{total}}^{(c)}(j) = T_{\text{enc}}^{(c)}(j) + T_{\text{op}}^{(c)}(j) + T_{\text{dec}}^{(c)}(j) \quad (6)$$

For a dataset subset $\mathcal{X}' \subseteq \mathcal{X}$ with $|\mathcal{X}'| = M$, the average latency for each stage is computed as

$$\bar{T}_{\text{enc}}^{(c)} = \frac{1}{M} \sum_{j=1}^M T_{\text{enc}}^{(c)}(j) \quad (7)$$

$$\bar{T}_{\text{op}}^{(c)} = \frac{1}{M} \sum_{j=1}^M T_{\text{op}}^{(c)}(j) \quad (8)$$

$$\bar{T}_{\text{dec}}^{(c)} = \frac{1}{M} \sum_{j=1}^M T_{\text{dec}}^{(c)}(j) \quad (9)$$

The overall average latency is

$$\bar{T}_{\text{total}}^{(c)} = \bar{T}_{\text{enc}}^{(c)} + \bar{T}_{\text{op}}^{(c)} + \bar{T}_{\text{dec}}^{(c)} \quad (10)$$

Each measurement is repeated $R = 5$ times, and the final reported value corresponds to

$$\mathbb{E}[\bar{T}^{(c)}] = \frac{1}{R} \sum_{r=1}^R \bar{T}_r^{(c)} \quad (11)$$

To enable comparison across configurations, a performance function is defined:

$$\Phi(c) = \bar{T}_{\text{total}}^{(c)} \quad (12)$$

Given two configurations c_1 and c_2 , c_1 is considered more efficient if

$$\Phi(c_1) < \Phi(c_2) \quad (13)$$

Each configuration c is subject to a set of constraints:

$$\mathcal{K}(c) = \{\text{scheme limitations, integration overhead, environment compatibility}\} \quad (14)$$

These constraints affect both feasibility and performance and are qualitatively analyzed alongside quantitative metrics. Memory consumption, throughput under concurrency, ciphertext expansion, and Central Processing Unit (CPU) utilization are outside the scope of the present evaluation.

The evaluation assumes an honest but curious DBMS/server model. Secret keys remain under client-side control, and the server is not assumed to access plaintext data during processing. The present study does not evaluate key rotation, multi-user policy enforcement, or advanced access-control workflows.

4. Experimental Work

This section presents the experimental setup through a set of four experiments. These experiments are designed to assess the feasibility and performance of HE integration across different DBMS, execution environments, and implementation strategies.

Each experiment corresponds to a specific configuration defined by the target DBMS, the integration mechanism, the programming environment, and the selected HE schemes and supporting cryptographic libraries. An experimental workflow is applied across all configurations, including the setup of the execution environment, the definition of

cryptographic parameters and key management procedures, and the execution of core operations such as encryption, homomorphic computation, and decryption.

4.1. OS and DBMS Setup

Next, we present the evaluated setups regarding the hardware and software, including Operating System (OS), DBMS, integration mechanisms, and cryptographic libraries. The hardware setup supported the evaluation of different DBMS, HE schemes, and integration strategies setups, as depicted in Table 1.

Table 1. Hardware and OS Setup.

OS	(1) Windows	(2) Linux Debian
Distribution	Windows 11 Pro	Debian GNU/Linux
Version	24H2	12.11
Kernel	10.0.26100.4946	6.1.0.38-amd64
Architecture	64 bits	
RAM	8 GB	
Storage	100 GB	
Processor	AMD Ryzen 7	
Cores	8	

Windows and Linux Debian OS supported the evaluation work. While the Windows OS supported Python-based evaluation integrated with SQL Server and PostgreSQL (PL/Python), Linux Debian OS supported custom C/C++ integration tools with PostgreSQL (Shared Objects) and MariaDB. The decision to select Debian Linux OS is due to its support libraries such as HELib [30,31] and TFHE [32,33]. These libraries require compatible Linux compilers and specific dependencies, namely GMP and NTL.

Running HE operations within DBMS depends on multiple factors, including the degree of integration with the database engine, the supported programming environments, and constraints imposed by the underlying OS. To systematically analyze these aspects, this work considers three distinct integration strategies: (i) execution of external code, (ii) integration through procedural languages, and (iii) native extensions to the database engine.

The DBMS, along with their corresponding integration mechanisms and categories for each experiment, are summarized in Table 2.

Table 2. DBMS and Integration Setup.

#	DBMS	Integration Mechanism	Integration Category	OS
1	SQL Server	Machine Learning Services (MLS)	Execution of external code associated with stored procedures	Windows
2	PostgreSQL	PL/Python Extension	Procedural language integrated into the database engine	Windows
3	PostgreSQL	Extensions in C/C++ (Shared Objects)	Native extension of the mechanism through dynamic libraries	Debian
4	MariaDB	User-Defined Functions (UDF)	User-defined functions invoked in SQL queries	Debian

The first experiment, the SQL Server, adopts the Machine Learning Service (MLS) mechanism [34] for running Python code within stored procedures. This approach rep-

resents an indirect integration model, where cryptographic computations are carried out outside the core database engine and invoked through SQL queries.

In the case of PostgreSQL, two experiments with different integration strategies were considered. The first experiment relies on the PL/Python extension [35], which allows Python code to be executed within the database engine. This approach provides a tighter level of integration and reduces reliance on external components, although it is subject to security and permission constraints.

The second PostgreSQL experiment employs C/C++ extensions via Shared Objects [36], allowing compiled code to run natively within the database engine. This method offers improved performance and reduced latency by avoiding interpretation overhead and external communication, at the cost of increased development complexity and configuration effort.

In the case of MariaDB, integration was achieved through User-Defined Functions (UDFs) [37] implemented in C/C++. This approach allows efficient support for cryptographic operations within SQL queries, although it provides less flexibility and requires more extensive implementation effort.

It is important to emphasize that the scope of this work was limited to the evaluation of different methods for integrating HE schemes into relational DBMS. Accordingly, the developed implementations relied exclusively on mechanisms already provided by the DBMS to extend their functionality, including the execution of external code, procedural languages, and compiled native extensions. No modifications were made to the internal architecture of the systems, the query optimizer, the transaction management mechanism, or any other core components of the database engines.

4.2. Libraries and Schemes

The following cryptographic libraries were selected for evaluation: TenSEAL [38,39], PHE [40,41], PyCryptodome [42,43], TNO [44,45], HELib, and TFHE. Table 3 depicts the libraries setups, including the versions, programming language, implemented HE schemes, and the OS setup on which they were run.

Table 3. Library Setup.

#	HE Library	Version	Language	Scheme	OS
1	TenSEAL	0.3.16	Python	BFV, CKKS	Windows
2	PHE	1.5.0	Python	Paillier	Windows
3	PyCryptodome	3.23.0	Python	RSA	Windows
4	TNO ⁽¹⁾	0.1.1	Python	BGV	Windows
5	HELlib	2.3.0	C++	BGV, CKKS	Debian
6	TFHE	1.0.1	C++	TFHE	Debian

⁽¹⁾ Tno-mpc-encryption-schemes-bgv.

Regarding the cryptographic parameters and key management, all schemes were configured to ensure a security level equivalent to 128 bits, which is widely recognized in the literature as suitable for sensitive applications with medium- to long-term protection requirements [46]. The adoption of this security level enables a practical balance between cryptographic robustness and computational cost, avoiding configurations with lower security, considered insufficient for long-term scenarios, as well as excessively high cryptographic parameters that would compromise the efficiency of encryption, homomorphic operation execution, and decryption processes.

Arithmetic operations were performed on the encrypted values, including addition, subtraction, multiplication, and division by integer or floating-point constants. The comparison was intended to analyze the behavior of the different schemes under equivalent experimental conditions and does not constitute an assessment of functional equivalence between schemes with distinct operational capabilities. It should also be noted that the developed implementation does not encompass functionalities such as joins, filtering, aggregation, indexing, range queries, updates, or workloads involving multiple columns. Table 4 summarizes the homomorphic operations support by the scheme and library.

Table 4. Library support for Homomorphic Operations and schemes.

#	Library	Scheme	Addition	Subtraction	Multiplication	Division
1	TenSEAL	BFV	Yes	Yes	Yes	No ⁽²⁾
2	TenSEAL	CKKS	Yes	Yes	Yes	No ⁽²⁾
3	PHE	Paillier	Yes	Yes ⁽¹⁾	No	No
4	PyCryptodome	RSA	No	No	Yes	No
5	TNO	BGV	Yes	Yes	Yes	No ⁽²⁾
6	HElib	BGV	Yes	Yes ⁽¹⁾	Yes	No ⁽²⁾
7	HElib	CKKS	Yes	Yes	Yes	No ⁽²⁾
8	TFHE	TFHE	Yes	Yes	Yes ⁽³⁾	No ⁽²⁾

⁽¹⁾ Subtraction is performed as addition with a negative operand (e.g., $7 - 3 = 7 + (-3)$). ⁽²⁾ Division is supported indirectly and with limitations (e.g., $8/2$ as 8×0.5). ⁽³⁾ Multiplication is achieved through repeated addition or equivalent circuit constructions.

Regarding HE schemes, different integration scenarios were evaluated. BFV, CKKS, Paillier, RSA, and BGV schemes were evaluated for PostgreSQL DMBS with PL/Python, and BFV, CKKS, Paillier, and RSA into the SQL Server. In PostgreSQL with Shared Objects and MariaDB, the BGV, CKKS, and TFHE schemes were evaluated.

For RSA, the variant without padding was adopted, as this configuration is frequently used in experimental studies investigating homomorphic properties. Although the use of RSA without padding is considered insecure in real-world communication settings [47], the introduction of padding mechanisms alters the mathematical structure of the encrypted message, transforming deterministic encryption into probabilistic encryption. As a result, the multiplicative property required to perform homomorphic operations on encrypted data is lost [48]. In this context, and solely for the purposes of experimental evaluation, RSA without padding was included as a technical reference for the execution of homomorphic operations and should not be regarded as a suitable solution for deployment in production environments.

4.3. Dataset

For the evaluation, a synthetic healthcare dataset [49] comprising 55,500 records and a diverse set of attributes representative of a hospital environment was used. Considering the computational limitations associated with the processing of HE schemes, as well as the constraints of the experimental environment, a controlled subset of 100 records was selected for testing. This sample was not intended to reproduce production workloads or assess large-scale scalability but rather to ensure consistent experimental conditions for comparative analysis. The selected subset preserves data diversity and maintains a balanced distribution of values in the Age attribute, enabling a reliable comparison across the different schemes and experimental scenarios.

Attribute selection for homomorphic operations was guided by the data types supported by each HE scheme, distinguishing between integer and decimal values. The Age attribute, which contains high distributed integer values, was chosen as it is compatible with the majority of the evaluated schemes and allows for consistent execution of arithmetic operations. Although the Room Number attribute also contains integer values, it was excluded due to its limited variability, which would reduce the relevance of the results. Similarly, the Billing Amount attribute, composed of decimal values, is natively supported only by the CKKS scheme; its use with other schemes would require data transformation and was therefore excluded from the evaluation.

4.4. Experiments

This section describes the four experimental setups. Table 5 summarizes the experimental setup to be conducted to evaluate the performance of the selected HE schemes across different DBMS platforms and integration approaches. Each experiment corresponds to a specific configuration defined by the target DBMS, execution environment, and implementation method, enabling a systematic comparison of Python-based and C++-based integrations.

The evaluated schemes and their respective libraries are described for each experimental configuration. Performance was quantified based on the mean latency obtained over five independent runs, considering the phases directly involved in cryptographic processing, namely encryption, homomorphic operation execution, and decryption. Key generation was functionally validated but excluded from the latency analysis, as the study focuses exclusively on the cryptographic phases directly associated with the operational data processing workflow.

Table 5. Experimental Setup.

#	DBMS	Integration	OS	HE Schemes	HE Library
1	SQL Server	MLS (Python)	Windows	BFV CKKS Paillier RSA	TenSEAL PHE PyCryptodome
2	PostgreSQL	PL/Python (stored functions)	Windows	BFV CKKS Paillier RSA BGV	TenSEAL PHE PyCryptodome TNO
3	PostgreSQL	Shared Objects (C/C++)	Debian	BGV CKKS TFHE	HElib TFHE
4	MariaDB	UDF (C/C++)	Debian	BGV CKKS TFHE	HElib TFHE

Regarding the SQL Server experiment (1), the required software includes the SQL Server DBMS and the Python environment used by the MLS mechanism. This environment also comprises libraries supporting data processing and manipulation, namely Revoscalepy, NumPy, Pandas, and Dill, which are used in experimental operations. The installation, administration, and management tools adopted for configuration and test execution are also included, namely Pip and SQL Server Management Studio (SSMS).

Regarding the PostgreSQL with PL/Python experiment (2), the software includes the PostgreSQL DBMS and the Python interpreter integrated into the engine through the PL/Python extension. This environment also comprises the mathematical and support libraries required for the execution of cryptographic operations, namely NumPy, Pandas, GMPy2, Mpmath, and SymPy. Additionally, the administration and development tools used in the experimental implementation are identified, namely PgAdmin, Visual Studio Code, and Pip.

In the PostgreSQL with C/C++ extensions (Shared Objects) experiment (3), the software includes the PostgreSQL DBMS and the C++ compiler used for the implementation and compilation of dynamic libraries. This environment also comprises the mathematical libraries GMP and NTL, which are essential for supporting the implemented cryptographic schemes. The development and administration tools adopted in the experimental context are also indicated, namely PgAdmin 4 and Visual Studio Code.

Finally, in the MariaDB with UDF in C/C++ experiment (4), the software includes the MariaDB DBMS and the C++ compiler used for the implementation and compilation of native UDF. This environment also comprises the mathematical libraries GMP and NTL, which support the cryptographic mechanisms underlying the developed UDF. The development and administration tools used during test execution are also indicated, namely DBeaver and Visual Studio Code.

The development software setup adopted across the four experiments is summarized in Table 6. Each setup includes the DBMS, execution environment, development libraries, and tools.

Table 6. Development Software Setup.

#	DBMS	Integration	Core Software	Libraries	Tools
1	SQL Server	MLS	SQL Server 2022 Python 3.10	Revoscalepy NumPy Pandas Dill	SSMS Pip
2	PostgreSQL	PL/Python	PostgreSQL 16.9 Python 3.11	NumPy Pandas GMPy2 Mpmath SymPy	PgAdmin 4 VS Code Pip
3	PostgreSQL	Shared Objects	PostgreSQL 16.9 C++ 12.2	GMP NTL	PgAdmin 4 VS Code
4	MariaDB	UDF	MariaDB 10.11 C++ 12.2	GMP NTL	DBeaver VS Code

5. Evaluation

This section presents the results obtained from the evaluation of the HE schemes BGV, TFHE, Paillier, RSA (without padding), BFV, and CKKS, applied to synthetic healthcare data across SQL Server (MLS), PostgreSQL (PL/Python and Shared Objects), and MariaDB (UDF) DBMS.

The average latency across the different phases of the cryptographic process is reported, namely encryption, homomorphic operations, whose specific operations per library and scheme are summarized in Table 4, and decryption, for each scheme and experimental scenario.

Key generation, encryption, and decryption stages were run at the client side in PostgreSQL with PL/Python, PostgreSQL with Shared Objects, and MariaDB, whereas in the SQL Server, these stages were executed directly within the database engine.

Table 7 presents a consolidated view of the experimental results across all evaluated configurations. The table summarizes the average latency, measured in milliseconds, for each HE scheme under different DBMS and integration strategies. Specifically, it reports the latency associated with the three primary stages of the cryptographic workflow: encryption (Enc), homomorphic operations (Ops), and decryption (Dec).

Table 7. Combined average latency (ms) across all experiments (best performance per phase in bold).

#	DBMS	Integration	Scheme	Enc	Ops	Dec
1	SQL Server	MLS	BFV	459.33	672.35	250.95
			CKKS	1172.26	1305.71	576.31
			Paillier	28,626.66	34,127.98	8549.18
			RSA	20.64	8529.31	8624.01
2	PostgreSQL	PL/Python	BFV	769.61	2431.07	1154.67
			CKKS	878.13	1784.06	1059.91
			Paillier	2658.63	24.98	2106.22
			RSA	19.59	17.99	8172.74
			BGV	18.15	89.27	151.03
3	PostgreSQL	Shared Objects	BGV	5663.16	2928.78	81,921.42
			CKKS	2578.27	67.01	14,796.83
			TFHE	646.24	5,640,478.83	6.66
4	MariaDB	UDF	BGV	1540.82	790.89	17,198.12
			CKKS	6381.49	299.98	40,331.35
			TFHE	12,248.72	6,533,857.19	3.53

The obtained latency results highlight the substantial computational overhead that still characterizes several HE schemes, particularly in Python-based integration environments and computationally intensive models such as Paillier and TFHE. In some configurations, homomorphic operations required execution times in the order of several seconds, which remains significantly above the latency expectations typically associated with conventional DBMS workloads. Consequently, the reported results should be interpreted primarily as evidence of experimental feasibility rather than practical real-time database performance.

Figure 1a illustrates the average latency of the encryption, homomorphic operation, and decryption phases for the evaluated HE schemes in the SQL Server using Python integration. The results highlight significant performance disparities, with Paillier exhibiting substantially higher latency across all phases, while RSA achieves very low encryption latency but higher costs during operations and decryption. The BFV and CKKS schemes exhibited, overall, the most balanced performance among the evaluated schemes, considering the latencies observed during the encryption, homomorphic operation execution, and decryption phases.

Figure 1b presents the average latency of the encryption, homomorphic operation, and decryption phases for the evaluated HE schemes in PostgreSQL using PL/Python integration. The results reveal distinct performance patterns across schemes. Paillier and RSA exhibit particularly low latency for homomorphic operations but incur higher costs in other phases, especially RSA during decryption. BFV and CKKS demonstrate more

balanced performance, albeit with higher operational latency compared to lightweight schemes. Notably, BGV achieves consistently low latencies across all phases, highlighting its efficiency within Python-based environments. These findings emphasize the trade-offs between computational efficiency and functional capabilities when selecting HE schemes for database integration.

Figure 1c presents the average latency of the encryption, homomorphic operations, and decryption phases for the evaluated HE schemes in PostgreSQL using integration with Shared Objects in C/C++. The results reveal extreme performance variability across schemes. TFHE exhibits exceptionally high latency for homomorphic operations, reflecting its bitwise computational model, while maintaining very low decryption costs. CKKS achieves highly efficient operation latency, significantly outperforming the other schemes in this phase. In contrast, BGV presents moderate performance during encryption and operations but incurs a substantial overhead during decryption. These findings highlight the strong dependency between execution model, implementation language, and cryptographic design, emphasizing the importance of selecting HE schemes according to the specific requirements of each processing phase.

Figure 1d illustrates the average latency of the encryption, homomorphic operation, and decryption phases for the evaluated HE schemes in MariaDB using integration with UDF in C/C++. The results highlight substantial performance disparities across schemes. TFHE exhibits extremely high latency during homomorphic operations, consistent with its bitwise computation model, while achieving very low decryption latency. CKKS demonstrates efficient performance for homomorphic operations but incurs relatively high costs in encryption and decryption phases. BGV presents a more balanced behavior, with moderate costs across all phases, although decryption remains comparatively expensive. These findings reinforce the impact of both cryptographic design and implementation strategy on overall system performance, particularly in high-performance database environments.

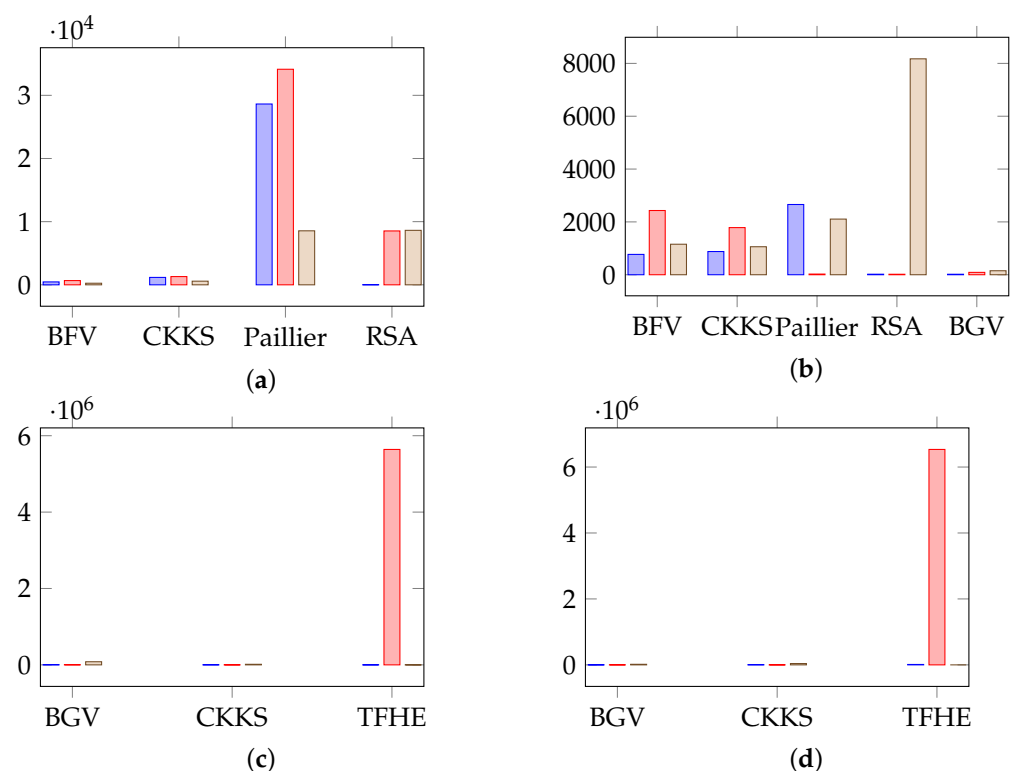


Figure 1. Comparison of encryption, homomorphic operation, and decryption latency across all DBMS and integration strategies. (a) SQL Server (MLS). (b) PostgreSQL (PL/Python). (c) PostgreSQL (C++ Shared Objects). (d) MariaDB (C++ UDF).

6. Discussion

The experimental evaluation highlights important trade-offs between HE schemes, integration strategies, and execution environments when deployed within DBMS. The results demonstrate that both the choice of cryptographic scheme and the integration approach have a significant impact on overall system performance, particularly in terms of latency across encryption, computation, and decryption phases.

A first key observation concerns the variability in performance across HE schemes. Lightweight schemes such as RSA (without padding) and Paillier exhibit very low latency for specific operations. However, these schemes show important limitations: RSA incurs high decryption costs and lacks support for general-purpose homomorphic operations, while Paillier suffers from significantly higher latency during encryption and decryption phases. These characteristics suggest limitations in scenarios that require a greater diversity of homomorphic operations in the context of the tests performed.

In contrast, the lattice-based schemes, namely BFV, CKKS, and BGV, exhibited a more balanced overall performance profile. These schemes support a broader range of homomorphic operations and recorded relatively stable latencies across the different phases evaluated. Among them, BGV produced particularly favorable results in the Python-based implementations (PL/Python), with low latencies throughout the various stages of the process. CKKS also stood out for its performance in the execution of homomorphic operations, particularly in the C/C++ implementations, where it achieved the lowest latencies in this phase among the evaluated schemes. Within the experimental scenarios considered, these findings highlight the strong overall performance of lattice-based approaches when compared with the other alternatives analyzed.

TFHE exhibited a distinct performance profile resulting from its bit-level computation model. Although it recorded low decryption times, it incurred very high computational costs during the execution of homomorphic operations, often exceeding those observed for the other schemes. This behavior is consistent with the characteristics of TFHE, which was designed for the evaluation of Boolean circuits rather than arithmetic processing. In the evaluated scenarios, these characteristics translated into substantially higher latencies during the computation phase.

A second important aspect concerns the impact of the integration strategy. The results suggest that approaches with closer integration to the database engine tend to achieve better performance. C/C++-based integrations, implemented through shared objects in PostgreSQL and UDF in MariaDB, generally recorded lower latencies than Python-based implementations during the execution of homomorphic operations, particularly in the case of CKKS. This behavior may be attributed to the elimination of interpretation overhead and the reduction of communication costs between the DBMS and external execution environments. However, these approaches required greater development and configuration effort when compared with Python-based integrations.

In contrast, Python-based integrations (SQL Server MLS and PostgreSQL PL/Python) stood out for their greater flexibility and ease of development, facilitating the implementation and evaluation of different cryptographic libraries. Although these approaches introduce additional overhead, which was particularly evident during the execution of homomorphic operations in lattice-based schemes, the results obtained show that they can deliver competitive performance in the evaluated scenarios, especially when combined with libraries such as TenSEAL and TNO.

The impact of both the DBMS and the operating system was also evident in the experimental results. PostgreSQL proved to be particularly versatile by supporting integrations through both Python and native C++ extensions, enabling direct comparison between different execution models. Debian environments, in turn, provided compatibility with

libraries such as HELib and TFHE, whereas Windows environments facilitated the integration of widely used Python libraries. The differences observed between these environments reflect not only the characteristics of the operating systems themselves but also the specific features of the libraries and integration mechanisms employed.

Across several of the evaluated scenarios, the encryption and decryption phases accounted for a substantial proportion of the total execution time, in some cases exceeding the latency associated with homomorphic operations. This behavior was particularly evident for certain schemes and experimental configurations, highlighting the importance of these phases in the overall observed performance.

Overall, the results indicate that the performance of HE schemes depends on a combination of factors, including their cryptographic characteristics, the library employed, the integration method adopted, and the execution environment. Within the scope of this study, BFV, CKKS, and BGV generally exhibited the most balanced performance among the evaluated schemes, whereas TFHE, Paillier, and RSA displayed more specialized behavior and distinct limitations in terms of either the operations supported or the observed latencies. These findings should, however, be interpreted within the specific context of the experimental scenarios considered.

The observations obtained contribute to a better understanding of the behavior of different HE schemes and integration methods in relational DBMS. It should be noted, however, that this study did not evaluate aspects such as memory consumption, throughput under concurrent workloads, integration with the query optimizer, index support, transactional behavior, or scalability to substantially larger data volumes. These aspects constitute limitations of the present work and represent important opportunities for future research.

7. Limitations and Potential Solutions

Although the results provide valuable evidence regarding the integration of HE into relational DBMS, several limitations must be considered. The experiments were conducted on a controlled subset of synthetic healthcare data comprising 100 records and focused exclusively on basic arithmetic operations over encrypted values. While this design ensured reproducibility and maintained a controlled experimental setting, it does not reflect the complexity of production-scale workloads. This limitation can be mitigated through the use of larger datasets and more representative query workloads incorporating aggregations, joins, filtering, updates, and range queries over encrypted data.

The performance evaluation was primarily based on average execution latency measured in a resource-constrained environment. Consequently, the analysis does not capture other important dimensions of system behavior, including variability, memory consumption, processor utilization, ciphertext expansion, and throughput under concurrent workloads. A more comprehensive assessment can be achieved by incorporating these complementary metrics, providing a broader view of scalability, resource efficiency, and operational stability.

A further limitation concerns the depth of integration achieved within the DBMS architecture. The evaluated approaches relied on existing integration mechanisms, including stored procedures, procedural languages, external code execution, and native extensions. Although these mechanisms enabled realistic experimentation across multiple platforms, they do not provide direct interaction with core database components such as the query optimizer, execution planner, or transaction manager. This constraint can be addressed through deeper architectural integration, including query rewriting mechanisms, optimizer-aware execution strategies, and specialized access methods designed for encrypted data processing.

The security evaluation was performed under an honest but curious threat model, assuming that cryptographic keys remain under client control. As a result, the study does not address challenges associated with multi-user environments, advanced key governance, or stronger adversarial models. More robust security guarantees can be achieved through dedicated key management services, Hardware Security Modules (HSMs), secure enclaves, and access-control architectures capable of supporting complex organizational environments.

Finally, the computational overhead inherent to current HE schemes remains a significant practical limitation. This issue was particularly evident in computationally intensive schemes such as TFHE and, to a lesser extent, in some Python-based implementations. Potential mitigation strategies include the adoption of hardware acceleration through Graphics Processing Units (GPUs) or Field-Programmable Gate Arrays (FPGAs), as well as hybrid privacy-preserving architectures combining HE with complementary techniques such as Secure Multi-Party Computation (SMPC) and Differential Privacy (DP). These approaches can reduce processing costs while preserving the security and privacy guarantees required for sensitive data management.

8. Conclusions

This study evaluated the integration of HE into relational DBMS by analyzing different cryptographic schemes and integration strategies across widely used platforms, namely SQL Server, PostgreSQL, and MariaDB. Motivated by the growing need for secure data processing in sensitive domains, this research sought to assess the practical feasibility of performing operations directly on encrypted data within database environments.

To this end, an experimental framework was developed that combined different DBMSs, integration mechanisms, and cryptographic schemes, including BFV, CKKS, BGV, TFHE, Paillier, and RSA. The evaluation focused on the complete cryptographic processing lifecycle, encompassing encryption, homomorphic computation, and decryption, with latency adopted as the primary performance metric.

The results demonstrate that the integration of HE operations into DBMS is technically feasible under the experimental conditions considered, although it involves trade-offs between performance, functionality, and implementation complexity. The Paillier and RSA schemes exhibited lower computational overhead for specific operations, whereas lattice-based schemes such as BFV, CKKS, and BGV provided a more favorable balance between functional flexibility and performance. In parallel, native C/C++-based integrations demonstrated significant efficiency advantages when compared with Python-based approaches.

Overall, the findings reinforce the potential of HE as an enabling technology for privacy-preserving data processing in database environments. Although challenges related to computational performance remain, the evidence obtained indicates that an appropriate combination of cryptographic schemes and integration mechanisms can provide a solid foundation for the development of future secure and privacy-oriented data management solutions.

Future research should focus on strengthening the practical adoption of HE within database environments through more mature integration architectures and improved query-processing capabilities. Particular attention should be given to HE-aware optimization techniques, interoperability across heterogeneous platforms, and the evaluation of emerging cryptographic schemes and libraries. In addition, the combination of HE with complementary privacy-preserving technologies and hardware acceleration mechanisms may further improve performance and expand the range of supported applications.

Author Contributions: Conceptualization, H.J. and J.H.; methodology, C.W.; software, H.J.; validation, C.W. and J.H.; formal analysis, J.H.; investigation, H.J.; resources, H.J.; data curation, H.J.; writing—original draft preparation, H.J.; writing—review and editing, C.W. and J.H.; visualization, H.J. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data sharing is not applicable.

Acknowledgments: The authors would like to thank the Research Center in Digital Services (CISeD) and the Polytechnic of Viseu for their support.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

AI	Artificial Intelligence
BC	Blockchain
BFV	Brakerski–Fan–Vercauteren
BGV	Brakerski–Gentry–Vaikuntanathan
CKKS	Cheon–Kim–Kim–Song
CPU	Central Processing Unit
DBMS	Database Management System
DHT	Distributed Hash Tables
DP	Differential Privacy
FHE	Fully Homomorphic Encryption
FPGA	Field-Programmable Gate Array
GDPR	General Data Protection Regulation
GMP	GNU Multiple Precision Arithmetic Library
GPU	Graphics Processing Unit
HE	Homomorphic Encryption
HSM	Hardware Security Module
MLS	Machine Learning Services
NTL	Number Theory Library
OS	Operating System
PHE	Partially Homomorphic Encryption
PHR	Personal Health Records
RAM	Random-Access Memory
RSA	Rivest–Shamir–Adleman
SMPC	Secure Multi-Party Computation
SQL	Structured Query Language
SSMS	SQL Server Management Studio
SWHE	Somewhat Homomorphic Encryption
TFHE	Fast Fully Homomorphic Encryption over the Torus
UDF	User-Defined Functions

References

1. Munjal, K.; Bhatia, R. A systematic review of homomorphic encryption and its contributions in healthcare industry. *Complex Intell. Syst.* **2023**, *9*, 3759–3786. <https://doi.org/10.1007/s40747-022-00756-z>.
2. Rivest, R.L.; Adleman, L.; Dertouzos, M.L. On Data Banks and Privacy Homomorphisms. *Found. Secur. Comput.* **1978**, *4*, 171–181.
3. Rivest, R.L.; Shamir, A.; Adleman, L. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM* **1978**, *21*, 120–126. <https://doi.org/10.1145/359340.359342>.
4. Acar, A.; Aksu, H.; Uluagac, A.S.; Conti, M. A Survey on Homomorphic Encryption Schemes: Theory and Implementation. *ACM Comput. Surv.* **2018**, *51*, 1–35. <https://doi.org/10.1145/3214303>.
5. Gentry, C. A fully homomorphic encryption scheme. Ph.D. Thesis, Stanford University, Stanford, CA, USA, 2009.

6. Rupnow, R.; Sassman, P. Isomorphism and Homomorphism as Types of Sameness. In *Proceedings of the 43rd Annual Meeting of PME-NA, Philadelphia, PA, USA, 14–17 October 2021*; North American Chapter of the International Group for the Psychology of Mathematics Education: Philadelphia, PA, USA, 2021.
7. Gentry, C. Homomorphic Encryption: A mathematical survey. In *International Congress of Mathematicians*; EMS Press: Berlin, Germany, 2022; Volume 2, pp. 956–1006. <https://doi.org/10.4171/ICM2022/165>.
8. Kogos, K.G.; Filippova, K.S.; Epishkina, A.V. Fully homomorphic encryption schemes: The state of the art. In *Proceedings of the 2017 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)*, St. Petersburg, Russia, 1–3 February 2017; pp. 463–466. <https://doi.org/10.1109/EIConRus.2017.7910591>.
9. Geva, R.; Gusev, A.; Polyakov, Y.; Liram, L.; Rosolio, O.; Alexandru, A.; Genise, N.; Blatt, M.; Duchin, Z.; Waissengrin, B.; et al. Collaborative privacy-preserving analysis of oncological data using multiparty homomorphic encryption. *Proc. Natl. Acad. Sci. USA* **2023**, *120*, e2304415120. <https://doi.org/10.1073/pnas.2304415120>.
10. EUR-Lex. Regulamento Geral Sobre a Proteção de Dados (RGPD). 7 January 2022. Available online: <https://eur-lex.europa.eu/PT/legal-content/summary/general-data-protection-regulation-gdpr.html> (accessed on 9 September 2025).
11. Vanin, F.N.d.S.; Policarpo, L.M.; Righi, R.d.R.; Heck, S.M.; da Silva, V.F.; Goldim, J.; da Costa, C.A. A Blockchain-Based End-to-End Data Protection Model for Personal Health Records Sharing: A Fully Homomorphic Encryption Approach. *Sensors* **2023**, *23*, 14. <https://doi.org/10.3390/s23010014>.
12. Froelicher, D.; Troncoso-Pastoriza, J.R.; Raisaro, J.L.; Cuendet, M.A.; Sousa, J.S.; Cho, H.; Berger, B.; Fellay, J.; Hubaux, J.P. Truly privacy-preserving federated analytics for precision medicine with multiparty homomorphic encryption. *Nat. Commun.* **2021**, *12*, 5910. <https://doi.org/10.1038/s41467-021-25972-y>.
13. Gandhi, B.M.; Vaghadia, S.B.; Kumhar, M.; Gupta, R.; Jadav, N.K.; Bhatia, J.; Tanwar, S.; Alabdulatif, A. Homomorphic Encryption and Collaborative Machine Learning for Secure Healthcare Analytics. *Secur. Priv.* **2024**, *8*, e460. <https://doi.org/10.1002/spy2.460>.
14. Konda, B.; Kasula, V.K.; Yenugula, M.; Yadulla, A.R.; Addula, S.R. Homomorphic Encryption and Federated Attribute-Based Multi-Factor Access Control for Secure Cloud Services in Integrated Space-Ground Information Networks. *Int. J. Commun. Inf. Technol.* **2022**, *3*, 33–40. <https://doi.org/10.33545/2707661X.2022.v3.i2a.103>.
15. Carvalho, M.; Sá, F.; Bernardino, J. Evaluation of the Impact of AES Encryption on Query Read Performance Across Oracle, MySQL, and SQL Server Databases. *Cryptography* **2025**, *9*, 77. <https://doi.org/10.3390/cryptography9040077>.
16. Singh, J. Homomorphic Encryption for Privacy-Preserving Data Processing in Distributed DBMS. *Int. J. Business Arts Sci. Study* **2025**, *6*, 13–22. <https://doi.org/10.70818/ijbass.2025.v06i02.2513>.
17. Hamza, R.; Hassan, A.; Ali, A.; Bashir, M.B.; Alqhtani, S.M.; Tawfeeg, T.M.; Yousif, A. Towards Secure Big Data Analysis via Fully Homomorphic Encryption Algorithms. *Entropy* **2022**, *24*, 519. <https://doi.org/10.3390/e24040519>.
18. Ezeogu, A.O. Homomorphic Encryption in Healthcare Analytics: Enabling Secure Cloud-Based Population Health Computations. *J. Adv. Res.* **2025**, *1*, 42–60.
19. M, A.D. Privacy Preservation using Federated Learning and Homomorphic Encryption: A Study. In *Proceedings of the 2022 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*, Falerna, Italy, 12–15 September 2022; pp. 1–8. <https://doi.org/10.1109/DASC/PiCom/CBDCom/Cy55231.2022.9927802>.
20. Ertürk, M.A. Ensuring the Privacy of Health Data in Digital Twins Using Fully Homomorphic Encryption. In *Proceedings of the 3rd International Conference on Scientific and Innovative Studies*, Konya, Turkey, 11–12 September 2024.
21. Lee, C.H.; Lim, K.H.; Eswaran, S. A Comprehensive Survey on Secure Healthcare Data Processing with Homomorphic Encryption: Attacks and Defenses. *Discov. Public Health* **2025**, *22*, 137. <https://doi.org/10.1186/s12982-025-00505-w>.
22. Zhao, D. Hermes: High-Performance Homomorphically Encrypted Vector Databases. *arXiv* **2025**, arXiv:2506.03308. <https://doi.org/10.48550/arXiv.2506.03308>.
23. Bian, S.; Zhang, Z.; Pan, H.; Mao, R.; Zhao, Z.; Jin, Y.; Guan, Z. HE3DB: An Efficient and Elastic Encrypted Database via Arithmetic-And-Logic Fully Homomorphic Encryption. In *Proceedings of the CCS '23: 2023 ACM SIGSAC Conference on Computer and Communications Security*, Copenhagen, Denmark, 26–30 November 2023; Association for Computing Machinery: New York, NY, USA, 2023; pp. 2930–2944. <https://doi.org/10.1145/3576915.3616608>.
24. Alesso, H.P. *AI Database Convergence*; AI HIVE: San Francisco, CA, USA, 2025.
25. Ali, A.; Al-rimy, B.A.S.; Alsubaei, F.S.; Almazroi, A.A.; Almazroi, A.A. HealthLock: Blockchain-Based Privacy Preservation Using Homomorphic Encryption in Internet of Things Healthcare Applications. *Sensors* **2023**, *23*, 6762. <https://doi.org/10.3390/s23156762>.
26. Shah, A.; Sivakumar, S.; N, P. Encrypted intelligence: A comparative analysis of homomorphic encryption frameworks for privacy-preserving AI. *J. Econ. Technol.* **2025**, *4*, 252–265. <https://doi.org/10.1016/j.ject.2025.08.001>.
27. Scheibner, J.; Ienca, M.; Vayena, E. Health data privacy through homomorphic encryption and distributed ledger computing: An ethical-legal qualitative expert assessment study. *BMC Med. Ethics* **2022**, *23*, 121. <https://doi.org/10.1186/s12910-022-00852-2>.

28. Zhang, Z.; Bian, S.; Zhao, Z.; Mao, R.; Zhou, H.; Hua, J.; Jin, Y.; Guan, Z. ArcEDB: An Arbitrary-Precision Encrypted Database via (Amortized) Modular Homomorphic Encryption. In *Proceedings of the CCS '24: 2024 on ACM SIGSAC Conference on Computer and Communications Security, Salt Lake City, UT, USA, 14–18 October 2024*; Association for Computing Machinery: New York, NY, USA, 2024; pp. 4613–4627. <https://doi.org/10.1145/3658644.3670384>.
29. Kupcova, E.; Pleva, M.; Khavan, V.; Drutarovsky, M. A Comparative Study of Partially, Somewhat, and Fully Homomorphic Encryption in Modern Cryptographic Libraries. *Electronics* **2025**, *14*, 4753. <https://doi.org/10.3390/electronics14234753>.
30. Halevi, S.; Shoup, V. Algorithms in HELib. In *Proceedings of the Advances in Cryptology—CRYPTO 2014*; Garay, J.A., Gennaro, R., Eds.; Springer: Berlin/Heidelberg, Germany, 2014; pp. 554–571. https://doi.org/10.1007/978-3-662-44371-2_31.
31. Halevi, S.; Shoup, V. HELib: An Implementation of Homomorphic Encryption. 16 February 2013. Available online: <https://github.com/homenc/HELlib> (accessed on 3 June 2025).
32. Chillotti, I.; Gama, N.; Georgieva, M.; Izabachène, M. Faster Fully Homomorphic Encryption: Bootstrapping in Less than 0.1 Seconds. In *Proceedings of the Advances in Cryptology—ASIACRYPT 2016*; Cheon, J.H., Takagi, T., Eds.; Springer: Berlin/Heidelberg, Germany, 2016; pp. 3–33.
33. Chillotti, I.; Gama, N.; Georgieva, M.; Izabachene, M. TFHE: Fast Fully Homomorphic Encryption Library over the Torus. 16 February 2016. Available online: <https://github.com/tfhe/tfhe> (accessed on 3 June 2025).
34. Microsoft. SQL Server Machine Learning Services. 16 December 2025. Available online: <https://learn.microsoft.com/pt-pt/sql/machine-learning/sql-server-machine-learning-services?view=sql-server-ver17> (accessed on 21 February 2026).
35. PostgreSQL Global Development Group. PL/Python—Python Procedural Language. 2026. Available online: <https://www.postgresql.org/docs/current/plpython.html> (accessed on 28 February 2026).
36. PostgreSQL Global Development Group. C-Language Functions; Available online: <https://www.postgresql.org/docs/current/xfunc-c.html> (accessed on 6 March 2026).
37. MariaDB Foundation. User-Defined Functions—MariaDB Server Documentation. 7 June 2025. Available online: <https://mariadb.com/docs/server/server-usage/user-defined-functions> (accessed on 21 February 2026).
38. OpenMined. TenSEAL: A Library for Encrypted Tensor Operations Using Homomorphic Encryption. 20 February 2025. Available online: <https://pypi.org/project/tenseal/> (accessed on 3 June 2025).
39. OpenMined. TenSEAL: A Library for Homomorphic Encryption on Tensor. 28 February 2025. Available online: <https://github.com/OpenMined/TenSEAL> (accessed on 3 June 2025).
40. CSIRO's Data61. Python Paillier: A Library for Partially Homomorphic Encryption in Python. 2 June 2016. Available online: <https://python-paillier.readthedocs.io/en/develop/> (accessed on 3 June 2025).
41. CSIRO's Data61. Python Paillier Library. 16 June 2022. Available online: <https://github.com/data61/python-paillier> (accessed on 3 June 2025).
42. Legrandin. PyCryptodome on PyPI. 17 May 2025. Available online: <https://pypi.org/project/pycryptodome/> (accessed on 3 June 2025).
43. Legrandin. PyCryptodome: A Self-Contained Python Package of Low-Level Cryptographic Primitives. 25 May 2025. Available online: <https://github.com/Legrandin/pycryptodome> (accessed on 3 June 2025).
44. TNO PET Lab. TNO PET Lab—Secure Multi-Party Computation (MPC)—Encryption Schemes—BGV. 15 July 2025. Available online: https://github.com/TNO-MPC/encryption_schemes.bgv (accessed on 20 August 2025).
45. TNO PET Lab. tno.mpc.encryption-schemes.bgv. 15 July 2025 Available online: <https://pydigger.com/pypi/tno.mpc.encryption-schemes.bgv> (accessed on 20 August 2025).
46. Xu, W.; Sun, J.; Cardell-Oliver, R.; Mian, A.; Hong, J.B. A Privacy-Preserving Framework Using Homomorphic Encryption for Smart Metering Systems. *Sensors* **2023**, *23*, 4746. <https://doi.org/10.3390/s23104746>.
47. Mu, C.; Yang, M.; Chen, Z.; Wang, B. Research on RSA Padding Identification Method in IoT Firmwares. *J. Phys. Conf. Ser.* **2020**, *1570*, 012061. <https://doi.org/10.1088/1742-6596/1570/1/012061>.
48. Gong, L.; Wang, M.; Zuo, X.; Li, S.; Wang, D. Using Transposition Padding to Get CCA2 Security from Any Deterministic Encryption Schemes. *IEEE Access* **2019**, *7*, 6765–6773. <https://doi.org/10.1109/ACCESS.2019.2891075>.
49. Patil, P. Healthcare Dataset. 8 May 2024. Available online: <https://www.kaggle.com/datasets/prasad22/healthcare-dataset> (accessed on 9 September 2025).

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.