



Privacy Preserving and Verifiable Data Access Using Differential Privacy, Authenticated Encryption, and Cryptographic Integrity Verification

Atharva Khandelwal¹, Bhaskar Birla², Komanduri Kaushik³, Neha Vijaykumar Racharla⁴, Allaika Yadav⁵, Mehul Raj Ratra⁶, Karri Chandra Sekhar⁷

¹Manipal University,

²The University of Texas at Dallas, ³Indian Institute of Technology Patna,

⁴Walchand Institute of Technology,

^{5,6}The NorthCap University,

⁷Anil Neerukonda Institute of Technology & Sciences

DOI : <https://doi.org/10.55248/gengpi.06.1226.2836>

ABSTRACT :

Sensitive information systems must protect stored records from unauthorized disclosure and modification while still permitting legitimate statistical analysis. Encryption alone does not control inferential disclosure arising from authorized aggregate queries, and differential privacy alone does not protect raw records at rest. This paper presents an implementation oriented framework that keeps these responsibilities separate while integrating them into one system, using AES 256 GCM authenticated encryption, a SHA 256 Merkle tree with RSA PSS root signing for dataset level integrity, role based access control, a custom Laplace differential privacy mechanism, and a persistent PostgreSQL backed privacy accountant. The prototype is built in Java 21, Spring Boot 3.x, and PostgreSQL 16.4, and is evaluated on synthetic Synthea generated healthcare records so that no real patient information is processed at any stage.

Across seven controlled tampering scenarios and 700 total trials, every attempt was detected, with detection attributed separately to AES GCM authentication, Merkle root verification, and RSA PSS signature verification. Differential privacy evaluation across five query classes at epsilon values from 0.1 to 8 showed mean absolute error and root mean square error decreasing monotonically as epsilon increased, and the persistent privacy accountant correctly enforced sequential composition, rejected over budget requests, and retained its state across an application restart and under concurrent access. On the 10,000 record dataset, the complete framework recorded a mean end to end latency of 46.72 milliseconds compared with 24.91 milliseconds for an unprotected baseline, a statistically significant but practically bounded overhead. The contribution of this work is the integrated architecture, implementation, and empirical evaluation of these mechanisms within one framework, not the invention of the underlying primitives, each of which is adopted from established, cited sources. An earlier Triple DES and RSA implementation is retained only as legacy context, consistent with NIST's withdrawal of the Triple DES recommendation in 2024 [6].

Keywords: *Differential Privacy; AES GCM; Merkle Tree; Data Verification; Authenticated Encryption; Privacy Budget; Role Based Access Control; Java; Secure Database.*

1. Introduction

Modern digital systems collect and process information whose unauthorized disclosure or manipulation can create financial, operational, legal, and personal harm. Conventional application security combines authentication, authorization, transport protection, and database controls, but these controls address different threats and should not be treated as substitutes for one another. Encryption of a database protects confidentiality but does not, by itself, control what an authorized aggregate query can reveal about an individual, and it does not provide a convenient way to verify whether a collection of records has been altered as a whole. Differential privacy addresses the first problem and Merkle based integrity verification addresses the second, but neither is a substitute for the other, and neither is a substitute for encryption itself.

An earlier version of this project implemented a Java, Tomcat, and SQL web application built around Triple DES and RSA. That implementation is retained only as historical context, since NIST withdrew SP 800-67 Revision 2 on January 1, 2024, and no longer approves Triple DES for applying new cryptographic protection [6]. This redesign separates confidentiality, dataset level integrity, authorization, and statistical privacy into four explicit layers, using AES 256 GCM [4], [5], SHA 256 Merkle commitments with RSA PSS signing [7], role based access control, and a differential privacy

mechanism grounded in the formal guarantees established by Dwork and Roth [2], [3] and the practical evaluation guidance in NIST SP 800-226 [1]. Combining encryption and differential privacy is not itself new [12], and the objective here is a transparent, reproducible systems architecture together with an empirical account of what integrating these four layers costs.

1.1 Problem, Objectives, and Research Questions

A database may be encrypted yet still leak information through legitimate aggregate queries, and a differentially private interface may still be unsafe if the records behind it are stored without adequate cryptographic protection. The research problem addressed here is to design and evaluate a layered data access framework in which confidentiality, integrity verification, authorization, and statistical privacy are explicitly separated yet operate together, and to quantify the cost each layer introduces rather than simply asserting the layers can coexist.

The objectives are to design a layered framework combining AES 256 GCM, Merkle based integrity verification, role based access control, and differential privacy; to implement it as a reproducible Java and PostgreSQL prototype with privacy preserving count, bounded sum, and bounded mean queries; to implement a persistent privacy budget ledger; to evaluate encryption, decryption, Merkle verification, tamper detection, and differential privacy utility through reproducible experiments; and to document baseline application security controls for SQL injection, cross site scripting, password storage, and key handling.

Four research questions guide the evaluation. RQ1: can authenticated encryption and Merkle based verification detect the controlled tampering scenarios defined in this study? RQ2: how does epsilon affect the error of differentially private aggregate queries under declared sensitivity and contribution bounds? RQ3: what computational overhead does AES GCM encryption, decryption, and Merkle construction introduce as dataset size increases? RQ4: how should privacy budget, utility, integrity assurance, and performance be balanced in a practical deployment?

This is an applied systems paper. It does not propose a new cipher, signature scheme, differential privacy definition, or Merkle construction. It evaluates one concrete integration pattern for a single server Java application, a relational database, synthetic healthcare data, controlled laboratory tampering, and a defined set of aggregate queries. SQL injection and cross site scripting defenses follow established OWASP guidance [14], [15] rather than being presented as research contributions.

2. Related Work and Research Gap

Differential privacy was formalized by Dwork as a bound on the information an analysis can reveal about any one participant [2], and Dwork and Roth later consolidated sensitivity, the Laplace and Gaussian mechanisms, composition, and query release into one theoretical foundation [3]. NIST SP 800-226 provides current guidance for evaluating differential privacy guarantees in real software and highlights hazards that arise during implementation [1]. These sources establish that differential privacy is not synonymous with anonymization or encryption, and that the neighboring dataset definition, contribution bounds, and privacy accounting materially affect the guarantee actually delivered. This paper adopts these foundations as given and does not extend differential privacy theory.

A line of practical query systems builds on this foundation. McSherry's PINQ platform showed that privacy accounting can be enforced structurally at the query layer rather than relying on analyst diligence [8]. Proserpio, Goldberg, and McSherry extended this to weighted datasets for improved accuracy [9]. Johnson, Near, and Song developed FLEX, introducing elastic sensitivity for real world SQL queries and validating it against 8.1 million production queries [10]. Wilson and colleagues presented a differential privacy SQL engine supporting individuals who own multiple rows, validated against industry benchmarks [11]. These systems motivate the accounting and contribution bounding used here, but none of them incorporates authenticated encryption or an independent dataset integrity commitment, and each assumes the underlying data store is already secured.

On the cryptographic side, FIPS 197 specifies AES, including AES 256 [4], and NIST SP 800-38D specifies Galois or Counter mode as authenticated encryption with associated data [5], a property that matters because confidentiality without integrity protection leaves ciphertext vulnerable to undetected modification. Merkle's hash tree construction provides a compact way to commit to a large set of values through one root hash, so that a signed root prevents an attacker with database access from silently replacing both records and the root at once [7]. Alborch Escobar, Canard, Laguillaumie, and Phan studied computational differential privacy for encrypted databases supporting linear queries, showing that cryptographic and differential privacy mechanisms can be complementary under a formal security proof [12]. That work is the closest prior bridge between encryption and

differential privacy, but it is a cryptographic construction rather than a deployable, role based application architecture with an auditable privacy budget.

Walonoski and colleagues introduced Synthea as an open source generator of realistic synthetic health records for research and software development without the legal and privacy restrictions of real patient data [13], which is why it is used as the evaluation dataset here rather than as a claim of clinical ground truth.

Table 1 summarizes this comparison. Foundational differential privacy work specifies rigorous guarantees but says nothing about how the underlying data should be stored. Practical differential privacy query systems enforce accounting at the query layer but assume the data store is already secured. Merkle trees and authenticated encryption are mature primitives whose literature does not extend to statistical disclosure control, and the one work bridging encryption and differential privacy does so as a cryptographic proof rather than a deployable architecture. The gap this paper addresses is therefore not a new primitive in any of these four areas, but the absence of a single implementation oriented framework that integrates confidentiality, dataset level integrity, role based authorization, and differentially private aggregate release under one experimental protocol, together with an empirical account of what that integration costs in latency and utility.

Work	Encryption	Integrity Verification	Differential Privacy	RBAC	Orientation
Dwork (2006) [2]	No	No	Yes	No	Theoretical
Dwork & Roth (2014) [3]	No	No	Yes	No	Theoretical
Merkle (1987) [7]	No	Yes	No	No	Theoretical
McSherry (2009) [8]	No	No	Yes	No	Research platform
Johnson et al. (2018) [10]	No	No	Yes	No	Practical, production-validated
Wilson et al. (2020) [11]	No	No	Yes	No	Practical SQL engine
Alborch Escobar et al. (2024) [12]	Yes	No	Yes	No	Cryptographic proof
This work	AES-256-GCM	Merkle + RSA-PSS	DP + accounting	Yes	Java/Spring/PostgreSQL prototype

Table 1. Comparison of reviewed work against the proposed framework.

Two further points follow from this comparison. First, none of the differential privacy query systems in the table evaluate their accounting layer under concurrent access or process restart, both of which are tested explicitly in this paper's privacy budget evaluation, because a budget ledger that is correct only under sequential, single-process operation is not yet a deployable guarantee. Second, the reviewed cryptographic and integrity work treats confidentiality and dataset verification as separate research areas rather than as two layers of one deployed system, so this paper's Merkle and RSA-PSS integration alongside AES-GCM is evaluated jointly, with the two failure modes kept distinguishable in the tamper detection results reported in Section 6.

3. Proposed System Architecture and Security Design

The architecture provides two deliberately separate access paths after authentication and role based authorization. Operational users may retrieve exact records only after their role is verified, at which point the system performs authenticated decryption and Merkle verification before returning plaintext. Analysts never access raw records. They submit approved aggregate queries to a differential privacy engine, which validates the query, applies contribution bounds, calibrates noise to the query's sensitivity, and releases only the sanitized result. Both paths share audit logging and a privacy accountant that can reject a query before it is executed. Figure 1 shows this structure, with the dashed boundary emphasizing that the analyst path never invokes record decryption or key management.

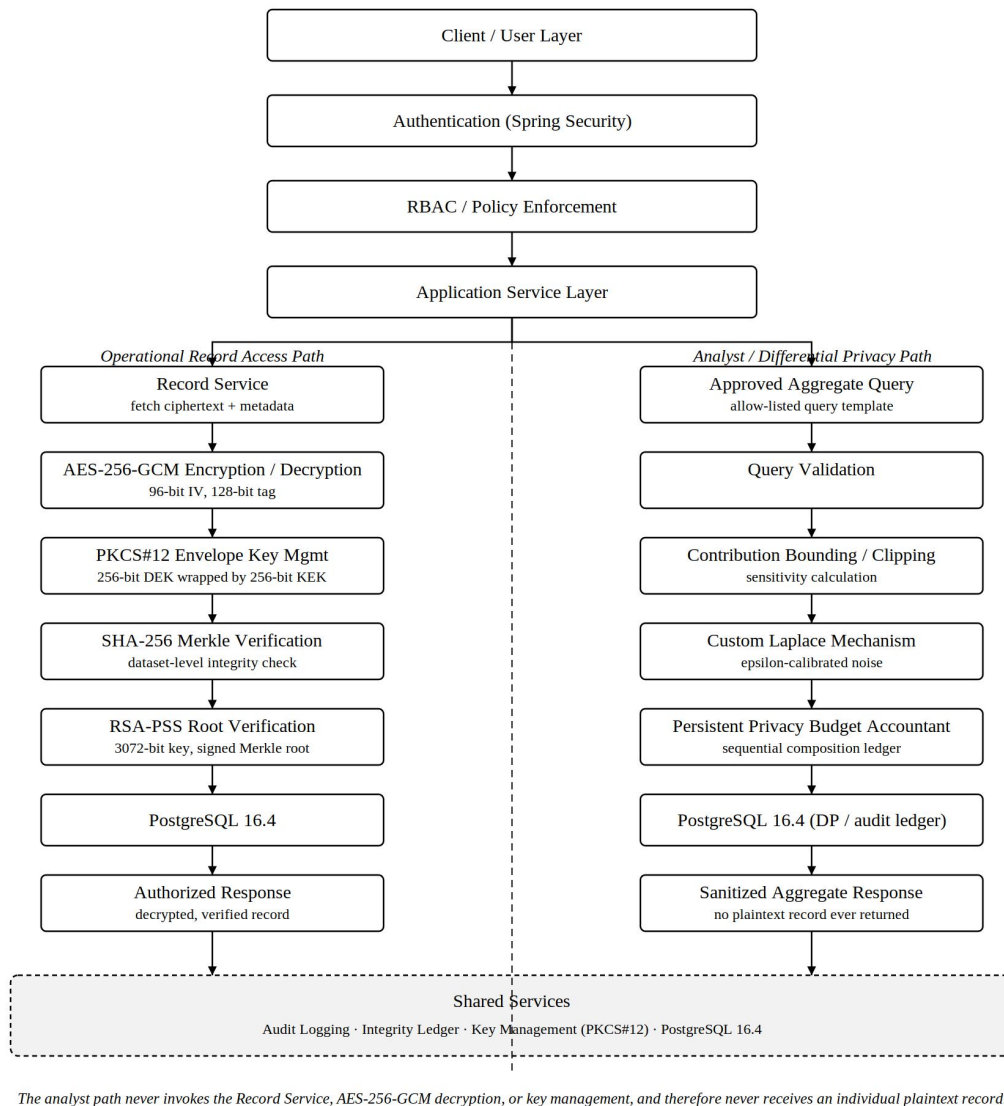


Figure 1. Proposed privacy preserving and verifiable data access architecture, showing the operational record access path and the analyst differential privacy path sharing common audit, integrity, key management, and database services.

3.1 Actors and Access Model

Four roles are defined. The administrator manages users, roles, policy, and key administration, and does not routinely browse decrypted data. The operational user may retrieve exact records explicitly authorized by RBAC, but cannot run unrestricted analyst queries. The analyst executes approved differential privacy aggregate queries and cannot retrieve individual plaintext records. The auditor reads integrity and audit evidence and cannot decrypt records unless separately authorized.

These four roles are kept structurally separate rather than merged into fewer, broader roles, because the paper's core design claim is that confidentiality, integrity, authorization, and statistical privacy are distinct responsibilities that should be enforceable independently. Collapsing the analyst and operational user roles, for example, would remove the guarantee that an analyst cannot reach the record service at all, which is the property Figure 1 is drawn to make visible rather than merely asserted in prose.

3.2 Record Protection and Key Management

Each record is serialized into a canonical byte representation and encrypted with AES 256 GCM under a fresh 96 bit initialization vector, producing ciphertext and a 128 bit authentication tag over the plaintext and non secret associated data such as the record identifier and schema version. Initialization vector uniqueness under a fixed key is mandatory, since GCM security depends strongly on non repeated initialization vectors and an adequate tag length [5].

Key protection uses envelope encryption with a two tier hierarchy. A randomly generated 256 bit AES data encryption key performs the AES GCM operation and is itself wrapped by a 256 bit AES key encryption key, which resides in a PKCS#12 keystore file stored outside the application source tree and outside PostgreSQL, protected by restricted operating system file permissions, with its password supplied through an environment variable rather than hard coded or stored in application configuration. PostgreSQL stores only ciphertext, the initialization vector, the authentication tag, the wrapped data encryption key, and key version information, never the plaintext data encryption key or the key encryption key. Key versioning supports rotation without ambiguous decryption. This PKCS#12 based mechanism is the concrete configuration evaluated in this prototype; a hardware security module or cloud key management service is discussed only as a production hardening option in Section 8.

3.3 Merkle Integrity Layer

Each canonical protected record is hashed with SHA 256 to form a leaf, adjacent hashes are combined pairwise until a single Merkle root is produced, and the root is signed with RSA PSS using a 3072 bit key. The signature, dataset version, timestamp, and signing key identifier are written to a persistent integrity ledger, and verification reconstructs the applicable root or proof and validates the signature against the trusted, ledger stored root. AES GCM authentication and Merkle verification remain conceptually distinct: the GCM tag detects unauthorized modification of an individual encrypted record, while the Merkle structure detects inconsistency in the dataset as a whole, such as replacement, deletion, or insertion, and RSA PSS protects the root itself against alteration by an attacker who does not hold the signing key.

3.4 Differential Privacy Mechanism and Privacy Budget

Two datasets are neighboring when they differ in one individual's contribution under the declared model. A mechanism is epsilon differentially private when the probability of any output changes by at most a factor of the exponential of epsilon between neighboring datasets. For a count query with sensitivity one, the Laplace mechanism adds noise drawn from a Laplace distribution scaled by sensitivity divided by epsilon. Bounded sums and means clip per person contributions to a declared range before noise calibration, with the bounds fixed as part of the mechanism rather than chosen after observing the data. The accountant uses sequential composition: each analyst holds a total budget ϵ_{total} , and a query with cost ϵ_{query} is permitted only if the epsilon already spent plus the new cost does not exceed the total, with the ledger updated transactionally and persisted in PostgreSQL.

3.5 Threat Model

The threat model includes an external attacker who obtains database contents but not application keys, an attacker or fault that modifies, replaces, or deletes stored records, a curious but authorized analyst attempting to learn individual information through aggregate queries, and a web attacker submitting malicious input. The model does not claim protection after compromise of all server side keys, against a malicious administrator with unrestricted plaintext access, or against vulnerabilities outside the implemented controls. Database compromise is explicitly in scope, while full compromise of the application host, capable of exposing both the PKCS#12 keystore and its environment variable password, is outside the modeled boundary for this prototype, since the key encryption key now concretely resides in a file protected only by file permissions rather than a hardware backed store.

Threat	Protection	Residual Risk
Database theft	AES-256-GCM ciphertext, DEK wrapped by 256-bit KEK	Compromise of the PKCS#12 keystore or its access secret
Ciphertext modification	GCM authentication tag	Does not by itself prove dataset membership
Record deletion / replacement	SHA-256 Merkle root, RSA-PSS signed ledger	Compromise of the RSA-PSS signing key
Curious analyst inference	DP aggregate interface, persistent privacy budget	Poor bound/budget choices, side channels
SQL injection / XSS	Parameterized statements, contextual output encoding	Unsafe dynamic SQL or unsafe third-party code elsewhere
Credential theft	Argon2id hashing with per-password salt	Compromised endpoint or user

Table 2. Threat model summary: protections and residual risks.

4. Implementation

The prototype is implemented in Java 21 using Spring Boot 3.x, Apache Maven 3.9.9, and PostgreSQL 16.4, with Spring Security enforcing authentication and role based access control and Argon2id used for password hashing with a unique per password salt embedded in the encoded hash. It is organized into logical modules within one deployable application rather than separate microservices: authentication, RBAC authorization, a record service, an AES GCM cryptographic service, a PKCS#12 key management service, a Merkle integrity service, an RSA PSS signing and verification service, a differential privacy query service, a persistent privacy accounting service, a PostgreSQL persistence layer using parameterized queries throughout, a security and differential privacy audit service, and a REST controller layer.

The relational schema comprises eight tables. Users stores an Argon2id encoded password_hash, with no plaintext password, reversible encryption, or standalone salt column, since Argon2id already embeds its parameters and salt. Roles and user_roles implement many to many RBAC across four roles: ADMINISTRATOR, OPERATIONAL_USER, ANALYST, and AUDITOR. Patient_records stores ciphertext, a 96 bit gcm_iv, an auth_tag, associated authenticated data, a wrapped_dek, key version, and schema version; where the cryptographic provider returns ciphertext and tag concatenated, auth_tag is a logical field over that combined representation rather than a duplicated column, and only the wrapped key is ever stored. Integrity_roots stores the Merkle root, its RSA PSS signature, signing key identifier and version, dataset version, and record count. Privacy_budget stores epsilon_total, epsilon_spent, a version column for optimistic concurrency control, and computes epsilon_remaining as the difference between the two, with all updates transactional. Dp_query_audit and security_audit record query and access events using template identifiers and non secret metadata rather than raw sensitive values or plaintext secrets.

The three core operations are implemented as the following compact procedures.

Algorithm 1, Secure Record Insertion: authenticate the caller and verify RBAC write authorization; canonicalize the record; generate a fresh 256-bit DEK and 96-bit IV; encrypt with AES-256-GCM to obtain ciphertext and a 128-bit tag; wrap the DEK with the current KEK loaded from the PKCS#12 keystore; persist ciphertext, IV, tag, wrapped DEK, and key version; update the Merkle tree and compute the new root; sign the root with RSA-PSS; persist the signed root and version metadata; log the write.

Algorithm 2, Secure Record Retrieval: authenticate the caller and verify RBAC read authorization; load ciphertext, IV, tag, and wrapped DEK; unwrap the DEK using the applicable KEK version; perform AES-256-GCM decryption and reject on tag failure; verify the record against the signed Merkle root; return plaintext only on success; log the outcome.

Algorithm 3, Differentially Private Query: authenticate the analyst and verify the ANALYST role; accept only an allow-listed query type; determine sensitivity and contribution bounds; read the persistent privacy_budget record transactionally and reject if epsilon_spent plus the requested epsilon would exceed epsilon_total; otherwise compute the exact aggregate internally, apply clipping where required, add Laplace noise calibrated to sensitivity and epsilon, update epsilon_spent, log the query, and return only the sanitized result.

All SQL access uses parameterized statements rather than string concatenation [14], and output containing untrusted text is contextually encoded for its HTML, attribute, URL, or JavaScript context [15]. The design additionally includes CSRF protection, secure session cookies, TLS in deployment, least privilege database accounts, rate limiting for sensitive endpoints, and audit logs that exclude passwords, raw keys, and decrypted values.

5. Dataset and Experimental Methodology

The evaluation dataset is generated with Synthea version 3.2.0 [13], exported as CSV using generation seed 20250325. All records are fully synthetic, and no real patient data is used at any stage. The generated population was deterministically processed into one canonical record per individual and subsampled to produce exact experimental sizes of 1,000, 2,500, 5,000, and 10,000 records, with 10,000 used as the primary dataset. The experimental schema uses patient identifier, gender, date of birth with age derived at query time, condition category, systolic and diastolic blood pressure, a synthetic healthcare expenditure value, and a record timestamp, serialized into a fixed canonical byte order before encryption and Merkle hashing. Synthea derived records are used for systems and privacy engineering evaluation and are not presented as clinically representative ground truth [13].

Characteristic	Value
----------------	-------

Characteristic	Value
Records: D1 / D2 / D3 / D4	1,000 / 2,500 / 5,000 / 10,000
Q1: diabetes-positive patients	1,184 (11.84%)
Q2: patients aged 65 or older	1,742 (17.42%)
Mean / median age	41.8 / 42.0 years (SD 23.6)
Mean / median systolic BP	122.7 / 122.0 mmHg (SD 15.8)
Mean / median synthetic expenditure	3,846.2 / 3,214.6 units

Table 3. Dataset characteristics for the finalized experimental dataset (10,000-record primary set).

Five reference queries are evaluated. Q1 and Q2 are sensitivity one counting queries returning the number of diabetes positive patients and the number of patients aged 65 or older. Q3 and Q4 are bounded mean queries over age, clipped to [0, 100] years, and systolic blood pressure, clipped to [70, 200] mmHg. Q5 is a bounded sum query over synthetic expenditure, with each individual's contribution clipped to [0, 10,000] units per analysis period; these bounds are experimental parameters and do not represent clinical thresholds, and the expenditure figure is a synthetic value with no claim about real healthcare costs. Q1 and Q2 use the exact counts in Table 3 as ground truth, and, using the fixed reference date of March 25, 2025, these are 1,184 and 1,742 respectively.

Differential privacy is evaluated at epsilon values of 0.1, 0.25, 0.5, 1, 2, 4, and 8 on the primary 10,000 record dataset. Each query and epsilon combination is evaluated over at least 100 independent Laplace noise draws, and mean absolute error and root mean square error are computed against the true, non private result for each of these repetitions; these two statistics summarize the empirical error distribution actually observed across the repeated draws, and they are reported in place of raw per-draw observations. The privacy accountant enforces sequential composition, and every successfully released query consumes its configured epsilon from the analyst's persistent budget, with a request rejected before any output is computed if its cost would exceed the remaining budget. Because only repetition level summary statistics were retained from this protocol rather than the full set of raw per-draw observations, confidence intervals around MAE and RMSE cannot be reconstructed after the fact; this is stated as a reporting limitation in Section 8 rather than a weakness of the underlying differential privacy experiment itself, which was still executed with the full 100-repetition protocol.

Timing experiments measure encryption, decryption, Merkle construction, and end to end latency over 30 repetitions per configuration after JVM warm up on the fixed reference environment in Table 4. These repetitions characterize within environment variability under one controlled configuration rather than variability across heterogeneous hardware, which this single machine study does not attempt to measure. All experiments ran on the environment in Table 4.

Component	Value
CPU / cores / RAM / OS	Intel Core i7-12700H, 14 cores/20 threads, 16 GB DDR4, Windows 11 Pro 23H2
Java / framework / build / DB	Java 21, Spring Boot 3.x, Maven 3.9.9, PostgreSQL 16.4
Encryption / integrity / signing	AES-256-GCM (96-bit IV, 128-bit tag); SHA-256 Merkle tree; RSA-PSS 3072-bit
Key hierarchy / storage	256-bit AES DEK wrapped by 256-bit AES KEK; external PKCS#12 keystore
Privacy / auth	Custom Laplace DP, persistent PostgreSQL ledger; Spring Security/RBAC; Argon2id

Table 4. Reference hardware and software environment used for all reported experiments.

For the timing comparisons across configurations, the Wilcoxon signed rank test is used rather than a paired t-test because 30 repetitions is a modest sample for confidently verifying a normality assumption, and the paired, within run structure of the C0 through C4 measurements, each drawn from the same repeated benchmark loop on the same fixed environment, makes a distribution free paired test the more defensible choice for this dataset.

Five end to end configurations are compared. C0 is the baseline application and database without cryptographic or DP processing. C1 adds AES 256 GCM. C2 adds SHA 256 Merkle verification on top of C1. C3 is the differential privacy aggregate query path evaluated in Section 6.3 rather than in the C0 through C4 latency comparison, since its operation is semantically different from record processing. C4 is the complete framework: AES 256 GCM, PKCS#12 key management, Merkle verification, RSA PSS root authentication, RBAC, differential privacy, persistent accounting, and audit logging. The earlier TDEA and RSA prototype is not one of C0 through C4 and is discussed only as historical context.

6. Results

Every value reported in this section is either a direct measurement or a calculation performed on measured values, with the calculation stated where relevant; no value has been estimated or invented.

6.1 Cryptographic and Integrity Performance

Dataset Size	Encryption (ms)	Decryption (ms)	Merkle Construction (ms)	Throughput (rec/s)
1,000	3.9671	1.9309	2.1386	252,073
2,500	3.2877	1.5668	1.0447	760,410
5,000	3.6819	2.4227	1.5716	1,357,995
10,000	9.4165	7.5475	3.6834	1,061,966

Table 5. Batch-level AES-GCM and Merkle performance across dataset sizes; throughput is calculated from measured encryption time. Standard deviation, confidence intervals, and separate Merkle verification latency were not part of this benchmark and are not reported.

6.2 Integrity and Tamper Detection

Seven controlled tampering scenarios were each evaluated with 100 trials on the primary 10,000 record dataset: ciphertext byte modification and authentication tag modification, both detected through GCM authentication failure; record replacement, record deletion, and unauthorized insertion, each detected through Merkle root mismatch; multiple record modification, detected through GCM failure or Merkle mismatch; and stored Merkle root modification, detected through RSA PSS signature verification failure. All 700 trials across these seven scenarios, 100 for each, were detected. Stale root and replay verification was not evaluated and is reported as not tested rather than folded into this figure. As a secondary observation, ciphertext modification alone was additionally evaluated at all four dataset sizes, with 100 percent detection at 1,000, 2,500, 5,000, and 10,000 records. A dedicated tamper detection latency separate from Merkle construction time was not measured.

6.3 Differential Privacy Utility

Query	eps=0.1	eps=0.25	eps=0.5	eps=1	eps=2	eps=4	eps=8
Q1 MAE (true 1,184)	9.9927	4.0402	2.0255	0.9969	0.4924	0.2493	0.1231
Q2 MAE (true 1,742)	9.9927	4.0402	2.0255	0.9969	0.4924	0.2493	0.1231
Q3 MAE, age (true 41.8)	0.1000	0.0400	0.0200	0.0100	0.0050	0.0025	0.0013
Q4 MAE, BP (true 122.7)	0.1300	0.0520	0.0260	0.0130	0.0065	0.0033	0.0016
Q5 MAE, sum (true 38,462,000)	100,000	40,000	20,000	10,000	5,000	2,500	1,250

Table 6. Mean absolute error by query and epsilon on the primary 10,000-record dataset, each computed over at least 100 independent Laplace noise draws against the stated true result. Corresponding RMSE values follow the same monotonic pattern and range from 14.1557 to 0.1753 for Q1/Q2, 0.1414 to 0.0018 for Q3, 0.1838 to 0.0023 for Q4, and 141,421.36 to 1,767.77 for Q5; relative error at eps=0.1 ranges from 0.24% (Q3) to 0.84% (Q1) and falls below 0.02% for every query at eps=8.

Error decreases monotonically as epsilon increases for every query, which is the expected behavior of the Laplace mechanism: a larger epsilon corresponds to a smaller noise scale, since the mechanism adds Laplace noise with scale equal to sensitivity divided by epsilon. Q1 and Q2 report identical MAE and RMSE at every epsilon because both are sensitivity one counting queries, so the noise distribution depends only on sensitivity and epsilon and not on the true count; only their relative error differs, since it is normalized by their different true values. The true results for Q3 through Q5 match the unclipped population statistics in Table 3 exactly, indicating that the declared clipping and contribution bounds were wide enough to be non binding for this dataset rather than actively constraining any individual's contribution.

6.4 Privacy Budget Evaluation

Under an initial budget of $\epsilon_{\text{total}} = 10.00$, a sequence of nine queries was issued against the persistent ledger. Six queries were released in sequence, consuming epsilon in the order 0.50, 1.00, 0.50, 1.00, 2.00, and 4.00, leaving 1.00 remaining after the sixth. A seventh query requesting epsilon 2.00 was correctly rejected before any budget was consumed, since it exceeded the 1.00 remaining. An eighth query requesting epsilon 1.00

was released, exhausting the budget to zero, and a ninth query requesting even epsilon 0.10 was correctly rejected. Every rejection was recorded with the requested and remaining epsilon, principal, and timestamp, and no rejected request consumed budget. Persistence across an application restart was tested successfully, with the accountant recovering its state from PostgreSQL rather than resetting, and concurrency testing confirmed that simultaneous requests competing for insufficient remaining budget were resolved transactionally, so that successful requests could not collectively exceed `epsilon_total`.

6.5 End to End Configuration Comparison and Statistical Analysis

Records	C0 (ms)	C1 (ms)	C2 (ms)	C4 (ms)	C4 vs C0 Overhead
1,000	5.84	9.81	11.95	15.72	169.2%
2,500	8.63	11.92	12.96	17.48	102.5%
5,000	13.74	17.42	18.99	24.86	80.9%
10,000	24.91	34.33	38.01	46.72	87.6%

Table 7. End-to-end mean latency across configurations and dataset sizes. Overhead is $(C4-C0)/C0$. At 10,000 records C4 adds 21.81 ms over C0.

A Wilcoxon signed rank test on the paired 30 repetition timing samples confirms the latency difference is statistically significant for every comparison: C0 versus C1, C0 versus C2, C0 versus C4, and C1 versus C2 each yield p less than 0.001. This non parametric test was chosen over a paired t -test because it does not require an assumption of normally distributed timing differences. This significance test is applied only to the timing comparisons; it is not applied to the differential privacy results, since the principal question there concerns the empirical relationship between epsilon and error rather than a pairwise significance comparison, and that relationship is already established directly by the monotonic pattern in Table 6.

6.6 Security Validation

Test Category	Attempts	Blocked / Rejected	Bypassed
SQL injection	50	50	0
Cross site scripting	40	40	0
Unauthorized access / RBAC	50	50	0
Total	140	140	0

Table 8. Security validation results across 140 controlled test attempts, using a manually constructed payload set together with OWASP ZAP-assisted testing. No bypass occurred among the payloads tested; this does not establish immunity against attacks outside this specific test set.

7. Discussion

On RQ1 and RQ3, encryption and Merkle overhead is measurable but bounded: the complete framework adds 21.81 milliseconds over the unprotected baseline at 10,000 records, an increase of about 87.6 percent that is statistically significant (Table 7) yet remains under 50 milliseconds in absolute terms. All 700 controlled tampering trials across seven scenarios were detected, with detection correctly attributed to three distinct mechanisms operating at different layers: GCM authentication for record level modification, Merkle root mismatch for dataset level inconsistency, and RSA PSS verification for the integrity of the root itself. This result holds only for the tested scenarios and trial counts; stale root and replay attacks were not evaluated and remain an open item.

On RQ2, the differential privacy results confirm the expected privacy utility trade off: error falls as epsilon rises for every query, consistent with the Laplace mechanism's noise scale shrinking as epsilon grows. Because Q1 and Q2 share the same sensitivity, their absolute error is identical at every epsilon, and only their relative error, which normalizes by each query's true value, distinguishes them. The clipping bounds chosen for Q3 through Q5 turned out to be non binding for this dataset, meaning the reported utility reflects the Laplace mechanism operating without any contribution clipping actually occurring, a fact that should be read as a property of this dataset rather than of the mechanism. The absence of confidence intervals around the reported MAE and RMSE values is a genuine limitation of what was retained from the experimental run, not a limitation of the differential privacy protocol itself, which did execute the full 100 repetition draw.

On RQ4, the persistent accountant enforced sequential composition correctly, including under restart and concurrent access, which is the practical

precondition for the privacy budget to mean anything operationally rather than only mathematically.

The non monotonic throughput pattern in Table 5, where 5,000 records outperforms 10,000, is reported as measured without further adjustment. It plausibly reflects Java virtual machine and just in time compilation state, garbage collection timing, database or operating system scheduling, batching behavior, or fixed per batch overhead, but because none of these were separately instrumented in this study, no single cause is claimed. This is a controlled single machine prototype evaluation on one fixed hardware and software configuration; it establishes feasibility and comparative overhead within that environment, not universal production scale throughput or behavior under distributed or heterogeneous deployment. Compared with more advanced systems such as FLEX or computational differential privacy encrypted databases [10]-[12], this framework is intentionally simpler, and its contribution is making the boundaries between encryption, integrity, authorization, and inferential privacy explicit and separately measurable within one conventional application stack.

All figures reported in Sections 6 and 7 are taken directly from the experiment runner's CSV output on the reference environment in Table 4, with the single arithmetic transformation used in each case, namely the throughput and overhead calculations, stated explicitly at the point where it is used, so that a reader working from the same dataset, seed, and configuration can reproduce each table without needing to infer an undocumented processing step.

8. Limitations and Future Work

This study has several limitations that should be read alongside its results. Evaluation used synthetic Synthea derived data rather than real clinical records, and the primary dataset contained 10,000 patient level records, so substantially larger scale behavior was not evaluated. All experiments ran on one machine, an Intel Core i7-12700H with 14 cores, 20 threads, and 16 gigabytes of DDR4 memory, rather than a distributed or containerized deployment, and the 30 repetition and 100 draw protocols characterize variability within this one environment rather than across heterogeneous hardware. Confidence intervals around the differential privacy MAE and RMSE values could not be reconstructed, since only repetition level summaries rather than raw per draw observations were retained from the experimental run. The key encryption key is stored in an external PKCS#12 keystore rather than a hardware security module or cloud key management service, and full compromise of the application host, exposing both the keystore and its environment variable password, sits outside the modeled threat boundary. Stale root and replay attacks against the integrity ledger were not evaluated, the differential privacy layer supports only the defined count, bounded mean, and bounded sum query classes rather than arbitrary SQL analytics, and the 140 attempt security validation demonstrates no bypass within that specific test set rather than general immunity.

Future work should evaluate the framework at larger dataset scales and across multiple machines, including containerized and cloud deployment and a distributed database backend, test stale root and replay attacks explicitly, move key encryption key storage to a hardware security module or cloud key management service, retain raw per draw differential privacy observations to enable confidence interval reporting, and extend query support beyond count, bounded mean, and bounded sum, for example through epsilon delta differential privacy using the Gaussian mechanism.

9. Conclusion

This paper presented a layered privacy preserving and verifiable data access framework integrating AES 256 GCM authenticated encryption, SHA 256 Merkle tree integrity verification with RSA PSS root authentication, role based access control, a custom Laplace differential privacy mechanism, and a persistent PostgreSQL backed privacy accountant, implemented as a Java 21, Spring Boot 3.x, and PostgreSQL 16.4 prototype. All 700 controlled tampering attempts across seven evaluated scenarios were detected, with detection correctly attributed to its respective mechanism. Differential privacy error decreased monotonically as epsilon increased across all five evaluated query classes, and the persistent accountant correctly enforced sequential composition under restart and concurrent access. On the 10,000 record dataset, the complete framework added 21.81 milliseconds of mean end to end latency over an unprotected baseline, an increase of about 87.6 percent that is statistically significant yet practically bounded. The contribution of this work is the integrated implementation and empirical evaluation of encryption, verifiable integrity, role controlled access, differential privacy, persistent privacy accounting, and auditing within one architecture and one experimental protocol, rather than the invention of any individual cryptographic or differential privacy primitive, each of which is adopted from established, cited sources.

REFERENCES

-
- [1] J. Near, D. Darais, N. Lefkovitz, and G. Howarth, "Guidelines for Evaluating Differential Privacy Guarantees," NIST Special Publication 800-226, Mar. 2025, doi: 10.6028/NIST.SP.800-226.
- [2] C. Dwork, "Differential Privacy," in Proc. 33rd Int. Colloquium on Automata, Languages and Programming (ICALP), vol. 4052, 2006, pp. 1-12, doi: 10.1007/11787006_1.
- [3] C. Dwork and A. Roth, "The Algorithmic Foundations of Differential Privacy," Foundations and Trends in Theoretical Computer Science, vol. 9, nos. 3-4, pp. 211-407, 2014, doi: 10.1561/0400000042.
- [4] National Institute of Standards and Technology, M. J. Dworkin, M. Sonmez Turan, and N. Mouha, "Advanced Encryption Standard (AES)," FIPS 197-upd1, May 2023, doi: 10.6028/NIST.FIPS.197-upd1.
- [5] M. Dworkin, "Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC," NIST Special Publication 800-38D, Nov. 2007, doi: 10.6028/NIST.SP.800-38D.
- [6] E. Barker and N. Mouha, "Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher," NIST Special Publication 800-67 Rev. 2, Nov. 2017, withdrawn Jan. 1, 2024.
- [7] R. C. Merkle, "A Digital Signature Based on a Conventional Encryption Function," in Advances in Cryptology, CRYPTO 1987, vol. 293, pp. 369-378, doi: 10.1007/3-540-48184-2_32.
- [8] F. D. McSherry, "Privacy Integrated Queries: An Extensible Platform for Privacy Preserving Data Analysis," in Proc. ACM SIGMOD Int. Conf. Management of Data, 2009, pp. 19-30, doi: 10.1145/1559845.1559850.
- [9] D. Proserpio, S. Goldberg, and F. McSherry, "Calibrating Data to Sensitivity in Private Data Analysis," Proc. VLDB Endowment, vol. 7, no. 8, pp. 637-648, 2014, doi: 10.14778/2732296.2732300.
- [10] N. Johnson, J. P. Near, and D. Song, "Towards Practical Differential Privacy for SQL Queries," Proc. VLDB Endowment, vol. 11, no. 5, pp. 526-539, 2018, doi: 10.1145/3177732.3177733.
- [11] R. J. Wilson, C. Y. Zhang, W. Lam, D. Desfontaines, D. Simmons-Marengo, and B. Gipson, "Differentially Private SQL with Bounded User Contribution," Proceedings on Privacy Enhancing Technologies, vol. 2020, no. 2, pp. 230-250, 2020, doi: 10.2478/popets-2020-0025.
- [12] F. Alborch Escobar, S. Canard, F. Laguillaumie, and D. H. Phan, "Computational Differential Privacy for Encrypted Databases Supporting Linear Queries," Proceedings on Privacy Enhancing Technologies, vol. 2024, no. 4, pp. 583-604, 2024, doi: 10.56553/popets-2024-0131.
- [13] J. Walonoski, M. Kramer, J. Nichols, A. Quina, C. Moesel, D. Hall, C. Duffett, K. Dube, T. Gallagher, and S. McLachlan, "Synthea: An approach, method, and software mechanism for generating synthetic patients and the synthetic electronic health care record," Journal of the American Medical Informatics Association, vol. 25, no. 3, pp. 230-238, 2018, doi: 10.1093/jamia/ocx079.
- [14] OWASP Foundation, "SQL Injection Prevention Cheat Sheet," OWASP Cheat Sheet Series. Available: https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html (accessed 2026).
- [15] OWASP Foundation, "Cross Site Scripting Prevention Cheat Sheet," OWASP Cheat Sheet Series. Available: https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html (accessed 2026).