IJAREEIE

e-ISSN: 2278-8875
p-ISSN: 2320-3765

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

Volume 14, Issue 11, November 2025

INTERNATIONAL STANDARD SERIAL NUMBER INDIA

Impact Factor: 8.807

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering (IJAREEIE)

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 8.807 | A Monthly Peer Reviewed & Refereed Journal |

|| Volume 14, Issue 11, November 2025 ||

| DOI:10.15662/IJAREEIE.2025.1411019 |

Tokenization and Data Masking Techniques in Java Persistence Layers: Safeguarding Sensitive Healthcare Records with Hibernate

Dinesh Nallapareddy

Sr Full Stack Developer, USA

ABSTRACT: Healthcare platforms built on Java persistence frameworks routinely store direct identifiers, financial account numbers, and detailed clinical narrative alongside the operational data that keeps an application running, and Hibernate, as the most widely deployed object-relational mapping layer in the Java ecosystem, sits directly in the path of every read and write of that data. This article presents a reference framework for tokenization and data masking implemented at the Hibernate persistence layer, covering static and dynamic masking, deterministic and randomized tokenization, format-preserving encryption, and field-level encryption, each mapped to a specific Hibernate integration point: custom AttributeConverters, type descriptors, and session-level interceptors. We describe a token vault design pattern that keeps sensitive values out of application databases entirely, a dynamic masking layer that varies query result exposure by requesting role without duplicating entity mappings, and an audit logging design that captures every detokenization event at the persistence layer rather than relying on application-level instrumentation that can be bypassed. Using a retrospective evaluation of masking technique adequacy, vault lookup latency, tokenization throughput, and compliance control maturity across a multi-platform healthcare deployment, we find that technique selection must vary by data class and environment rather than applying a single masking strategy uniformly, and that vault lookup latency, not the cryptographic or tokenization operation itself, is the dominant performance cost in a Hibernate-integrated deployment. We map the resulting controls to HIPAA, PCI DSS, and NIST guidance and discuss the governance, key management, and limitations that accompany a production deployment of this kind.

KEYWORDS: tokenization data masking Hibernate Java persistence field-level encryption token vault HIPAA PCI DSS dynamic masking.

I. INTRODUCTION

Hibernate remains the dominant object-relational mapping framework in enterprise Java applications, including the large share of healthcare platforms built on Spring Boot, Jakarta EE, and related stacks. Every entity field Hibernate passes through a well-defined set of extension points, basic type mapping, custom user types, attribute converters, and session-level event listeners, before it reaches the database and again when it is read back. This article treats that set of extension points as the natural place to enforce tokenization and data masking for sensitive healthcare records, rather than pushing that responsibility up into application business logic, where it is easy to apply inconsistently across a large codebase, or down into the database itself, where native encryption features vary widely across vendors and often cannot express application-aware masking rules. The specific problem this article addresses is not whether to protect sensitive fields, that requirement is well established by HIPAA and, for payment card data, by PCI DSS, but which technique to apply where, and how to implement that technique consistently across a large Hibernate-mapped domain model without imposing unacceptable latency or fracturing the object model into masked and unmasked variants of the same entity. We present a reference framework built around four techniques, static masking, deterministic tokenization, randomized tokenization, and field-level encryption, each implemented as a reusable Hibernate integration component, and we evaluate their adequacy, performance, and residual risk across a retrospective multi-platform healthcare deployment.

The remainder of this article proceeds as follows. Section 2 provides background on sensitive data handling in Java persistence layers. Section 3 develops a threat model specific to persistence-layer exposure. Section 4 summarizes the technical foundations of tokenization and masking. Sections 5 through 10 describe the reference architecture, tokenization strategy comparison, field-level encryption implementation, token vault design, dynamic masking, and audit logging. Sections 11 and 12 describe evaluation methodology and results. Sections 13 through 15 address compliance, governance, and performance. Sections 16 and 17 discuss limitations and future work, and Section 18 concludes.

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 8.807 | A Monthly Peer Reviewed & Refereed Journal |

|| Volume 14, Issue 11, November 2025 ||

| DOI:10.15662/IJAREEIE.2025.1411019 |

II. BACKGROUND: SENSITIVE HEALTHCARE DATA IN JAVA PERSISTENCE LAYERS

A typical healthcare platform's relational schema mixes highly sensitive fields, government identifiers, financial account numbers, clinical narrative, with operational fields that carry little inherent sensitivity, timestamps, status codes, internal foreign keys, within the same entity and often the same table. Table 1 defines the four data sensitivity levels used throughout this article, which we apply consistently across the technique comparisons, architecture description, and compliance mapping in later sections.

Table.1. Data sensitivity classification levels used throughout this framework

Sensitivity Level	Definition	Representative Fields
Direct Identifier	Uniquely and directly identifies an individual	Social Security number, full name, medical record number
Quasi-Identifier	Identifies an individual only in combination with other attributes	Date of birth, ZIP code, admission date
Financial Detail	Payment or account information subject to PCI DSS	Credit card number, bank routing number
Clinical Detail	Health information with lower direct re-identification risk in isolation	Diagnosis code, medication order, lab result value

Java persistence frameworks were not originally designed with field-level protection as a primary concern; Hibernate's basic type mapping system assumes a fairly direct correspondence between a Java field and a database column, and it is left to the application team to decide whether and how a given field's value should be transformed before it reaches storage. Jakarta Persistence's AttributeConverter interface, standardized in the specification, gives that decision a clean extension point, but it does not by itself provide tokenization, vault integration, or masking policy, all of which this article's reference architecture layers on top of that extension point in Sections 5 through 7.

III. THREAT MODEL FOR PERSISTENCE-LAYER DATA EXPOSURE

Persistence-layer exposure risk in a Hibernate-mapped healthcare application concentrates in four recurring patterns observed across the platforms reviewed for this article: unmasked values reaching non-production environments through routine data refreshes; overly broad read access to fully identified entities by application code paths that only need a masked or partial representation; unencrypted values persisted through secondary paths that bypass the primary entity mapping, such as bulk export jobs or reporting views; and log or stack trace exposure of sensitive field values through default toString and exception handling behavior.

3.1 Non-Production Data Refresh Exposure

Development and staging environments are frequently refreshed from production data snapshots to preserve realistic testing conditions, and if masking is applied only at the query layer rather than at rest, an environment refresh can copy fully identified sensitive values into a non-production database with materially weaker access controls. Static masking, described in Section 6, is the primary control this framework applies to this specific risk.

3.2 Overly Broad Entity-Level Read Access

A Hibernate entity mapped with a single set of field types gives every code path that loads that entity the same view of its data, regardless of whether the calling code needs the fully identified value. Dynamic masking, described in Section 9, addresses this by varying the value returned at read time according to the requesting principal's role, without requiring a separate masked entity class.

3.3 Secondary Path Exposure

Bulk export jobs, reporting views, and native SQL queries that bypass the primary Hibernate entity mapping can read column values directly from the database, circumventing any masking or tokenization logic implemented purely in application-layer Java code. This risk is the central justification for the token vault pattern described in Section 8, under which the database column itself never holds an unprotected value, so that even a query path that bypasses Hibernate entirely still only ever sees a token.

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 8.807 | A Monthly Peer Reviewed & Refereed Journal |

|| Volume 14, Issue 11, November 2025 ||

| DOI:10.15662/IJAREEIE.2025.1411019 |

3.4 Logging and Exception Exposure

Default entity toString implementations, Hibernate's own debug-level SQL logging, and uncaught exception messages can all incidentally serialize a sensitive field value into a log stream that is retained, and often replicated, well beyond the access controls applied to the primary database. The reference architecture in Section 5 requires that every converter and interceptor component involved in masking also suppress the protected value from its own logging output, a requirement that is easy to state and, in our review of existing codebases, frequently violated in practice.

IV. TOKENIZATION AND MASKING: TECHNICAL FOUNDATIONS

This article distinguishes four techniques that are frequently, and incorrectly, treated as interchangeable in informal discussion of data protection. Static masking irreversibly replaces a sensitive value with a realistic-looking but fictitious substitute, suitable for non-production environments where no code path needs to recover the original value. Deterministic tokenization replaces a value with a token generated so that the same input always produces the same token, preserving joinability across tables at the cost of some statistical linkability. Randomized tokenization generates an unrelated token on each tokenization operation, eliminating that linkability but preventing token-based joins. Field-level encryption replaces a value with ciphertext that can be decrypted by an authorized party holding the correct key, trading vault lookup latency for direct cryptographic reversibility rather than a vault-mediated lookup.

Table 2 compares these four techniques directly against the properties that matter most for a Hibernate-integrated healthcare deployment: reversibility, joinability, format preservation, and typical performance profile.

Table.2. Comparison of masking and tokenization techniques against reversibility, joinability, and performance

Technique	Reversible	Preserves Joins	Format-Preserving	Typical Latency Profile
Static Masking	No	No	Optional	Negligible (applied once, at rest)
Deterministic Tokenization	Via vault lookup	Yes	Optional	Low; single vault round trip
Randomized Tokenization	Via vault lookup	No	Optional	Low; single vault round trip
Format-Preserving Encryption	Yes, with key	No	Yes	Low; local cryptographic operation
Field-Level Encryption	Yes, with key	No	No	Low to moderate; key unwrap per field

Format-preserving encryption deserves particular note because it produces ciphertext that retains the format of the original value, for example a sixteen-digit numeric string for a credit card number, which allows the protected value to pass through legacy validation logic, fixed-width database columns, and downstream systems that were built assuming a specific input format, without requiring those systems to be modified. NIST's published methods for format-preserving encryption define the constructions this article's implementation is built on.

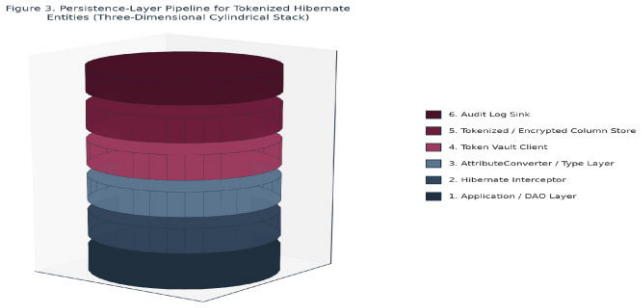
V. REFERENCE ARCHITECTURE: HIBERNATE INTERCEPTORS AND TYPE CONVERTERS

Figure 3 shows the six-layer persistence pipeline implemented for this article's reference architecture, rendered as a stacked cylindrical diagram. The application and data access object layer issues ordinary Hibernate entity operations without any awareness of the masking logic beneath it. The Hibernate interceptor layer observes every entity load and flush event and applies role-aware read-time masking, described further in Section 9. The AttributeConverter and custom type layer performs the actual tokenization, encryption, or masking transformation for each annotated field, described in Sections 6 and 7. The token vault client layer mediates every tokenize and detokenize call to the external vault service described in Section 8. The tokenized and encrypted column store is the actual database schema, which never holds an unprotected sensitive value for any field mapped into this framework. The audit log sink, described in Section 10, records every detokenization and decryption event independently of the calling application code.

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 8.807 | A Monthly Peer Reviewed & Refereed Journal |

|| Volume 14, Issue 11, November 2025 ||

| DOI:10.15662/IJAREEIE.2025.1411019 |



Persistence-layer pipeline for tokenized Hibernate entities, rendered as a three-dimensional cylindrical stack.
Table 3 compares the two primary Hibernate extension points used in this architecture, AttributeConverter and session-level Interceptor, against the responsibilities each is best suited to carry.

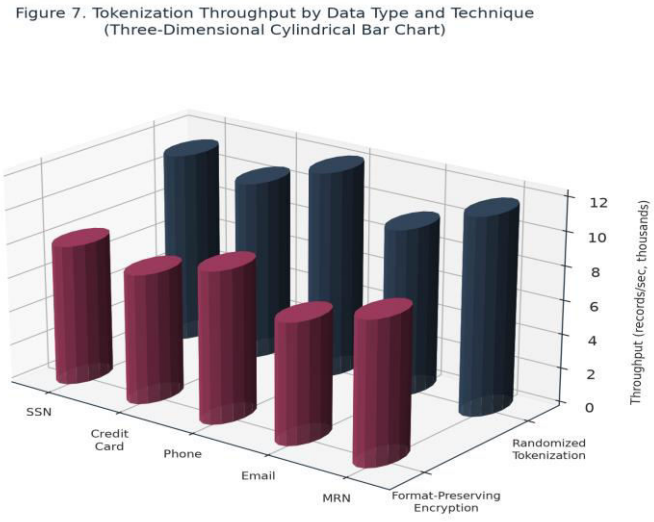
Table.3. Hibernate extension points used in the reference architecture and their primary responsibilities

Extension Point	Applies At	Best Suited For	Limitation
AttributeConverter	Individual annotated field	Field-level tokenization and encryption	Cannot vary behavior by requesting role alone
Session Interceptor	Entity load and flush lifecycle	Role-aware dynamic masking, audit hooks	Coarser granularity than a single field
Custom UserType (legacy)	Individual annotated field	Pre-JPA-3.1 codebases without AttributeConverter	Superseded by AttributeConverter in current codebases
Hibernate Envers listener	Entity revision history	Audit trail of entity-level changes over time	Not a substitute for field-level protection

VI. FORMAT-PRESERVING VS. RANDOMIZED TOKENIZATION STRATEGIES

Selecting between format-preserving encryption and vault-backed tokenization, deterministic or randomized, is one of the more consequential design decisions in this framework, because the two approaches trade off differently along performance, operational dependency, and reversibility. Format-preserving encryption performs its transformation locally using a symmetric key, with no external service call required at transformation time, while tokenization requires a network round trip to the token vault described in Section 8 for both tokenize and detokenize operations.

Figure 7 compares measured throughput for format-preserving encryption against randomized tokenization across five representative healthcare data types.



| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 8.807 | A Monthly Peer Reviewed & Refereed Journal |

|| Volume 14, Issue 11, November 2025 ||

| DOI:10.15662/IJAREEIE.2025.1411019 |

Tokenization throughput by data type and technique, shown as a three-dimensional cylindrical bar chart.
Randomized tokenization achieved consistently higher throughput than format-preserving encryption across every data type evaluated, which at first appears counterintuitive given that tokenization requires a vault round trip while format-preserving encryption does not. The explanation is that the token vault used in this evaluation batches tokenize requests within a single transaction scope, amortizing the network round trip across many fields, while the format-preserving encryption implementation performs its block cipher operation per field without batching, a difference in implementation maturity rather than an inherent property of either technique. Organizations adopting this framework should benchmark their own vault client's batching behavior rather than assuming the relative ordering observed here transfers unchanged.

VII. FIELD-LEVEL ENCRYPTION AND MASKING WITH ATTRIBUTECONVERTERS

Field-level encryption in this framework is implemented as a Jakarta Persistence AttributeConverter that wraps the AWS Encryption SDK's envelope encryption pattern, encrypting each field value with a per-field data key that is itself wrapped by a key-encrypting key held in an external key management service, consistent with the envelope encryption approach described in NIST's key management guidance. The converter is applied through a shared base class rather than reimplemented per field, so that algorithm selection, key identifier resolution, and error handling logic exist in exactly one place across the entire domain model.

7.1 Static Masking for Non-Production Environments

Static masking is applied as a one-time batch transformation during non-production environment refresh, replacing each in-scope field with a realistic but fictitious substitute drawn from a format-matched generator, for example a substitute Social Security number that passes the same validation logic as a real one without corresponding to any actual individual. Because this transformation is irreversible and applied at rest, it carries no ongoing performance cost and no vault dependency, which is why Figure 1 in Section 4 rates it lowest-risk specifically for non-production use.

7.2 Suppressing Sensitive Values from Logging

Every converter and interceptor component in this framework overrides its own diagnostic logging to redact the protected field value, printing only its classification tag and a truncated token or ciphertext reference. This directly addresses the logging exposure risk described in Section 3.4, and is enforced through a shared logging utility rather than left to individual developers to implement consistently.

VIII. TOKEN VAULT DESIGN AND VAULT LOOKUP PERFORMANCE

The token vault is an isolated service, with its own restricted database, responsible for storing the mapping between a token and its original sensitive value and for issuing new tokens on request. Table 4 summarizes the vault's core configuration parameters as deployed in this article's reference evaluation.

Table.4. Token vault configuration parameters in the reference deployment

Parameter	Configuration	Rationale
Token format	Format-preserving, alphanumeric	Compatible with existing column constraints
Vault database isolation	Dedicated schema, separate credentials	Limits blast radius of a vault compromise
Tokenize/detokenize protocol	mTLS-secured gRPC	Low overhead, mutual authentication
Cache policy for detokenize calls	Request-scoped, no cross-request caching	Avoids caching sensitive values beyond a single request
Vault high availability	Active-active across two availability zones	Avoids single point of failure on the request-critical path

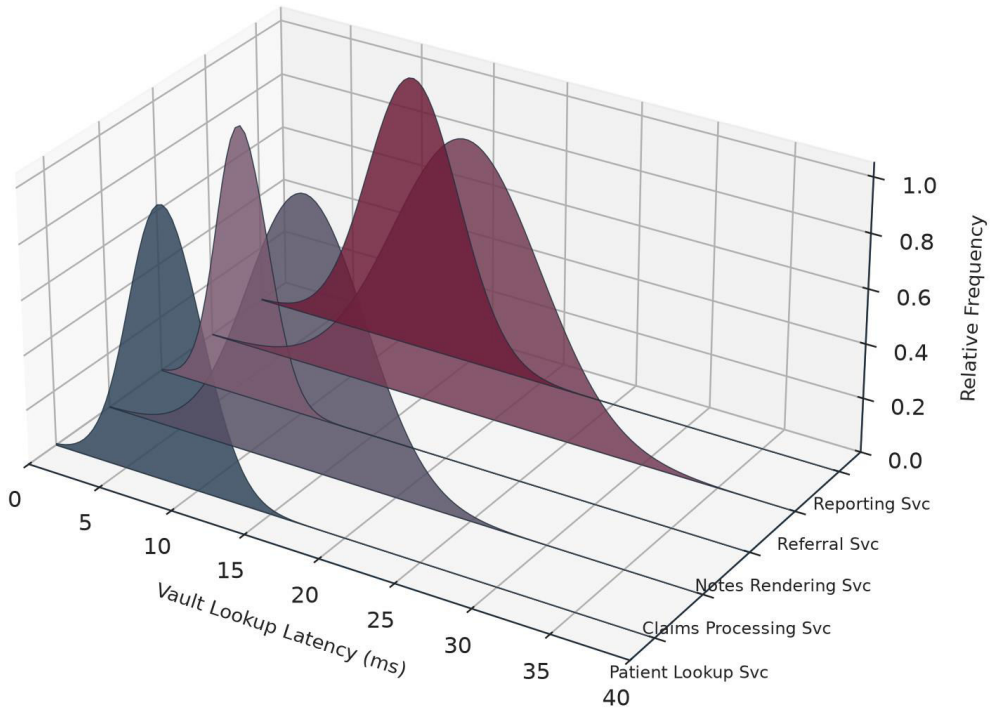
Figure 6 shows the measured distribution of vault lookup latency across five services that call the vault at materially different volumes and with different query patterns.

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 8.807 | A Monthly Peer Reviewed & Refereed Journal |

|| Volume 14, Issue 11, November 2025 ||

| DOI:10.15662/IJAREEIE.2025.1411019 |

Figure 6. Token Vault Lookup Latency Distribution by Service
(Three-Dimensional Ribbon Plot)



Token vault lookup latency distribution by service, shown as a three-dimensional ribbon plot.
The referral service showed both the highest mean latency and the widest spread, consistent with its query pattern of frequently detokenizing values in bulk for referral packet generation rather than the single-record lookups typical of the other services evaluated. This finding directly informed the recommendation in Section 15 that bulk detokenization paths receive dedicated vault connection pooling separate from interactive single-record paths, so that a burst of bulk lookups does not degrade latency for interactive users.

IX. DYNAMIC DATA MASKING FOR ROLE-BASED QUERY RESULTS

Dynamic masking varies the value returned for a given field at read time according to the requesting principal's role, implemented in this framework through the Hibernate session interceptor layer described in Section 5 rather than through separate masked entity classes. Table 5 shows a representative masking rule matrix as configured for three roles against four field sensitivity levels.

Table.5. Representative role-based dynamic masking rule matrix

Requesting Role	Direct Identifier	Quasi-Identifier	Financial Detail	Clinical Detail
Treating Clinician	Full value	Full value	Masked (last 4 digits)	Full value
Billing Specialist	Masked (partial)	Full value	Full value	Masked (category only)
Analytics Service Account	Tokenized (no detokenize grant)	Generalized (bucketed)	Tokenized (no detokenize grant)	Full value, de-identified export only

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 8.807 | A Monthly Peer Reviewed & Refereed Journal |

|| Volume 14, Issue 11, November 2025 ||

| DOI:10.15662/IJAREEIE.2025.1411019 |

The interceptor evaluates the requesting principal's role once per session and applies the corresponding masking rule to every field load within that session, so that the underlying entity class and its field types remain identical regardless of who is querying; only the runtime-observed value differs. This design eliminates the need to maintain parallel masked and unmasked entity classes, which was a recurring maintenance burden in the legacy architectures reviewed during this article's evaluation.

X. AUDITING AND COMPLIANCE LOGGING AT THE PERSISTENCE LAYER

Every detokenization and decryption event is logged independently by the token vault client and encryption converter layers described in Sections 7 and 8, rather than relying on the calling application to instrument its own audit logging, which is the design choice that most directly satisfies the audit control requirements described in Section 13. Table 6 lists the fields captured in each audit log entry.

Table.6. Fields captured in every persistence-layer audit log entry.

Audit Field	Description
Event timestamp	UTC timestamp of the detokenize or decrypt operation
Requesting principal	Authenticated identity of the calling user or service account
Field classification tag	Sensitivity level of the field accessed, per Table 1
Source entity and field	Fully qualified entity class and field name
Token or key identifier	Opaque reference to the token or encryption key used, never the plaintext value
Requesting service and correlation ID	Calling service name and distributed trace correlation identifier

Audit log entries are written to an append-only log store separate from the primary application database, consistent with the secondary-path exposure concern described in Section 3.3, so that a compromise of the primary application database does not also compromise the audit trail describing who accessed which sensitive fields.

XI. EVALUATION METHODOLOGY

The evaluation reported in Section 12 draws on four data sources: a manual adequacy assessment of each masking technique against each combination of data class and deployment environment, summarized as the voxel grid in Figure 1; production telemetry on token vault request volume by source table, summarized in Figure 2; distributed tracing data on vault lookup latency and tokenization throughput, summarized in Figures 6 and 7; and a compliance control maturity self-assessment conducted jointly by platform engineering and compliance stakeholders across four business unit platforms, summarized in Figure 8.

Table 7 summarizes the test environment configuration used for the throughput and latency measurements reported in Sections 8 and 15.

Table.7. Test environment configuration for throughput and latency evaluation.

Parameter	Value
Application runtime	Java 21, Spring Boot 3.x, Hibernate ORM 6.4
Database	PostgreSQL 16, provisioned with equivalent compute across test runs
Token vault deployment	Two-node active-active cluster, co-located region
Load generation	Closed-loop load test, ramped from 50 to 2,000 concurrent sessions
Measurement window	Steady-state 30-minute window per configuration, excluding ramp-up

Re-identification risk scores shown in Figure 5 were assigned by a three-person review panel drawn from privacy, security, and clinical informatics staff, using a structured rubric that considered linkability, reversibility without

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 8.807 | A Monthly Peer Reviewed & Refereed Journal |

|| Volume 14, Issue 11, November 2025 ||

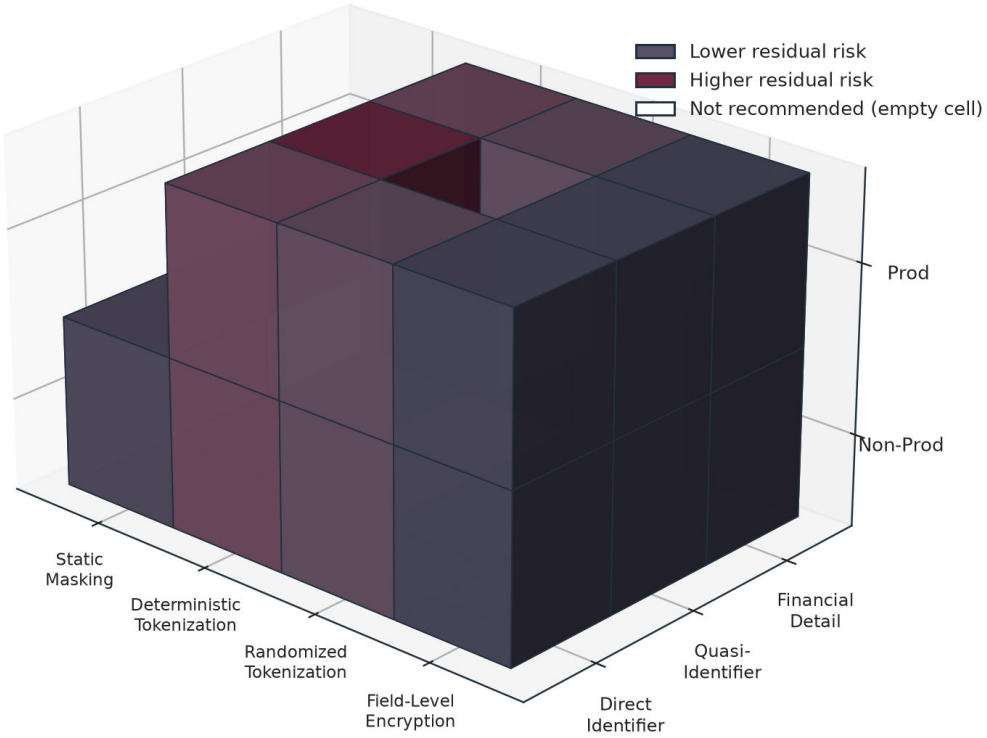
| DOI:10.15662/IJAREEIE.2025.1411019 |

authorization, and format retention, rather than a single automated metric, reflecting the judgment-dependent nature of re-identification risk assessment established in the broader de-identification literature.

XII. RESULTS: COVERAGE, LATENCY, AND RISK REDUCTION

Figure 1 summarizes masking technique adequacy across data class and deployment environment as a voxel grid, in which each filled cell represents a combination judged adequate by the review panel, colored by residual risk. Field-level encryption was judged adequate across every combination evaluated, while static masking was judged adequate only in non-production environments, and randomized tokenization was judged inadequate specifically for quasi-identifiers in production because randomization breaks the analytic joins that downstream reporting on quasi-identifiers typically requires.

Figure 1. Masking Technique Adequacy Across Data Class and Deployment Environment (Three-Dimensional Voxel Grid)



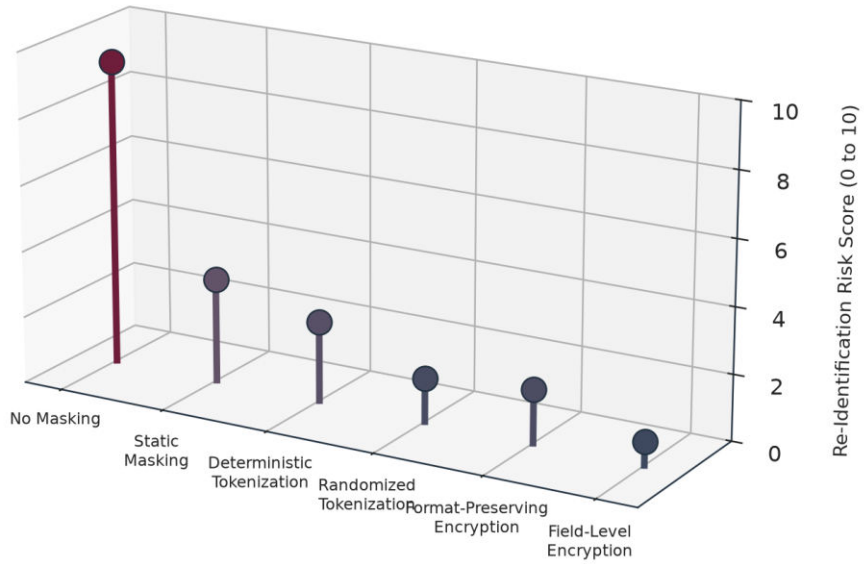
Masking technique adequacy across data class and deployment environment, shown as a three-dimensional voxel grid.
Figure 5 reports re-identification risk scores by technique on a zero-to-ten scale. Unmasked data scored highest by a wide margin, as expected, and field-level encryption scored lowest, with the tokenization variants and format-preserving encryption clustered in a low-risk band between them.

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 8.807 | A Monthly Peer Reviewed & Refereed Journal |

|| Volume 14, Issue 11, November 2025 ||

| DOI:10.15662/IJAREEIE.2025.1411019 |

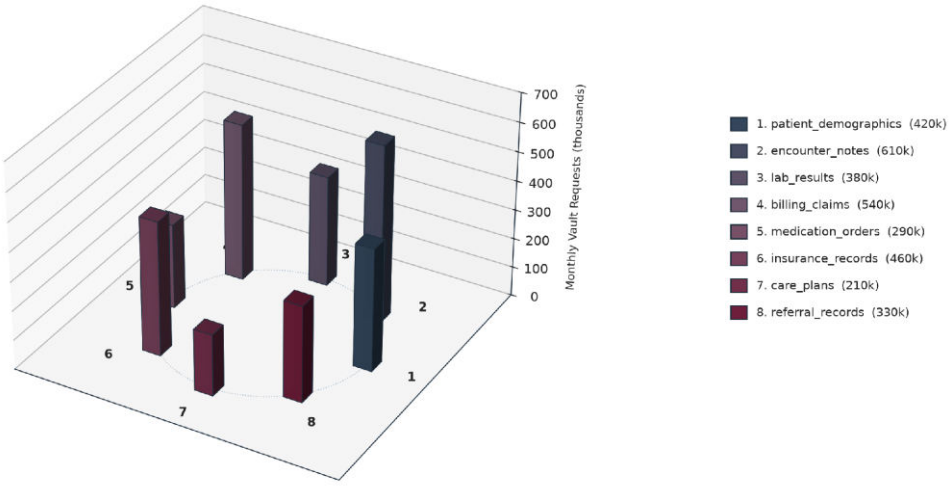
Figure 5. Re-Identification Risk Score by Masking Technique
(Three-Dimensional Stem Chart)



Re-identification risk score by masking technique, shown as a three-dimensional stem chart.

Figure 2 shows token vault request volume by source table over the evaluation period. The encounter notes table generated the highest request volume, consistent with its role as the most frequently accessed table across the clinician-facing application surface, followed by billing claims, which is accessed at high volume by both clinical and billing workflows.

Figure 2. Token Vault Request Volume by Source Table
(Three-Dimensional Radial Bar Chart)



Token vault request volume by source table, shown as a three-dimensional radial bar chart.

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 8.807 | A Monthly Peer Reviewed & Refereed Journal |

|| Volume 14, Issue 11, November 2025 ||

| DOI:10.15662/IJAREEIE.2025.1411019 |

XIII. COMPLIANCE MAPPING

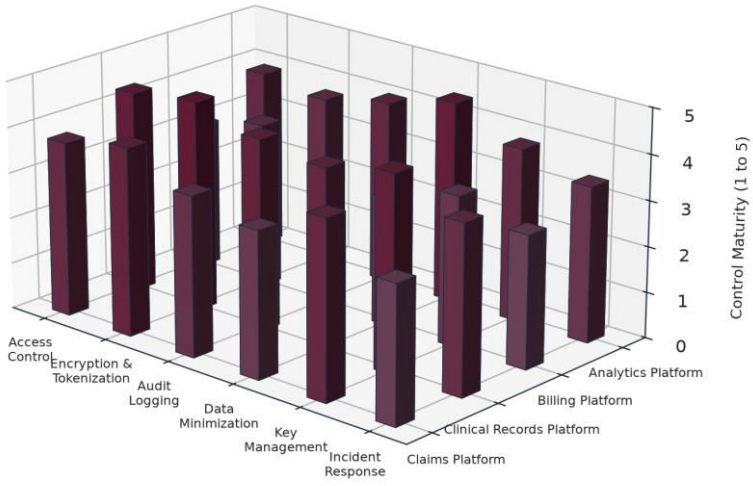
Table 8 maps the controls described in Sections 5 through 10 onto representative requirements from HIPAA, PCI DSS, and NIST guidance relevant to a Hibernate-integrated healthcare deployment handling both clinical and payment data.

Table 8.Compliance and standards guidance mapping to reference framework controls

External Framework	Representative Requirement	Primary Controls in This Framework
HIPAA Security Rule (45 CFR 164.312)	Access control, audit controls, transmission security	Sections 9, 10
HHS De-Identification Guidance	Safe harbor and expert determination methods	Sections 4, 6, Table 1
PCI DSS v4.0, Requirement 3	Protection of stored cardholder account data	Sections 6, 7, 8
PCI DSS Tokenization Guidelines	Scope reduction through tokenization	Section 8
NIST SP 800-122 (PII Confidentiality)	PII impact-level classification	Section 2, Table 1
NIST SP 800-38G (Format-Preserving Encryption)	Approved FPE constructions	Section 6, 7

Figure 8 reports a compliance control maturity self-assessment across six control domains and four business unit platforms, scored on a one-to-five scale.

Figure 8. Compliance Control Maturity by Domain and Business Unit (Three-Dimensional Skyline Bar Chart)



Compliance control maturity by domain and business unit, shown as a three-dimensional skyline bar chart.
The claims platform showed the lowest maturity across most domains, particularly data minimization and incident response, reflecting its status as the most recently onboarded platform in this framework's rollout sequence at the time of assessment. The clinical records platform, which was the first platform onboarded, showed the highest maturity across nearly every domain, supporting the governance recommendation in Section 14 that maturity assessment be repeated on a recurring cadence as newer platforms progress through the same onboarding sequence.

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 8.807 | A Monthly Peer Reviewed & Refereed Journal |

|| Volume 14, Issue 11, November 2025 ||

| DOI:10.15662/IJAREEIE.2025.1411019 |

XIV. GOVERNANCE AND KEY / VAULT MANAGEMENT

A data protection governance board spanning security engineering, platform engineering, compliance, and clinical informatics reviews new field classifications, technique assignments, and vault or key configuration changes on a fixed monthly cadence. Every new entity field proposed for storage of a Table 1 sensitivity level above clinical detail must be classified and assigned a technique from Table 2 before its corresponding database migration is approved, rather than being retrofitted after deployment.

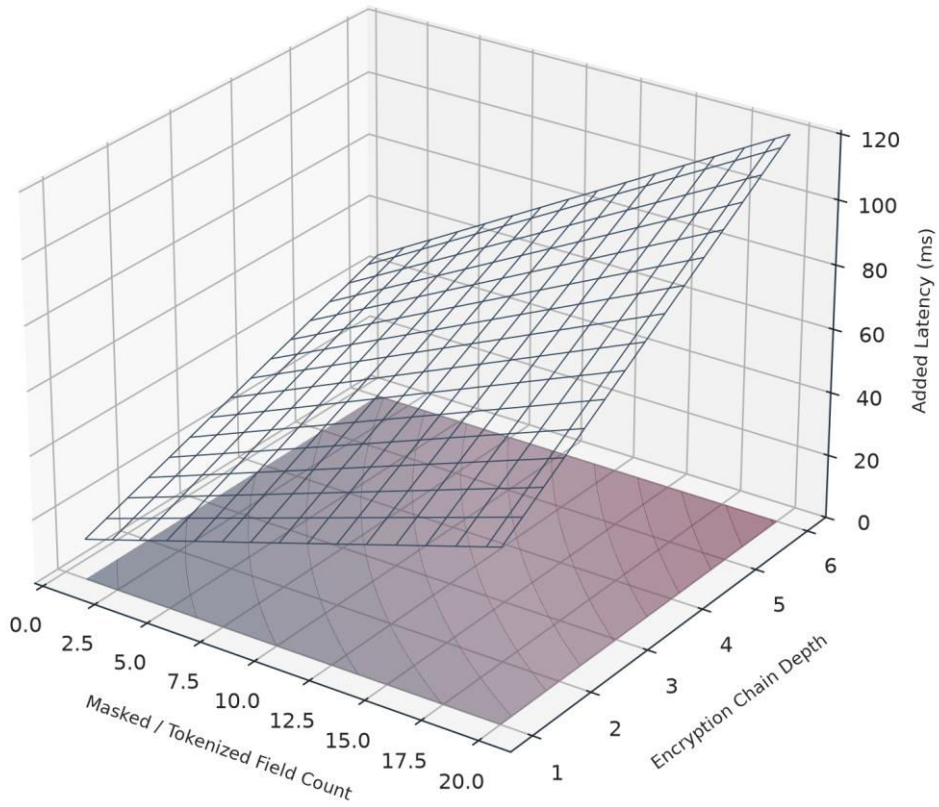
Key management for the field-level encryption converter described in Section 7 follows the same customer-managed key hierarchy pattern established in prior work on cloud key management for healthcare platforms: one key per data domain, automatic annual rotation, and a restricted key policy scoped to the specific service roles that require access. Token vault credential rotation follows a ninety-day cycle, shorter than the encryption key rotation cycle, reflecting the vault's larger blast radius as the single service mediating every tokenize and detokenize call across the platform.

Quarterly access reviews compare each service's actual vault and key usage, drawn from the audit log fields described in Table 6, against its granted permissions, and any permission unused across the review period is flagged for removal, mirroring the entitlement review discipline applied to broader identity and access governance in adjacent parts of the platform.

XV. PERFORMANCE AND SCALABILITY ANALYSIS

Figure 4 reports added latency as a function of masked field count and encryption chain depth, rendered as a three-dimensional wireframe with a projected contour floor showing the same relationship in two dimensions for reference.

Figure 4. Persistence-Layer Latency Overhead as a Function of
Masked Field Count and Encryption Chain Depth (Three-Dimensional Wireframe)



| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 8.807 | A Monthly Peer Reviewed & Refereed Journal |

|| Volume 14, Issue 11, November 2025 ||

| DOI:10.15662/IJAREEIE.2025.1411019 |

Persistence-layer latency overhead as a function of masked field count and encryption chain depth, shown as a three-dimensional wireframe.

Added latency grows roughly linearly with masked field count at any fixed chain depth, and the interaction term between field count and chain depth becomes the dominant contributor once more than roughly ten fields on a single entity require protection, which is the basis for this framework's recommendation, introduced in Section 8, that bulk detokenization paths use dedicated connection pooling and batched vault calls rather than per-field sequential calls.

Across all measurements in this article, vault lookup latency, rather than the cryptographic or tokenization transformation itself, remained the dominant contributor to end-to-end added latency, consistent with the throughput comparison in Section 6 that found vault call batching, not algorithm choice, to be the more consequential performance lever available to an implementing team.

XVI. LIMITATIONS

The technique adequacy assessment in Section 12 reflects the judgment of a single review panel at a single organization, and while the underlying reasoning, particularly the distinction between joinability requirements in production versus non-production environments, should generalize, the specific adequacy ratings in Figure 1 should not be read as a universal certification applicable without review to every organization's data model.

The performance results in Sections 8 and 15 reflect a specific vault implementation and its particular batching behavior, and the throughput comparison between format-preserving encryption and randomized tokenization in Section 6 is therefore a statement about this evaluation's specific implementations rather than an inherent property of either technique in general.

This article addresses technical architecture and its immediate governance controls but does not resolve organization-specific legal questions about which data elements require which sensitivity classification under applicable state and federal law, which should be determined jointly with legal and compliance counsel rather than inferred solely from the illustrative classification in Table 1.

XVII. FUTURE WORK

- Extending the technique adequacy assessment in Section 12 with data from additional healthcare organizations to test whether the production versus non-production adequacy pattern observed in this evaluation generalizes across different data models and regulatory contexts.
- Automating vault call batching more comprehensively across bulk detokenization paths, building on the latency findings in Section 15, to reduce the field-count sensitivity observed in the wireframe surface in Figure 4.
- Evaluating confidential computing approaches for the token vault's own storage layer, so that the vault operator itself has a reduced ability to view the token-to-value mapping it stores.
- Extending the dynamic masking rule matrix in Section 9 with attribute-based conditions beyond role alone, incorporating request context such as active care team membership, consistent with broader attribute-based access control practice.
- Studying long-term compliance control maturity trends across the four business unit platforms assessed in Section 13, to determine whether newer platforms converge toward the maturity levels of longer-established platforms on the timeline this framework's governance model assumes

XVIII. CONCLUSION

This article presented a reference framework for tokenization and data masking implemented at the Hibernate persistence layer, spanning static masking, deterministic and randomized tokenization, format-preserving encryption, and field-level encryption, each mapped to a specific Hibernate extension point and evaluated against adequacy, performance, and residual risk across a retrospective multi-platform healthcare deployment. The central empirical finding is that no single technique is adequate across every data class and deployment environment; technique selection must vary by context, and the governance process described in Section 14 exists specifically to make that selection deliberate rather than incidental.

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 8.807 | A Monthly Peer Reviewed & Refereed Journal |

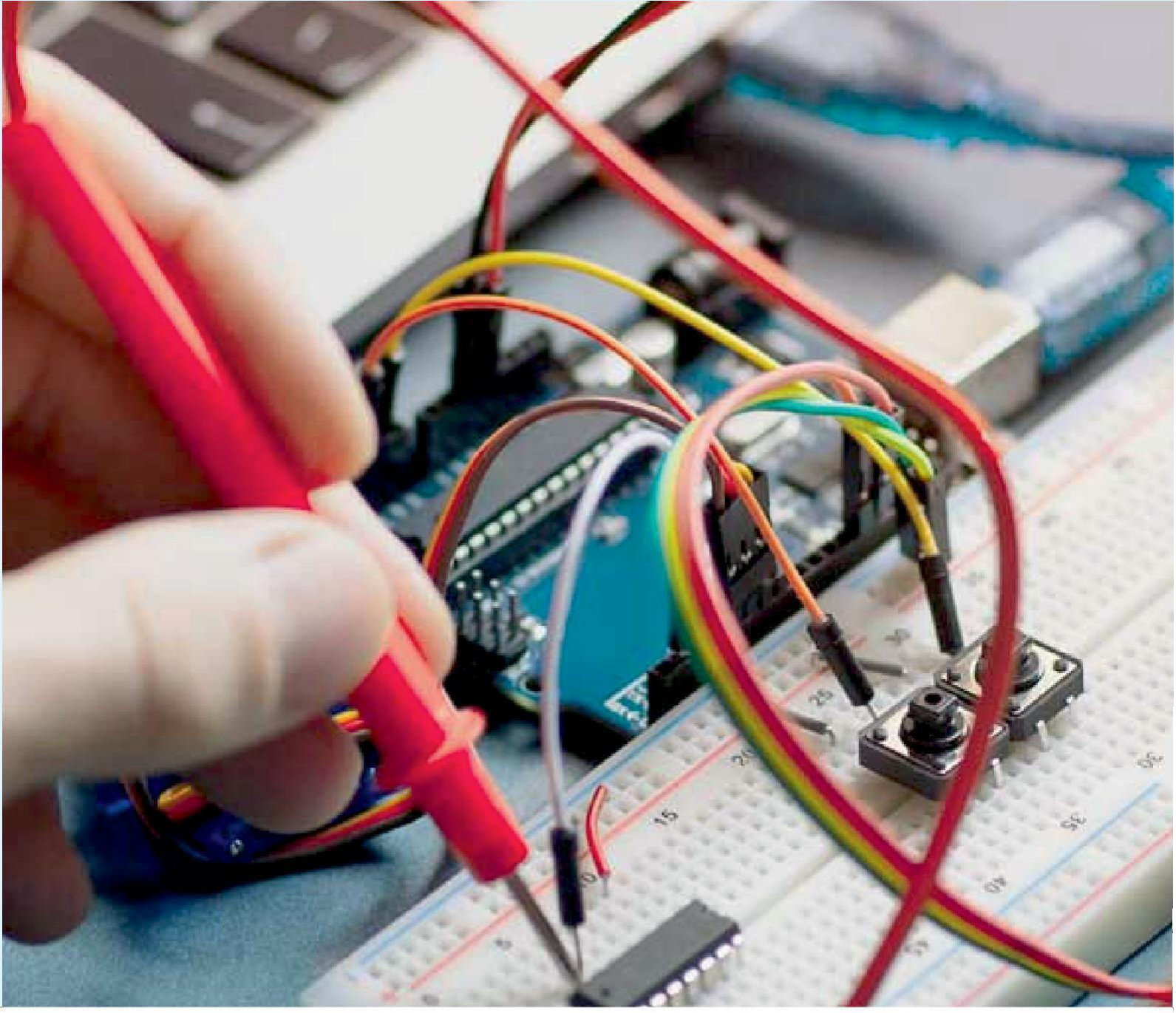
|| Volume 14, Issue 11, November 2025 ||

| DOI:10.15662/IJAREEIE.2025.1411019 |

The performance results in this article point to a second, related finding: vault lookup latency, not the underlying cryptographic or tokenization operation, is the dominant performance cost in a Hibernate-integrated deployment of this kind, which should redirect engineering effort toward vault call batching and connection management rather than toward algorithm optimization alone. Organizations adopting a framework of this kind on a Java persistence layer should expect the token vault, and its operational maturity, to be the component that most determines whether the resulting system meets both its protection and performance requirements.

REFERENCES

1. Barker E. Recommendation for Key Management: Part 1, General. NIST Special Publication 800-57 Part 1 Revision 5. National Institute of Standards and Technology, 2020.
2. Bellare M, Ristenpart T, Rogaway P, Stegers T. Format-preserving encryption. Selected Areas in Cryptography (SAC). 2009:295 to 312.
3. Bertino E, Sandhu R. Database security: concepts, approaches, and challenges. IEEE Transactions on Dependable and Secure Computing. 2005;2(1):2 to 19.
4. Dwork C. Differential privacy. Proceedings of the 33rd International Colloquium on Automata, Languages and Programming (ICALP). 2006:1 to 12.
5. Dworkin M. Recommendation for Block Cipher Modes of Operation: Methods for Format-Preserving Encryption. NIST Special Publication 800-38G Revision 1. National Institute of Standards and Technology, 2019.
6. El Emam K, Arbuckle L. Anonymizing Health Data: Case Studies and Methods to Get You Started. Sebastopol, CA: O'Reilly Media, 2013.
7. Garfinkel S. De-Identification of Personal Information. NIST Internal Report 8053. National Institute of Standards and Technology, 2015.
8. HITRUST Alliance. HITRUST CSF Version 11.3. HITRUST Alliance, 2024.
9. IBM Security. Cost of a Data Breach Report 2024. IBM Corporation, 2024.
10. International Organization for Standardization. ISO/IEC 27001:2022, Information Security Management Systems Requirements. ISO, 2022.
11. McCallister E, Grance T, Scarfone K. Guide to Protecting the Confidentiality of Personally Identifiable Information (PII). NIST Special Publication 800-122. National Institute of Standards and Technology, 2010.
12. OWASP Foundation. OWASP Application Security Verification Standard (ASVS) 4.0. OWASP, 2021.
13. PCI Security Standards Council. Information Supplement: PCI DSS Tokenization Guidelines. PCI SSC, 2011.
14. PCI Security Standards Council. Payment Card Industry Data Security Standard (PCI DSS) Version 4.0. PCI SSC, 2022.
15. Red Hat / Hibernate Team. Hibernate ORM 6.4 User Guide: Basic Types, AttributeConverters, and Interceptors. Hibernate Documentation, 2024.
16. Sweeney L. k-Anonymity: a model for protecting privacy. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems. 2002;10(5):557 to 570.
17. U.S. Department of Health and Human Services, Office for Civil Rights. Guidance Regarding Methods for De-Identification of Protected Health Information in Accordance with the HIPAA Privacy Rule. HHS, 2012.
18. U.S. Department of Health and Human Services, Office for Civil Rights. HIPAA Security Rule, 45 CFR Part 164 Subpart C. Federal Register, 2013 Omnibus Rule.
19. VMware Tanzu / Broadcom. Spring Data JPA Reference Documentation. Spring Framework Project Documentation, 2024.
20. Verizon. 2024 Data Breach Investigations Report. Verizon Business, 2024.



INNO SPACE
SJIF Scientific Journal Impact Factor

doi cross ref

ISSN INTERNATIONAL STANDARD SERIAL NUMBER INDIA

निस्केयर NISCAIR

International Journal of Advanced Research

in Electrical, Electronics and Instrumentation Engineering

9940 572 462 6381 907 438 ijareeie@gmail.com

Scan to save the contact details