

A Secure Architecture for Privacy-Preserving Audit of Banking LLM Chat Logs Using Application-Level Searchable Encryption

Ahmad Evander Ruizhi Xavier - 18223064

Program Studi Sistem dan Teknologi Informasi

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung, Jalan Ganesha 10 Bandung

E-mail: evanderruizhi@gmail.com , 18223064@std.stei.itb.ac.id

Abstract— The rapid integration of Large Language Models (LLMs) in the digital banking sector introduces many vulnerabilities regarding customer data privacy and regulatory compliance, particularly during data transit and storage. This paper proposes an application-level cryptographic middleware architecture to mitigate these risks at the application layer. The framework implements a Smart Masking Engine to sanitize sensitive data parameters from the chat prompt attributes to prevent personal data leakage and exploitation. To secure chat logs with the chatbot, the middleware utilizes an isolated Searchable Symmetric Encryption framework driven by HMAC-SHA256 and HKDF key derivation to achieve isolated index separation. Furthermore, data integrity within the storage layer is ensured using AES-256 GCM encryption to the contextual metadata. Experiments and evaluations demonstrate that the proposed architecture successfully prevents identity leakage and protects against unauthorized database tampering. This proposed application-level middleware gives a minimal processing overhead of only 1.31 milliseconds, costing only 0.03% of the total system execution latency.

Keywords—searchable encryption; data masking; blind index; HKDF; AEAD; digital banking; audit

I. INTRODUCTION

The financial field is now defined with the advanced integration of Artificial Intelligence (AI) to drive automation and customer engagement. On the digital banking sector, this technological evolution has rapidly shifted toward core consumer facing interactions through the deployment of Large Language Models (LLMs) that are deployed as conversational, problem-solving assistants. These advanced chatbots helps customers by navigates the usage of application's features, provides guidance of transactional related activities, and deliver customer service tasks faster. Driven by the urgent demand for financial services and digital transformation, many digital banking vendors have considered these generative AI engines as critical components of their operations.

Despite the high autonomy showed by generative AI engines, their integration into financial operations cannot rely entirely on machine execution. According to a framework established by Yablonsky (2021) regarding the levels of AI-based automation, generative AI engines are mostly operated

within the “machine-led, human-governed” level [1]. Under this level of AI maturity, the core task execution is predominantly managed by the AI engine with human oversight on decision making based on their defined governance framework. In a digital banking sector, compliance auditors and human supervisors must actively monitor the chat logs from the chatbot. This intervention is critical to audit the performance of the chatbot, ensure its behavioral alignment with organization's guidelines, and execute real-world actions for customer service tasks that can be done automatically (such as processing customer complaint).

However, this interdependent architecture introduces a severe vulnerability at the intersection of data privacy and regulatory compliance. In practical scenarios, banking customers frequently engage with LLM interfaces with a low awareness of security. They might unwittingly input highly sensitive and restricted data parameters such as Personal Identification Number (PIN), credit card credentials, Card Verification Values (CVV), and national identification numbers directly into the chatbot prompts without knowing the potential threats. Consequently, these sensitive credentials become permanently ingested into the historic chat log database.

This exposure creates a vulnerability on the system that can inflict financial loss to the customers and ruin the organization's image. If the persistence layer holding these unstructured chat logs is exploited by unauthorized actors, it inevitably triggers catastrophic data breaches that leads to financial fraud and identity theft. Traditional cryptographic defenses that rely strictly on standard storage-level encryption fail to solve this problem comprehensively because human auditors legally require seamless access to the chat logs to perform their monitoring duties. Traditional systems are forced to decrypt entire data blocks during this monitoring process to ensure ease of use. The decryption of chat logs exposes sensitive customer credentials to the auditor and creates significant risks of insider threats by violating privacy regulations like GDPR [2].

To bridge this operational gap, financial platforms require transformation in security protocols that decouples data utility from data visibility. This research addresses this problem by

introducing a secure architecture for privacy-preserving audit of banking LLM chat logs using application-level searchable encryption. This solution enforces a zero-trust storage model by migrating cryptographic controls upward to the application middleware. It sanitizes explicit customer data through smart masking within the chat prompt and utilizes Searchable Symmetric Encryption (SSE) alongside Authenticated Encryption with Associated Data (AEAD) for the auditing process. This ensures that compliance auditors can query, filter, and monitor chat logs using blind tokens without gaining access to the underlying plaintext sensitive parameters, thereby establishing an unbreachable defense for digital banking infrastructures.

II. LITERATURE REVIEW AND RELATED WORKS

A. AI Governance and Maturity Framework

The integration of LLM within digital banking platforms requires a good governance model to mitigate the risks of unregulated data generation and AI hallucinations. According to Yablonsky's enterprise maturity framework, advanced conversational AI deployment must operate at the "machine-led, human-governed" level [1]. Under this framework, the AI system autonomously performs response generation and decision making, while human compliance auditors retain oversight and governing authority. Human auditors must continuously evaluate chatbot behaviors to ensure strict alignment with financial regulations and organizational compliance.

B. Application-Level Cryptography

Traditional enterprise database architectures rely heavily on storage-level security mechanisms to secure the stored information. However, IBM highlights that conventional deployments inherently place runtime and database administrators inside the system's trust boundary [3]. This method introduces many vulnerabilities such as security breaches, credential theft, and misuse of administrative privilege. To mitigate these risks, organizations must adopt application-level encryption which shifts cryptographic controls into the application runtime middleware. This approach ensures sensitive data payloads are encrypted in memory before being stored, making the database layer entirely blind to plaintext data and isolating sensitive records from internal infrastructure components.

C. Searchable Encryption (SE)

Standard encryption primitives transform plaintexts into ciphertexts which makes database query operations impossible without full data decryption. To allow compliance auditors monitor logs without decrypting the entire database, Amorim and Costa demonstrate that Searchable Encryption (SE) can be utilized to enable keyword searches directly over encrypted data domains [4]. Within this domain, an index-based Searchable Symmetric Encryption (SSE) architecture allows the system to generate a distinct searchable ciphertext (SC) by cryptographically linking the encrypted message (M') with a secure keyword index (I). Authorized auditors can execute isolated lookups by submitting a trapdoor or search token (T_Q)

derived from a query (Q). This allows the semi-trusted server to apply the search token to identify and retrieve the matched encrypted data segments without decrypting or gaining visibility into the underlying plaintext variables.

D. Authenticated Encryption with Associated Data (AEAD)

While integrated Authenticated Encryption (AE) schemes efficiently guarantee both data confidentiality and authenticity for structured payloads, they lack the mechanism to bind non-encrypted contextual metadata to the ciphertext. To address this limitation, Rogaway formalizes Authenticated Encryption with Associated Data (AEAD) to securely transmit messages in which specific parts remain clear for routing or parsing purposes while the entire data string is cryptographically authenticated [5]. AEAD framework ensures that any associated data (H) such as a network packet header is tightly bound to the encrypted message core (C) and its authenticity tag (T). This guarantees the failure of unauthorized attempts to manipulate the associated data, swap data rows, or alter the ciphertext layout during transmission from the validation process at the receiver's end. With that, the information from the associated data will preserve its integrity and become unforgeable.

E. Related Works

The optimization of privacy-preserving systems using searchable encryption frameworks has been widely researched in previous studies. A recent study conducted by Ma, Jia, Du, Ha, and Li introduced Dynamic Searchable Symmetric Encryption based on Anonymization and Differential Privacy (DADP) which was specifically used to defend cloud environments against query recovery attacks [6]. DADP operates by synthesizing original search indexes into randomized and irreversible fake indexes that utilize the anonymization hash technology. This method effectively prevents adversaries from recovering the underlying index structures. DADP also integrates differential privacy mechanics to process composite indexes and protect keyword information from inference based on query results. By offering this double layer privacy preservation, the DADP scheme achieves type-I backward privacy that can withstand seven distinct types of query recovery attacks and enhances query efficiency by 10 to 100 times compared to existing traditional solutions.

While the DADP framework effectively mitigates query recovery attacks in cloud databases, its core mechanisms have specific trade-offs that limit its applicability to generative AI infrastructures. The integration of differential privacy parameters relies on injecting noise to mask keyword frequencies which is unacceptable for digital banking compliance audits where absolute chat log precision is required. Additionally, their anonymization hashing is designed for structured and predefined index catalogs which doesn't fit for unstructured and dynamic conversational records generated by LLMs. This research bridges that specific gap by combining an exact smart masking engine on the prompt chat with deterministic AEAD context binding. This proposed framework guarantees uncompromised audit trail precision, prevents data or keyword frequency leakage from multiple user

logs, and provides resistance against tampering from unauthorized access.

III. METHODOLOGY

A. System Architecture

This architecture positions application-level security middleware between the Client Layer and the External Layer (LLM API). The system isolates the storage environment by establishing an explicit cryptographic trust boundary within the Flask execution runtime memory. The flow begins at the Client Layer where the Banking App User Interface captures prompt requests from the user. The prompt payload is then dispatched to the Flask Security Middleware Engine which orchestrates a pipeline of operations before any external transmission occurs.

First, the Prompt Interceptor captures the inbound string to prevent direct passthrough. The raw text is then routed to the Smart Masking Engine to mask sensitive data parameters. The external LLM API is kept entirely outside the middleware engine, it only interacts by receiving fully anonymized tokens and returning the responses back to the Prompt Interceptor. Once the Prompt Interceptor sends the responses to the client, the middleware initiates its persistent log isolation sequence. The Cryptographic Key Manager leverages HKDF parameters to derive the Blind Index Key and AES-256 GCM Key. The Blind Index Key is then transmitted to the Blind Index Generator to produce search index tags (I) while the AES-256 GCM Key is transmitted to the AEAD Encryptor/Decryptor engine to compile the database payload by appending the ciphertext (M') with a 16-byte authentication tag (T). The Audit Log Database in the Storage Layer remains completely blind to the context and stores only the Encrypted Chat Logs Table that contains blind indexes and authenticated cryptographic structures. The flow of this process is provided by the diagram in Figure 1 below:

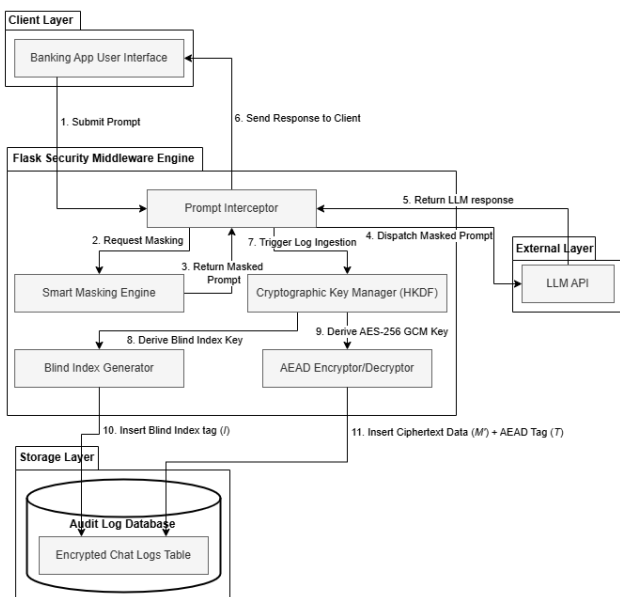


Fig. 1. System Architecture Diagram

B. System Sequence and Lifecycle

The system lifecycle within the proposed middleware has two distinct phases which are the Log Ingestion and Encryption Phase and the Audit Log Decryption Phase.

1) *Log Ingestion and Encryption Phase:* This phase is initiated when the Client submits a chat prompt containing sensitive data parameters such as Personal Identification Numbers (PIN). The Context & Interceptor immediately captures this payload and routes it internally to the Smart Masking Engine. This engine executes Regular Expression (regex) scanning routines to mask the sensitive data parameters with non-reversible tokens to ensure that the external LLM API only receives and processes anonymized text strings. Once the sanitized response is returned by the LLM to the user interface, the middleware triggers a background log ingestion routine. The Key Derivation (HKDF) module extracts the application's master key alongside the unique Username to dynamically derive two operational keys: K_{enc} and K_{idx} . The AEAD and Index Engine ingests these operational keys to compile the secure log envelope. The engine then computes Blind Index (I) by passing the target keyword through an HMAC-SHA256 function using K_{idx} . Concurrently, the chat log (P) is being encrypted using AEAD with the AES-256-GCM mode of operation. The engine encrypts the payload using K_{enc} and a non-repeating Initialization Vector (IV) to produce the ciphertext (C). The Username and Timestamp are injected directly into the cryptographic loop as Additional Authenticated Data (AAD), producing a 16-byte authentication tag (T). The engine then executes a structured database injection, inserting the components (I, C, T, AAD) into the persistent storage layer. The Sequence for Log Ingestion and Encryption Phase is illustrated by the diagram in Figure 2 below:

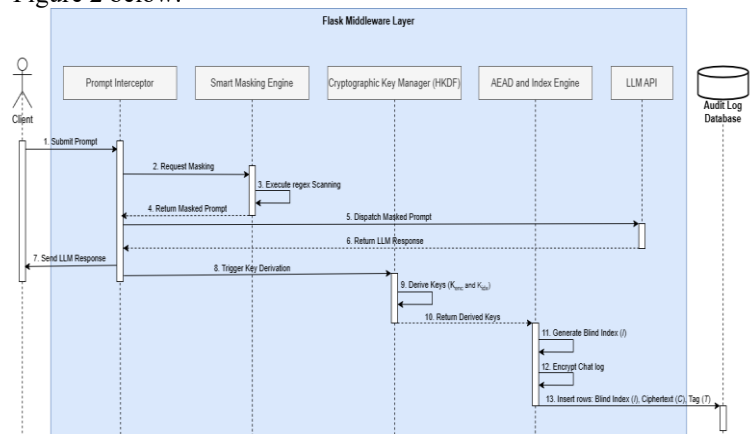


Fig. 2. Sequence Diagram of Log Ingestion and Encryption Phase

2) *Audit Log Decryption Phase:* the initiation of this phase begins when an authorized Compliance Auditor attempts to query the chat logs using a keyword string (Q). The AEAD and Index Engine intercepts this lookup request and invokes the HKDF framework to re-derive the user specific K_{idx} . The

engine then computes a cryptographic trapdoor search token (T_Q) using the formula below:

$$T_Q = \text{HMAC}_{\text{SHA256}}(K_{idx}, Q) \quad (1)$$

The middleware then issues a direct *SELECT* query against the database where the blind index column matches the trapdoor token. Upon retrieving the matched encrypted data rows, the middleware evaluates the authentication tag (T) against the accompanying *AAD* metadata context. If the integrity validation succeeds, the ciphertext (C) is decrypted in memory using K_{enc} and the final plaintext log payload is securely rendered to the auditor’s interface. The Sequence for Audit Log Decryption Phase is illustrated by the diagram in Figure 3 below:

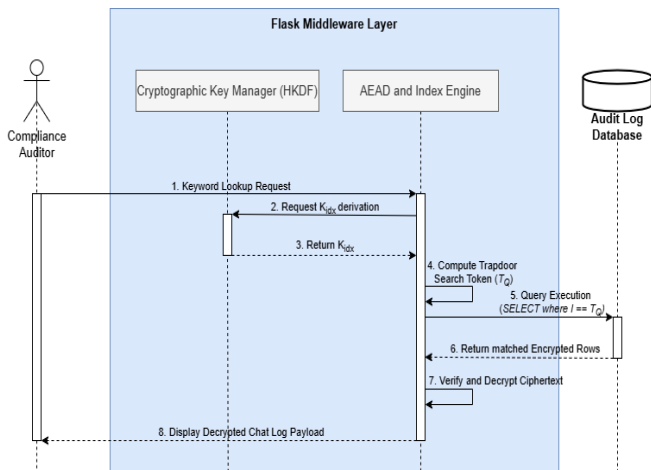


Fig. 3. Sequence Diagram of Audit Log Decryption Phase

IV. RESULT AND DISCUSSION

A. System Functional Evaluation

1) *Smart Masking Evaluation:* The evaluation of the smart masking process was initiated by injecting a payload containing sensitive data parameters like PIN or National ID into the chat prompt. The primary objective of this test was to verify whether the middleware successfully masks plaintext sensitive attributes before being submitted to the chatbot. Figure 4 below shows the simulation of this test:

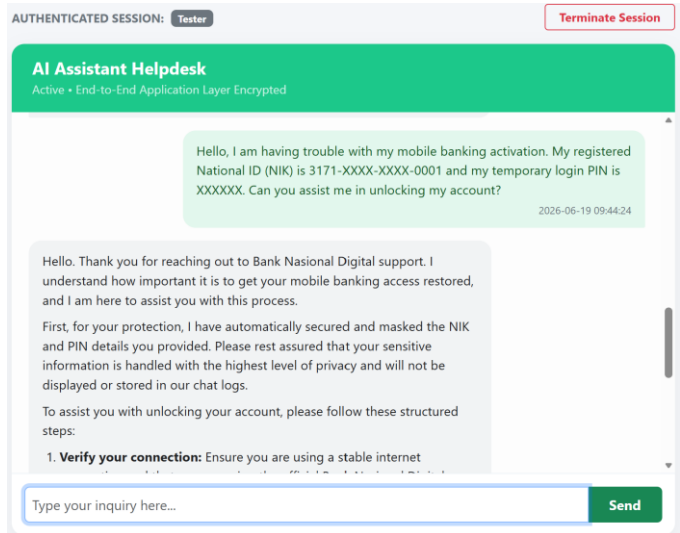


Fig. 4. Smart Masking Result on the Chat Prompt

The results shown in Figure 4 confirmed that the Smart Masking Engine successfully masks the sensitive parameters in runtime memory. The evaluation of regex patterns allowed the middleware to isolate the specific sensitive identity and credential substrings without disrupting the surrounding sentence structure.

2) *Searchable Encryption Lookup Verification:* The operational capacity of the SSE framework was validated by simulating a compliance audit lookup routine. For Example, an authorized auditor initiated a keyword search to retrieve chat logs containing the string "transaction" demonstrated in Figure 5 below:

Search Query Metric Results for: "transaction"

LOG ID	CUSTOMER ID	TIMESTAMP	INBOUND PROMPT	OUTBOUND RESPONSE
3	tester	2026-06-19 10:12:57	My last transaction failed!	Hello. Thank you for reaching out to Bank Nasional Digital support. I understand how concerning it is when a transaction does not go through as expected, and I am here to assist you in resolving this matter. Please rest assured that I have taken the necessary steps to secure your account information. For your privacy and security, any sensitive data provided in our conversation—such as NIK, card numbers, or PINs—has been automatically masked and secured by our system. To help me investigate this further, could you please provide the following details? 1. The approximate time of the transaction. 2. The amount of the failed transaction. 3. Any error message or code you may have received on your screen. Please do not share your full card number or PIN. Once you provide these details, I will be able to look into the transaction status for you. In the meantime, I have generated a secure Internal Ticket ID: BND-99284-X for your issue. You can quote this ticket number if you need to escalate this matter to our human support team for an instant resolution. I am ready to help—please provide the details above whenever you are ready.
1	evendad798	2026-06-19 10:29:57	My last transaction failed!	I am very sorry to hear that you are experiencing issues with your recent transaction. I understand how stressful it can be when your funds are involved, and I am here to assist you in resolving this as quickly as possible. Please rest assured that I have generated a secure Internal Ticket ID: BND-88294-X for your case. You can quote this number if you need to escalate this to our human support team for an instant resolution. To help me investigate this for you, could you please provide the following details? 1. The approximate date and time of the transaction. 2. The amount that was intended to be transferred. 3. Any error message you received on your screen. SECURITY NOTICE: Please be advised that if you provide any sensitive information such as your NIK, full Card Number, or PIN, my system will automatically mask and secure those credentials to ensure your privacy and account safety. Please do not share your PIN with anyone, including our staff. Once you provide the transaction details, I will be able to check the status of your funds and guide you through the next steps. I am here to help you get this sorted out.

Fig. 5. SSE Lookup Result for the Keyword "transaction"

The lookup process from Figure 5 shows exact precision during data retrieval. The AEAD and Index Engine constructed the correct trapdoor token (T_Q) corresponding to the auditor’s keyword input. The database successfully matched the recomputed tokens without executing a full table scanning routine. Because the lookup matches the index tags without revealing the plaintext chat logs, the system satisfies the requirements of a zero-knowledge search model.

3) *Index Isolation Analysis*: A test simulating multi-user data collision was executed to investigate the middleware's resilience against index frequency and pattern deduction attacks. Two distinct users were configured to ingest the same inbound prompt (log with ID 1 and ID 3) as shown in Figure 5 from the previous test. Figure 6 below shows the chat logs database records for the search index:

id	username	timestamp	search_index
1	evander2703	2026-06-19 10:29:57	4c90fc725c0c926e455e542dc06f06cb36f3...
2	tester	2026-06-19 10:32:29	E89ee542f36ecc46af2c01f7dc9153349af4...
3	tester	2026-06-19 10:32:57	e406d41c03228c95c919512452ae44d5c1de...

Fig. 6. Search Index Records for All Chat Logs in the Database

The query execution from Figure 6 shows that the log with ID 1 and ID 3 have different encrypted search index. The different index values prove the expected functionality of the HKDF) module. By extracting the application's master key alongside the distinct Username as an isolated salt variable, the derived index keys remain separated ($K_{idx_A} \neq K_{idx_B}$). This makes the computation of T_Q for the same keyword produce entirely different outputs and successfully breaking the index frequency correlation that commonly compromises standard SSE frameworks.

4) *Storage Security Evaluation*: To evaluate the system's security capabilities against data exposure within the storage layer, an insider breach simulation was executed. This test utilizes a dedicated endpoint route (`/hacker/view`) that directly fetch the raw database dump, simulating a successful SQL injection or administrator privilege leakage. The result of this simulation is shown on Figure 7 below

ID	USERNAME	TIMESTAMP	ENCRYPTED_PROMPT	NONCE_PROMPT	TAG_PROMPT	ENCRYPTED_RESPONSE	NONCE_RESPONSE	TAG_RESPONSE	SEARCH_INDEX
1	evander2703	2026-06-19 10:29:57
2	tester	2026-06-19 10:32:29
3	tester	2026-06-19 10:32:57

Fig. 7. Simulation of Database Leakage

As demonstrated in Figure 7, if an unauthorized user fetches the database dump, all columns besides ID, Username, and Timestamp remain entirely unreadable. Those columns only show the hex ciphertext values (C), making them unusable for exploitation. To further test the system's active resistance against tampering, another injection attack by swapping the Username rows is executed. Figure 8 below shows the chatroom of the affected user during this attack:

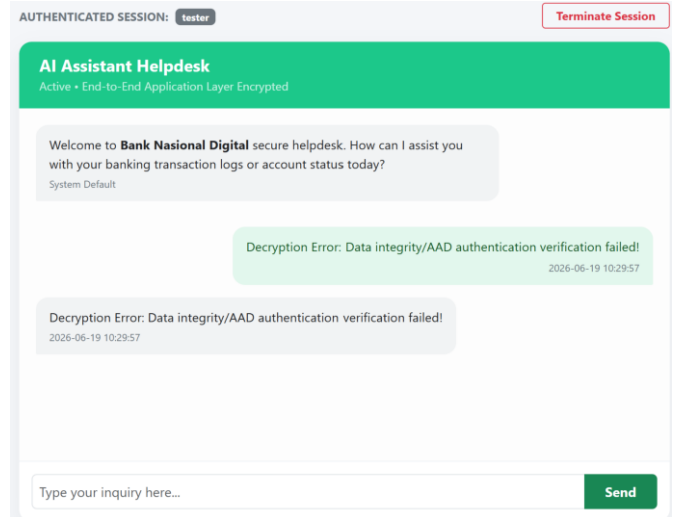


Fig. 8. Simulation of Context Row Swapping

The simulation in Figure 8 validates the robust cryptographic binding provided by the authenticated encryption scheme. When the middleware attempts to process the altered rows, the verification loop re-evaluates the 16-byte authentication tag (T) alongside the contextual metadata injected as AAD . Because any modification invalidates the authentication equation, the system immediately rejects the decryption of the data. This mechanism guarantees that even if a hacker successfully gains full read access to the database dump, confidentiality of the database records remains intact. And if a hacker tries to tamper with the database record's values, they won't gain any plaintext information from the user's chat logs.

B. Performance and Overhead Evaluation

To measure the operational cost of applying the middleware, the runtime execution of the chatbot was recorded. Table I below shows the runtime execution from the previous test in Figure 4:

TABLE I. CHATBOT RUNTIME EXECUTION

Performance Metrics	Execution Latency (ms)
LLM Connection	3,758.22
Middleware Processing Cost	1.31
Total Latency	3,759.43

The chatbot runtime execution data from Table I shows that the application-level middleware gives a minimal computational cost during execution. The entire process of the middleware layer only takes about 1.31 milliseconds. Compared to the latency of the LLM itself (3,758.22 ms), the middleware's overhead costs only 0.03% of the total runtime. This confirms that the proposed framework provides high information security without noticeable latency degradation. The proposed framework proves its viability for integration into digital banking infrastructures.

V. CONCLUSION

This research successfully implements a secure architecture for an application-level middleware layer for securing LLM chat logs in digital banking environments. Based on the evaluations from Chapter IV, several conclusions from the system are summarized:

1) The integration of the Smart Masking Engine effectively masks sensitive data parameters from the user (such as PIN or National ID records) before being processed by the LLM. This mechanism neutralizes the risk of data leakage to unauthorized users without disrupting the flow of customer support interactions.

2) The middleware achieves isolated index separation by using HMAC-SHA256 blind indexing coupled with HKDF-SHA256 key derivation that binds the application's master key to unique usernames. This approach successfully breaks index frequency and pattern correlation attacks by computing a different trapdoor search token for the same keyword string on different users. The AEAD and Index Engine are also proven to reconstruct the trapdoor search token for all matching keywords from every user.

3) The implementation of the encryption loop with AES-256 GCM mode ensures the storage-layer security. The system successfully protects the chat log database from direct database dump and context row swapping attacks by automatically throwing validation errors and blocking data decryption upon detecting these attempted attacks.

4) The performance and overhead evaluation confirms that the entire process of the middleware layer gives a minimal processing overhead of only 1.31 milliseconds, costing only 0.03% of the total system execution latency. This proves that the proposed framework delivers high information security without significant latency degradation, making it suitable for digital banking infrastructures.

In summary, the developed application-layer middleware serves as a highly viable and efficient architecture that bridges the gap between utilizing generative AI and maintaining information security in the digital banking sector.

VI. FUTURE WORKS

Based on the limitations of this research, several improvements are proposed for future development:

1) Using dynamic Named Entity Recognition (NER) to replace the current regex-based tokenization engine.

2) Transition the database architecture to a permissioned blockchain network to achieve distributed mutability.

3) Enhance the current SSE framework beyond the current single-exact-keyword matching model.

SOURCE CODE AT GITHUB REPOSITORY

[evanderruizhix/sse-banking-llm-audit](https://github.com/evanderruizhix/sse-banking-llm-audit)

ACKNOWLEDGMENT

The author would like to express gratitude to Dr. Ir. Rinaldi Munir, M.T. for his dedication and patience in lecturing the II4021 Cryptography course. The author would also like to appreciate fellow friends in Informatics and Information System and Technology, Class of 2023, for their support and motivation during this research.

REFERENCES

- [1] S. Yablonsky, "AI-driven platform enterprise maturity: from human led to machine governed," *Kybernetes*, vol. 50, no. 10, pp. 2753-2789, Oct. 2021.
- [2] P. Voigt and A. von dem Bussche, *The EU General Data Protection Regulation (GDPR): A Practical Guide*. Cham, Switzerland: Springer, 2017.
- [3] IBM, "Enhance your data security posture with a no-code approach to application-level encryption," *IBM Tech Docs*, 2023. [Online]. Available: <https://share.google/n69nb4QvLZDw5xzi4>
- [4] I. Amorim and I. Costa, "Leveraging Searchable Encryption through Homomorphic Encryption: A Comprehensive Analysis," *Mathematics*, vol. 11, no. 13, p. 2948, Jul. 2023.
- [5] P. Rogaway, "Authenticated-encryption with associated-data," in *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS '02)*, Washington, DC, USA, 2002, pp. 98-107.
- [6] C. Ma, C. Jia, R. Du, G. Ha, and M. Li, "Privacy-preserving searchable encryption based on anonymization and differential privacy," in *Proceedings of the 2024 IEEE International Conference on Web Services (ICWS)*, 2024, pp. 371-382.

STATEMENT

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 19 Juni 2026



Ahmad Evander Ruizhi Xavier 18223064