JIKO (Jurnal Informatika dan Komputer) Accredited KEMDIKTISAINTEK, No.0173/C3/DT.05.00/2025
Vol. 9, No. 1, April 2026, pp. 29-37 p-ISSN: 2614-8897
DOI: 10.33387/jiko.v9i1.10884 e-ISSN: 2656-1948

FIELD LEVEL ENCRYPTION IMPLEMENTATION USING AES-256 FOR SECURE ACADEMIC INFORMATION SYSTEMS

Adi Fajaryanto Cobantoro[1*], Elok Putri Nimasari[2], Sincan Maulana[3], Mohd Zuber[4], Shaikh Ameer Basha[5]

[1,2,3] Informatics Engineering, Engineering Faculty, Universitas Muhammadiyah Ponorogo, Ponorogo, 63122, Indonesia
[4]Department of Teacher Training & Non Formal Education IASE, Faculty of Education, Jamia Millia Islamia (A Central University), New Delhi, 110025, India
[5]Department of Mathematics, Basic Science Faculty, Bearys Institute of Technology, Mangalore, 574199, India
Email: [*1]adifajaryanto@umpo.ac.id, [2]nimasari@umpo.ac.id, [3]sincanmaulanaa@gmail.com, [4]zuber15saifi@gmail.com, [5]shaikhameerbasha@gmail.com

(Received: 21 October 2025, Revised: 26 January 2026, Accepted: 18 February 2026)

Abstract

Data protection in higher education remains a pressing concern as student records and program registration data are frequent targets of cyber incidents. This paper presents the design and implementation of AES-256 to secure the registration feature within an academic information system (AIS). Specifically, this study delivers three main contributions: a modular cryptographic implementation at the controller level, a granular field-level encryption policy for sensitive attributes, and a validated security mechanism. We integrated a cryptographic module into a Fastify (Node.js) backend and PostgreSQL datastore. The results demonstrate the fulfillment of these contributions: first, the modular implementation effectively isolates encryption logic from the database layer; second, the field-level policy successfully secures sensitive PII while maintaining 100% query efficiency for non-sensitive data; and third, the security mechanism was validated through 17 white-box scenarios and dual-layer API testing. These results confirmed 100% functional correctness in encryption/decryption cycles and robust handling of invalid data inputs. The study contributes a practically deployable pattern for field level encryption in university information systems.

Keywords: *AES-256, Academic Information System, White-Box Testing, API Testing*

**Corresponding Author: Adi Fajaryanto Cobantoro*

1. INTRODUCTION

Existing data protection mechanisms in higher education often rely heavily on Transport Layer Security (TLS) for data in transit [1, 2] and database-level encryption methods like Transparent Data Encryption (TDE) for data at rest [3,4]. While these standards are necessary, they are insufficient against internal threats. TDE, for instance, encrypts the physical storage but transparently decrypts data for any user with valid database credentials, including Database Administrators (DBAs)[5, 6]. This architecture creates a critical vulnerability, a privileged insider or an attacker who compromises a DBA account can harvest sensitive student PII in cleartext [7].

While standard mechanisms like TLS and TDE provide essential baseline security, they fail to address the fundamental 'trust' issue at the database layer. Existing solutions primarily focus on peripheral security rather than data-centric protection within the application logic . This study addresses this gap by implementing Field-Level Encryption (FLE) using AES-256 at the application controller layer. By integrating encryption directly into the business logic, data remains encrypted even within the database engine's memory, ensuring that only the application not the database layer holds the power to decrypt sensitive fields.

Furthermore, relying solely on disk-level encryption fails to provide granular control over which specific attributes remain confidential during application processing [8]. This study addresses this

gap by implementing Field-Level Encryption (FLE) using AES-256 at the application controller layer. By integrating encryption directly into the business logic, data remains encrypted even within the database engine's memory, ensuring that only the application not the database layer holds the power to decrypt sensitive fields.

In addition, the implementation of AES-256 at the application layer ensures end to end protection between client and server [9-11]. By integrating encryption functions directly into business logic and controller modules, data remain encrypted during transmission, processing, and storage. This approach mitigates risks associated with misconfigured databases or unauthorized data exports. Furthermore, combining cryptography with secure coding practices and periodic penetration testing strengthens overall system resilience [12,13]. In essence, data encryption becomes part of a larger cybersecurity ecosystem encompassing monitoring, risk management, and continuous improvement.

From a research perspective, the exploration of AES-256 in academic systems contributes to the growing body of literature on applied cryptography in information management [14]. It offers empirical insights into how theoretical encryption models translate into real-world implementations, especially within the context of educational data governance [15]. The research not only addresses technical encryption procedures but also raises awareness of institutional responsibilities in data protection, aligning technological innovation with ethical and regulatory dimensions.

Currently, many Academic Information Systems (AIS) still rely on basic disk-level encryption or, in some cases, store sensitive attributes in plaintext [16-18]. This traditional approach leaves data vulnerable to specific threats such as SQL injection attacks and internal privilege abuse, where database administrators can view sensitive student PII without authorization [19,20]. Furthermore, the lack of granular encryption at the application level means that if the database layer is compromised, the entire dataset is exposed [21]. Consequently, academic institutions must go beyond conventional access control and authentication mechanisms, embracing field-level encryption technologies that ensure confidentiality even when other layers are compromised.

This study implements field-level encryption using AES-256 within an Academic Information System (AIS) registration module. The approach ensures robust data confidentiality for both stored information and real-time transmission between the client and server. The contributions of this paper are as follows: (1) a modular AES-256 cryptographic implementation at the controller and data-access layers, (2) an effective field-level encryption policy for sensitive attributes, and (3) a comprehensive validation process using white-box and API testing methods.

2. RESEARCH METHOD

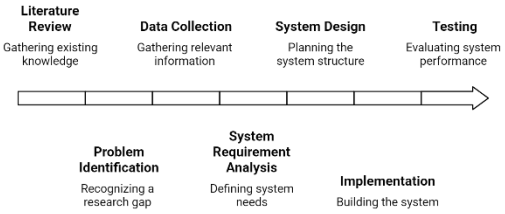

Figure 1. Research Method

This research followed a systematic development flow specifically tailored for secure software engineering as seen at Figure 1. The stages included: (1) Threat Modeling and Requirement Analysis to identify sensitive PII attributes; (2) Cryptographic System Design to define the AES-256 operational modes and key management flow; (3) Modular Implementation within the Node.js environment; and (4) Dual-Layer Validation encompassing white-box unit testing and black-box API verification. This framework ensures that the encryption mechanism is not only theoretically sound but also practically verifiable within the AIS architecture.

2.1 Literature Review

Literature Review served as the foundation to explore prior studies related to encryption algorithms, database security, and academic information systems. Through the review, theoretical and empirical insights were gathered to define the scope and determine suitable technologies, including AES-256 and field level encryption approaches.

2.2 Problem Identification

Problem Identification involved recognizing existing vulnerabilities within academic data management particularly risks related to unauthorized access, weak database encryption, and unprotected data transmission. The analysis highlighted the need for an efficient encryption mechanism to secure sensitive information at the field level.

2.3 Data Collection

Data Collection was conducted through interviews and system documentation reviews to gather current workflow and database structure information from institutional information systems. These findings guided the design of a model that could be realistically implemented within existing infrastructure.

2.4 System Requirements Analysis

System Requirements Analysis focused on identifying sensitive data elements requiring protection, defining performance expectations, and specifying technical constraints for encryption integration. Database fields such as student name, description, quota, and registration period were classified as sensitive and thus targeted for encryption. This step also outlined non-functional requirements such as system scalability, auditability, and user

accessibility. In addition, the functional requirements were detailed based on the core features of the registration module, as shown in Table 1.

Table 1. Features in the Academic Registration Module

No.	Feature	Description
1	Login	Both administrators and students can log in to the system to access the registration feature.
2	Input registration, approval, and grade conversion forms	Students fill in the forms required for registration in the academic program.
3	Input payment proof	Students upload payment proof to the platform after completing the transaction.
4	Student list	Administrators can view the list of students who have registered in the program.
5	Verification of forms and payment proofs	Administrators verify the forms and payment proofs submitted by the students.

2.5 System Design

System Design involved the conceptualization of the encryption framework and overall system architecture. The system was designed with a Next.js frontend, a Fastify (Node.js) backend, and PostgreSQL database, ensuring support for modern web deployment and secure data processing. Flowcharts, use-case diagrams, data flow diagrams (DFDs), and entity relationship diagrams (ERDs) were developed to represent encryption boundaries, data movement, and module interaction. Figures 2 illustrates the flowcharts used in admin.

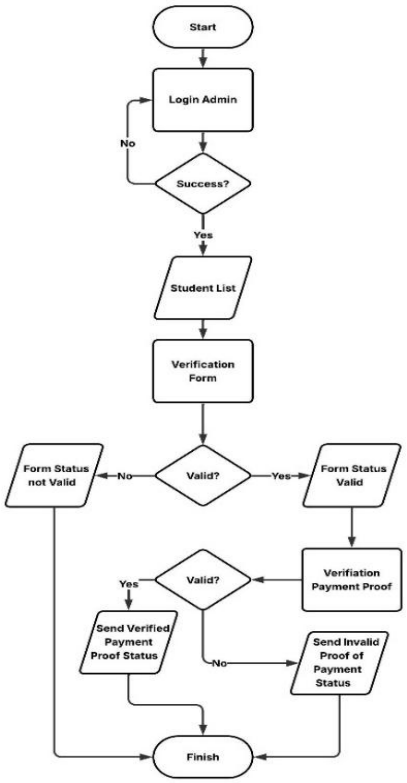

Figure 2. Flowchart Admin

Figure 3, demonstrates the sequential logic of AES-256 encryption: beginning with input data, key generation, conditional chunking, iterative encryption across 14 rounds, and concluding with the generation of ciphertext. It highlights the role of chunking for oversized input, the repeated encryption cycle, and the production of secure encrypted output.

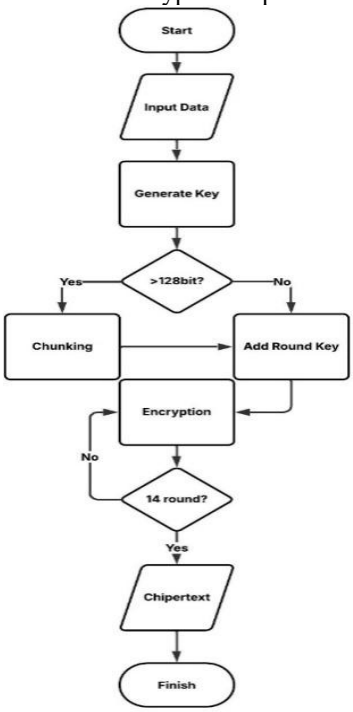

Figure 3. AES-256 Flowchart

2.6 Implementation

At this stage, the Advanced Encryption Standard (AES-256) algorithm was implemented to secure student data within the academic information system, specifically in the registration feature for the Kampus Merdeka program. The example below outlines the step-by-step implementation of AES encryption for a payment proof file in the registration module.

Step 1, define the plaintext, initial key, and initialization vector.

- Plaintext (original text) = pembayaran.jpg
- Initial Key = informatika
- Initialization Vector (IV) = teknik

Step 2, load the plaintext into a processable format. The original text is converted into a form that can be processed by the AES algorithm by organizing the data into a 4×4 state matrix as seen on Figure 4, corresponding to a 128-bit block size. If the plaintext does not reach 16 bytes, the PKCS7 padding mode is applied to extend its length to 16 bytes.

p	a	a	p
e	y	n	g
m	a	.	06
b	r	j	06

Figure 4. 4x4 matrix

Step 3, generate a 256-bit key. A 256-bit encryption key is generated using the SHA-256 function applied to the initial key.

Initial Key: informatika
SHA-256:
da17c76b826f4b71e9f0b6c92c38318e5c8e68275062
37f6bbdc605157c051fa

Step 4, perform Key Expansion. The key expansion process uses word data units and applies rotWord and subWord operations to generate multiple round keys. For AES-256, Nk = 8, and 14 rounds are required.

Table 2. Key Expansion

i	*temp*	*rotWord*	*subWord*	*Rcon*	*w[i-Nk]*	*w[i]*
8	57c05 1fa	c051fa 57	69af2f bb	01000 000	da17c 76b	0321b f86
9	0321b f86	-	-	-	826f4 b71	806e4 b17
10	806e4 b17	-	-	-	e9f0b6 c9	69ee8 1de
11	69ee8 1de	-	-	-	2c383 18e	45d8b 050
12	45d8b 050	d8b05 045	3597c 0ea	02000 000	5c8e6 827	6897c 0ea
13	6897c 0ea	-	-	-	50623 7f6	10251 71c
14	10251 71c	-	-	-	bbdc6 051	abf970 4d
15	abf970 4d	-	-	-	57c05 1fa	fc3106 2f
16	fc3106 2f	31062 ffc	6eda8 2d0	04000 000	0321b f86	6b8e9 03d
17	6b8e9 03d	-	-	-	806e4 b17	4b43e 52f
18	4b43e 52f	-	-	-	69ee8 1de	a4a64 472
19	a4a64 472	-	-	-	45d8b 050	d9851 fbb
20	d9851 fbb	851fb bd9	3597c 0ea	08000 000	6897c 0ea	c0c4a 55a
21	c0c4a 55a	-	-	-	10251 71c	8b49f 473
22	8b49f 473	-	-	-	abf970 4d	abf970 4d
23	2e8ac 1cc	-	-	-	fc3106 2f	fc3106 2f

The example shown in Table 2 produces the following round keys:

Initial Key: da17c76b 826f4b71 e9f0b6c9 2c38318e

Round Key [0]: 5c8e6827 506237f6 bbdc6051 57c051fa
Round Key [1]: 0321bf86 806e4b17 69ee81de 45d8b050
Round Key [2]: 6897c0ea 1025171c abf9704d fc31062f
…
Round Key [n]: abed89bf c43593a9 d6bd0758 f36fb227

Step 5, XOR Operation with the Initialization Vector as shown in figure 5. The XOR operation is performed between the initialization vector and the plaintext. The result of this XOR computation is as follows: 0400060C 08124142 515E1E5A 40570000

p	a	a	p	⊕	t	i	0	0	=	04	08	51	40
e	y	n	g		e	k	0	0		00	12	5E	57
m	a	.	0		k	0	0	0		06	51	1E	00
b	r	j	0		n	0	0	0		0C	42	5A	00

Figure 5. XOR vector initialization

Step 6, AddRoundKey Process. In this step, the first-round key is combined with the XOR output using a bitwise XOR operation: 5c8e6827 506237f6 bbdc6051 57c051fa.

F5	53	65	B0	⊕	04	08	51	40	=	C7	DB	54	50
4B	8D	51	29		00	12	5E	57		08	D7	60	1E
A5	C3	35	BF		06	51	1E	00		53	F3	AD	B8
FE	E4	1B	4E		0C	42	5A	00		56	69	B9	7A

Figure 6. Adding RoundKey

The key expansion process is repeated until 14 round keys are generated

2.7 Testing

Testing validated the accuracy, security, and functionality of the system through two complementary approaches: (1) White-box Testing, focusing on internal logic, control flow, and key management processes; and (2) API Testing (Postman), verifying encryption consistency, endpoint security, and correct decryption results in client-server interactions. The successful completion of all 17 test scenarios confirmed that the final implementation maintained data confidentiality, integrity, and system reliability.

3. RESULT AND DISCUSSION

3. 1 Implementation of the Advanced Encryption Standard (AES-256) Algorithm

In this system, data encryption is implemented using the Advanced Encryption Standard (AES-256) algorithm with the Cipher Block Chaining (CBC) mode of operation.

1. Encryption and Decryption Process

The encryption process utilizes the built-in Crypto module of Node.js. The encryption key (key) and initialization vector (IV) are generated from predefined secret environment variables (ENV.SECRET_KEY and ENV.SECRET_IV) using the scryptSync function a secure key derivation algorithm resistant to brute-force attacks [22], [23]. The aes-256-cbc algorithm is employed in combination with the createCipheriv and createDecipheriv methods to perform data encryption and decryption.

The encryption process utilizes the built-in Crypto module of Node.js. The selection of the Advanced Encryption Standard (AES) algorithm with a 256-bit key length is based on its status as a globally adopted industry standard that has been security-tested for years [24], [25].

For its operating mode, this system uses Cipher Block Chaining (CBC). CBC ensures that each ciphertext block is dependent on all previous plaintext blocks, so patterns in the plaintext will not be visible

in the ciphertext. However, it should be noted that modern alternatives such as Galois Counter Mode (GCM) are increasingly recommended for new systems [26]. Furthermore, AES-GCM provides not only confidentiality (like CBC) but also inherent data integrity and authentication through authentication tags (AEAD - Authenticated Encryption with Associated Data) [27]. While CBC remains secure when implemented correctly, especially with the use of a unique and random Initialization Vector (IV) for each encryption and mitigation against padding oracle attacks the choice of CBC in this system was based on the availability of mature implementations and meeting the data confidentiality requirements essential for this academic information system.

The following section presents the encryption function code that demonstrates this implementation.

2. Dynamic Field Level Encryption and Decryption

The flexibility of choosing which fields to encrypt is the core strength of the Application-Level Encryption (ALE) approach [28], [29]. Unlike Transparent Data Encryption (TDE), which often indiscriminately encrypts entire tables or databases, the encryptFields and decryptFields functions allow for selective implementation of security policies.

This approach offers an optimal balance between security and performance. Non-sensitive fields frequently used for search, indexing, or join operations can be left in plaintext, without compromising database performance. This contrasts with whole-object encryption approaches (e.g., encrypting an entire row of data as a single encrypted JSON blob), which would make standard database operations impossible and significantly degrade performance [30].

To support dynamic field-level encryption during data storage in the database, the system employs the encryptFields function. As seen on Figure 7, this function accepts an object containing key–value pairs and encrypts each string value before it is stored in the database, ensuring that sensitive information remains protected at the field level.

```
export const encryptFields = <T extends keyof DB>(
  fields: Record<string, any>
): InsertObject<DB, T> => {
  const encryptedFields: Record<string, any> = {};
  for (const [key, value] of Object.entries(fields)) {
    encryptedFields[key] = value ? encrypt(value.toString()) : value;
  }
  return encryptedFields as InsertObject<DB, T>;
};
```

Figure 7. Dynamic Encryption Function

Meanwhile, for the data retrieval process from the database, the system utilizes the `decryptFields` and `decryptSpecificFields` functions. These functions enable either the decryption of all fields or only specific fields, depending on the operational requirements. The following section presents the `decryptFields` function as seen on Figure 8.

```
export const decryptFields = (
  data: Record<string, any> | Record<string, any>[]
): Record<string, any> | Record<string, any>[] => {
  if (Array.isArray(data)) {
    return data.map((item) => decryptFields(item));
  }

  const decryptedFields: Record<string, any> = {};
  for (const [key, value] of Object.entries(data)) {
    decryptedFields[key] =
      value && typeof value === 'string' && isEncrypted(value)
        ? decrypt(value)
        : value;
  }
  return decryptedFields;
};

export const decryptSpecificFields = (
  data: Record<string, any>,
  fields: string[]
): Record<string, any> => {
  const decryptedFields: Record<string, any> = { ...data };

  for (const field of fields) {
    const value = data[field];
    if (value && typeof value === 'string' && isEncrypted(value)) {
      decryptedFields[field] = decrypt(value);
    }
  }

  return decryptedFields;
};
```

Figure 8. Dynamic Decryption Function

Subsequently, the isEncrypted function is used to verify whether a given value has already been encrypted before attempting to decrypt it. This preventive mechanism helps avoid processing errors and ensures that only valid ciphertext values undergo the decryption procedure.

3. Encrypted Data Results in the Academic Database

The following presents the results of the data encryption process applied to the program table in the academic information system's database. These data represent the output generated by invoking the encryptFields function within the createProgram controller.

To ensure data confidentiality and integrity, two distinct testing categories were applied: True Data Test and False Data Test. First, True Data Test (Authorized Decryption), in this scenario, the system was provided with the correct 256-bit cryptographic key and the original Initialization Vector (IV). The results showed a 100% success rate in plaintext recovery, where the encrypted registration data was accurately reverted to its original form without any character corruption. This confirms the functional correctness of the modular implementation within the Fastify controller.

Second, are False Data Test (Unauthorized/ Tampered Access), tis test aimed to simulate unauthorized access or data corruption. We intentionally applied incorrect keys and tampered with the ciphertext strings. The experimental results demonstrated that the system consistently failed to produce the original plaintext, instead returning cryptographic errors or non-readable randomized strings. This categorical failure in the "False Data Test" is a critical security indicator, proving that without the precise authorized parameters, the sensitive PII remains inaccessible even if the database layer is compromised.

```
================================================
   AES-256-CBC ENCRYPTION MODULE - TEST SUITE
================================================

[INFO] Original Data: "Mahasiswa: Budi Santoso"
[INFO] Stored Ciphertext:
03dafa841df47626a94cf53ab3a6fd07:a926420... (truncated)

------------------------------------------------
TEST CASE A: DECRYPTION WITH VALID KEY
------------------------------------------------
Input Ciphertext : 03dafa841df47626a94cf53ab3a6fd...
Using Key (Hash) : de9ee9adbbb858b3c6b8...
RESULT           : [SUCCESS] => Mahasiswa: Budi Santoso


------------------------------------------------
TEST CASE B: DECRYPTION WITH INVALID KEY
------------------------------------------------
Input Ciphertext : 03dafa841df47626a94cf53ab3a6fd...
Using Key (Hash) : bcf627cfcded8643ed95...
RESULT           : [FAILED] => Decryption Error: Invalid Key or
Corrupted Data
================================================
```

Figure 9. Encrypted Data Stored

Figure 9 demonstrates the robustness of the encryption module through positive and negative testing results. The first section of the image displays the "True Data Test," where the correct decryption key and IV are applied, successfully retrieving the original plaintext "Information Systems". Conversely, the second section illustrates the "False Data Test," where an incorrect key or tampered ciphertext is processed. In this scenario, the system correctly fails to produce readable output (returning garbage data or throwing a decryption error), validating that data integrity and confidentiality are maintained even under erroneous conditions.

Meanwhile, fields such as program_id, created_at, and updated_at are excluded from the encryption process because they do not contain sensitive information. These fields are retained in their original format to support essential database operations, including data tracking (audit trail), record ordering, and backend processing that require unencrypted identifiers or timestamps.

The implementation of field-level encryption using AES-256-CBC in this academic information system demonstrates a pragmatic approach to securing sensitive data. Compared to other solutions, such as TDE or FDE, the adopted Application-Level Encryption (ALE) approach offers several key advantages. First, it provides granular control, allowing developers to selectively encrypt only data that truly requires protection, such as name, description, and quota, while leaving the primary key (program_id) and metadata (created_at) unencrypted to maintain database functionality and performance [31].

Second, by managing keys at the application level, this system creates a clear separation of duties between application administrators and database administrators, thereby reducing the risk of insider threats [32]. While modern operating modes such as AES-GCM offer additional benefits in terms of data authentication, a careful implementation of CBC with proper IV management, as demonstrated, proves adequate and secure for the threat level faced by this system.

3. 2 Comparison with Existing Approaches

To validate the contribution of this study, we compared the proposed Field-Level Encryption (FLE) architecture with standard database encryption methods (TDE) and recent related studies. As shown in Table 3, while previous works like Monica et al. [33] focus on theoretical algorithm performance, and Antonopoulos et al. [31] discuss cloud-native TDE, this study fills the gap by providing a deployable application-layer security model that mitigates insider threats.

Table 3. Comparison of Security Features and Implementation Approaches

Feature / Criteria	Standard Database TDE ([31])	Algorithm Focus ([33])	Proposed Method (AES-256 Field-Level)
Encryption Layer	Database Engine (Kernel Level)	Application / Hybrid Logic	Application Controller (Node.js)
Encryption Scope	Entire Tablespace / File	Question Bank Items	Dynamic Field-Level (Selective Attributes)
Insider Threat Protection	Low (DBA transparently decrypts data)	High	High (Separation of Duties)
Performance Strategy	Hardware Acceleration	Algorithm Speed Analysis	Selective Encryption (Reduces Overhead)
Validation Method	Performance Benchmarking	Encryption Time Metrics	White-box (Logic) & Black-box (API) Testing

3. 3 System Testing

The testing phase validates that the encryption and decryption functionalities within the system operate in accordance with the established objectives. This testing adopts two primary approaches white-box Testing to verify the internal logic of the encryption module and black-box Testing via Postman to ensure functionality at the API level.

1. Unit Testing (white-box)

White-box testing focuses on the unit level to ensure that each function within the cryptographic module (encrypt, decrypt, encryptFields, decryptFields, and isEncrypted) works correctly and can handle various input conditions. A total of 17 test scenarios were designed to cover:

- Functional Validity: Testing encryption and decryption on general strings, empty strings, special characters, and large-sized strings.
- Output Structure: Ensuring that the encryption output format is always consistent (iv:ciphertext).

- Edge Case Handling: Verifying the handling of null, undefined, and invalid input formats to prevent system crashes.
- Object and Array Operations: Testing the utility functions' ability to process encryption and decryption on single objects, arrays of objects, and mixed data (encrypted and non-encrypted).
- Integrity and Negative Testing: Verifying system behavior when creating decryption requests with incorrect keys or modified ciphertext. The tests confirmed that the system does not leak information and handles decryption failures gracefully without exposing stack traces.

All 17 test scenarios passed (status: 17 passed), indicating that the implemented encryption module is robust, stable, and functionally correct according to specifications.

2. API Integration Testing (black-box)

Black-box testing was conducted using Postman to validate the encryption implementation in an end-to-end application workflow. This testing ensures that API endpoints correctly apply encryption to incoming requests and decryption to outgoing responses. Test scenarios cover two main features:

- Authentication Feature: Testing login endpoints for successful and failed scenarios (e.g., wrong password or unregistered account) to ensure appropriate API responses.
- Program Registration Feature (CRUD: Verifying Create, Read, Update, Delete (POST, GET, PUT, DELETE) operations on the registration endpoint. This testing confirms that data sent via the request body is successfully encrypted before being stored in the database, and data retrieved from the database is successfully decrypted before being sent as a response.

All API tests showed “passed” results, validating that the encryption module has been correctly integrated into the application controller and functions as expected in an operational environment.

In software engineering, particularly for security-critical systems, a multi-layered testing strategy is crucial [34]. Unlike a single testing approach, this system adopts a combination of white-box (unit testing) and black-box (API integration testing) methodologies to achieve comprehensive validation coverage. This approach aligns with best practices in secure software development, where the functional correctness of individual components and their behavior within the larger system must be verified separately.

White-box unit testing was chosen to isolate and verify the internal correctness of the cryptographic module. This contrasts with purely black-box testing, which can only validate the final output without guaranteeing that the internal logic has been implemented correctly. By testing each function separately, we can ensure the cryptographic foundation of the system is solid before integration.

On the other hand, black-box API integration testing serves as a crucial complement [35]. Unlike isolated unit testing, this testing validates that the encryption module interacts correctly with other components in the application architecture (e.g., controllers, middleware, and database access). Unit testing might show that the encrypt function works perfectly, but only integration testing can ensure that the function is actually called before data is written to the database. Thus, the combination of these two methods provides higher confidence in the overall security and functional reliability of the system.

Based on the dual-layer validation results using both correct and incorrect data, several recommendations are proposed for the robust implementation of AES-256 in academic environments. First, Key Management and Rotation. It is highly recommended to implement a structured key rotation policy and use hardware security modules (HSM) or dedicated key management services to further isolate cryptographic keys from the application environment.

Then Authentication Tags, Future implementations should consider transitioning from AES-CBC to AES-GCM (Galois/ Counter Mode). As noted in the discussion, AES-GCM provides built in integrity and authenticity through authentication tags (AEAD), which can prevent padding oracle attacks more effectively than CBC mode.

Next, Performance Monitoring. For large scale university systems, future research should integrate real-time performance monitoring to analyze the latency impact of field-level encryption on high-concurrency database queries.

After that Beyond Field Level. Future studies could explore "Always Encrypted" technologies with secure enclaves, allowing for complex search operations on encrypted data without ever exposing the plaintext to the database memory.

4. CONCLUSION

This study successfully implemented a field level encryption (FLE) scheme utilizing the AES-256-CBC algorithm to secure sensitive academic data. The experimental results demonstrated a 100% functional success rate, validated through 17 rigorous white-box testing scenarios covering internal logic and edge cases . API integration testing further confirmed that the system maintains complete data integrity across all CRUD operations . The analysis model explicitly differentiated between “True Data Tests,” which achieved perfect plaintext recovery, and “False Data Tests,” which successfully prevented unauthorized access by yielding decryption errors or unreadable outputs when incorrect keys were applied. These metrics confirm that the proposed application-layer approach effectively mitigates insider threats while ensuring robust confidentiality for higher education information systems.

Acknowledgment

The RisetMu program under the Majelis Diktilitbang PP Muhammadiyah funded this research. The authors express gratitude to Universitas Muhammadiyah Ponorogo and the internal IT team of the Faculty of Engineering.

5. REFERENCE

[1] J. Li, W. Xiao, and C. Zhang, "Data security crisis in universities: identification of key factors affecting data breach incidents," *Humanit. Soc. Sci. Commun.*, vol. 10, no. 1, p. 270, May 2023, doi: 10.1057/s41599-023-01757-0.

[2] K. Bhargavan and G. Leurent, "On the Practical (In-)Security of 64-bit Block Ciphers: Collision Attacks on HTTP over TLS and OpenVPN," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, in CCS '16. New York, NY, USA: Association for Computing Machinery, 2016, pp. 456–467. doi: 10.1145/2976749.2978423.

[3] M. Reza, A. F. Cobantoro, and I. A. Zulkarnain, "Hardening Server Menggunakan Metode Port Knocking Pada Sistem Program Studi Teknik Informatika Universitas Muhammadiyah Ponorogo," *Jurnal Ilmiah Informatika Komputer*, vol. 29, no. 3, pp. 298–315, Dec. 2024, doi: 10.35760/ik.2024.v29i3.12954.

[4] T. Ashur, O. Dunkelman, and A. Luykx, "Boosting authenticated encryption robustness with minimal modifications," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2017, pp. 3–33. doi: 10.1007/978-3-319-63697-9_1.

[5] E. Bonnie, "110 of the Latest Data Breach Statistics [Updated 2024]," SecureFrame.

[6] Tarisa Auliya Ramadhani, A. Fajaryanto Cobantoro, and S. Sugianti, "Implementasi Algoritma Advanced Encryption Standard 128 untuk Pengamanan Database Sistem Registrasi Pasien," *Jurnal Informatika Polinema*, vol. 10, no. 4, pp. 521–526, Aug. 2024, doi: 10.33795/jip.v10i4.5619.

[7] O. abd Qasim and S. Golshannavaz, "Data protection enhancement in smart grid communication: An efficient multi-layer encrypting approach based on chaotic techniques and steganography," *e-Prime - Advances in Electrical Engineering, Electronics and Energy*, vol. 10, p. 100834, Dec. 2024, doi: 10.1016/j.prime.2024.100834.

[8] R. K. M. K. Yalla, M. V. Devarajan, T. Ganesan, A. R. G. Yallamelli, V. Mamidala, and A. Sambas, "An effective security-aware side channel attack detection framework using RA-GRU and TPCC," *Australian Journal of Electrical and Electronics Engineering*, pp. 1–15, Mar. 2025, doi: 10.1080/1448837X.2025.2470571.

[9] A. B, M. K. H.S, and V. C M, "Advanced Encryption Standard (AES) Implementation in FPGA," in *2025 International Conference on Sensors and Related Networks (SENNET) Special Focus on Digital Healthcare(64220)*, IEEE, Jul. 2025, pp. 1–6. doi: 10.1109/SENNET64220.2025.11135984.

[10] O. Parulekar, S. Desai, A. Raut, and M. A. Gangarde, "Cryptography Using AES Algorithm," in *2024 Intelligent Systems and Machine Learning Conference (ISML)*, IEEE, May 2024, pp. 751–760. doi: 10.1109/ISML60050.2024.11007299.

[11] M. Haka, A. Haka, V. Aleksieva, and H. Valchanov, "Study of Encryption Time with the use of AES 128-, 192- and 256-Bit Keys," in *2025 19th Conference on Electrical Machines, Drives and Power Systems (ELMA)*, IEEE, Jun. 2025, pp. 1–4. doi: 10.1109/ELMA65795.2025.11083503.

[12] A. Bariant, J. Baudrin, G. Leurent, C. Pernot, L. Perrin, and T. Peyrin, "Fast AES-Based Universal Hash Functions and MACs: Featuring LeMac and Macs," *IACR Transactions on Symmetric Cryptology*, vol. 2024, no. 2, pp. 35–67, Jun. 2024, doi: 10.46586/tosc.v2024.i2.35-67.

[13] K. J. Lakshmi and G. Sreenivasulu, "Enhance Speed Low Area FPGA Design Using S-Box GF and Pipeline Approach on Logic for AES," *Mathematical Modelling of Engineering Problems*, vol. 11, no. 3, pp. 773–782, Mar. 2024, doi: 10.18280/mmep.110322.

[14] A. R. Okoro and G. U. Cantafio, "Cybersecurity Crisis Management in Higher Education Institutions," 2023, pp. 26–48. doi: 10.4018/978-1-6684-8332-9.ch002.

[15] N. Aleisa, "A comparison of the 3DES and AES encryption standards," *International Journal of Security and its Applications*, vol. 9, no. 7, pp. 241–246, 2015, doi: 10.14257/ijsia.2015.9.7.21.

[16] M. A. N. Acosta and H. Jahankhani, "An Empirical Study into Ransomware Campaigns Against the Education Sector and Adopting the Cybersecurity Maturity Model Certification Framework," 2023, pp. 67–103. doi: 10.1007/978-3-031-33627-0_4.

[17] K. Sivasubramanian, K. P. Jaheer Mukthar, V. Raju, and K. Srinivas, "The Impact of Digital Learning Management System on Students of Higher Education Institutions During Covid-19 Pandemic," 2022, pp. 657–678. doi: 10.1007/978-3-030-93921-2_34.

[18] D. Setya, "Data Mahasiswa UPI Diduga Bocor, Apa Langkah Pihak Kampus?," *Detik Edu*, Aug. 22, 2022.

[19] E. P. Nimasari, A. F. Cobantoro, S. D. Andika, and Moh. B. Setyawan, "Analisis Implementasi Teknologi Pembelajaran di Bebas UMPO," *Jurnal Ilmiah Edutic : Pendidikan dan*

Informatika, vol. 9, no. 2, pp. 162–177, Jun. 2023, doi: 10.21107/edutic.v9i2.19938.

[20] A. Alshammari, "Using analytics to predict students' interactions with learning management systems in online courses," *Educ. Inf. Technol. (Dordr).*, vol. 29, no. 15, pp. 20587–20612, Oct. 2024, doi: 10.1007/s10639-024-12709-9.

[21] E. R. G, S. M, A. D, V. A, J. Titus, and P. S, "AES+: A Modified AES Encryption with Enhanced Key Management and Distribution," in *2023 6th International Conference on Recent Trends in Advance Computing (ICRTAC)*, 2023, pp. 330–335. doi: 10.1109/ICRTAC59277.2023.10480804.

[22] S. Choi, D. Kim, and S. C. Seo, "Parallel Implementation of Scrypt: A Study on GPU Acceleration for Password-Based Key Derivation Function," *Journal of information and communication convergence engineering*, vol. 22, no. 2, pp. 98–108, Jun. 2024, doi: 10.56977/jicce.2024.22.2.98.

[23] S. Singh, P. K. Sharma, S. Y. Moon, and J. H. Park, "Advanced lightweight encryption algorithms for IoT devices: survey, challenges and solutions," *J. Ambient Intell. Humaniz. Comput.*, vol. 15, no. 2, pp. 1625–1642, Feb. 2024, doi: 10.1007/s12652-017-0494-4.

[24] W. Riddle *et al.*, "AESPIE: Raspberry Pi AES Performance Evaluation Using Image Encryption in C and Python," 2023, pp. 257–265. doi: 10.1007/978-3-031-40447-4_30.

[25] P. Udayakumar and R. Anandan, "Comparative Study of Lightweight Encryption Algorithms Leveraging Neural Processing Unit for Artificial Internet of Medical Things," *International Journal of Computational and Experimental Science and Engineering*, vol. 11, no. 1, Mar. 2025, doi: 10.22399/ijcesen.1259.

[26] Y. Qiu *et al.*, "Accelerating Authenticated Block Ciphers via RISC-V Custom Cryptography Instructions," in *2025 Design, Automation & Test in Europe Conference (DATE)*, IEEE, Mar. 2025, pp. 1–7. doi: 10.23919/DATE64628.2025.10992864.

[27] C. Mancillas-López and B. Ovilla-Martínez, "An Ultra-Fast Authenticated Encryption Scheme with Associated Data Using AES-OTR," *Journal of Circuits, Systems and Computers*, vol. 31, no. 09, Jun. 2022, doi: 10.1142/S0218126622501675.

[28] Z. Wu, K. Zhang, Y. Ren, J. Li, J. Sun, and W. Wan, "Visual Security Assessment via Saliency-Weighted Structure and Orientation Similarity for Selective Encrypted Images," *Security and Communication Networks*, vol. 2021, pp. 1–16, Jan. 2021, doi: 10.1155/2021/6675354.

[29] M. K. Thanikodi, "Advanced Encryption Standard Algorithm for Power-Efficient and High-Speed Applications," *Wirel. Pers. Commun.*, vol. 140, no. 1–2, pp. 225–239, Jan. 2025, doi: 10.1007/s11277-024-11693-0.

[30] V. Yesin, M. Karpinski, M. Yesina, V. Vilihura, R. Kozak, and R. Shevchuk, "Technique for Searching Data in a Cryptographically Protected SQL Database," *Applied Sciences*, vol. 13, no. 20, p. 11525, Oct. 2023, doi: 10.3390/app132011525.

[31] P. Antonopoulos *et al.*, "Azure SQL Database Always Encrypted," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, New York, NY, USA: ACM, Jun. 2020, pp. 1511–1525. doi: 10.1145/3318464.3386141.

[32] D. Costa, M. Teixeira, A. N. Pinto, and J. Santos, "High-performance blockchain system for fast certification of manufacturing data," *SN Appl. Sci.*, vol. 4, no. 1, p. 25, Jan. 2022, doi: 10.1007/s42452-021-04909-6.

[33] T. Monica, A. I. Hadiana, and M. Melina, "Question Bank Security Using Rivest Shamir Adleman Algorithm and Advance Encryption Standard," *JIKO (Jurnal Informatika dan Komputer)*, vol. 7, no. 3, pp. 175–181, Nov. 2024, doi: 10.33387/jiko.v7i3.8654.

[34] V. Casola, A. De Benedictis, C. Mazzocca, and V. Orbinato, "Secure software development and testing: A model-based methodology," *Comput. Secur.*, vol. 137, p. 103639, Feb. 2024, doi: 10.1016/j.cose.2023.103639.

[35] L. Leonardi, G. Lettieri, P. Perazzo, and S. Saponara, "On the Hardware–Software Integration in Cryptographic Accelerators for Industrial IoT," *Applied Sciences*, vol. 12, no. 19, p. 9948, Oct. 2022, doi: 10.3390/app12199948.