Transport and Telecommunication, 2025, volume 26, no. 4, 362-372
Transport and Telecommunication Institute, Lauvas 2, Riga, LV-1019, Latvia
DOI 10.2478/ttj-2025-0028

A NOVEL AES-BASED ENCRYPTION MODEL WITH MULTI-FEATURE SECURITY ENHANCEMENTS

Mohammad Awis Al Lababede[1], Ashraf M A Ahmad[2]

[1,2] Department of Computer Science, King Hussein School of Computing Sciences,
Prince Sumaya University for Technology
Amman, Jordan, PO Box 1438, 11941
[1]awis97@hotmail.com

Information security is one of the critical challenges for organizations and individuals. For that, they try to obtain an encryption algorithm that provides enhanced protection, in addition to offering multiple security features. Common encryption algorithms often offer only encryption functionality directly, while some modes are used to ensure additional features such as authentication and integrity. However, these modes are also vulnerable to various attacks and have several limitations. Therefore, in this paper, we present a new model that enhances AES performance by introducing new features such as authentication and integrity, and by resisting common issues like bit flipping. The proposed model's performance was evaluated on a textual dataset and then compared with several robust algorithms, including AES, DES, XDES, and Triple DES. The results show that the proposed model's performance was very good and nearly equivalent to these algorithms, while adding the aforementioned features directly, without relying on any modes.

Keywords: AES, encryption, security enhancements, integrity, authentication

1. Introduction

With the increasing volume of data and the increasing frequency of data transfer and storage operations, individuals and institutions struggle to maintain the confidentiality and integrity of information (Anyanwu *et al*., 2024). Especially in sensitive sectors such as military, banking, and healthcare (Fatima *et al*, 2019), the confidentiality, integrity, and availability of data are critical to ensuring operational security and trust. Unauthorized access or modification of data in these fields can lead to significant financial loss, compromised national security, or threats to human health. Therefore, robust encryption and authentication mechanisms are indispensable to protect sensitive information and maintain system reliability. Any unauthorized modification or exposure of transmitted data can lead to severe consequences. As a result, organizations have been pushed to implement stronger data security measures. Among these measures are the main principles of the CIA triad (Confidentiality, Integrity, and Availability) along with authentication mechanisms. Confidentiality means that only authorized parties can access the transmitted or stored data. Integrity makes sure that data stays accurate and untampered with, while availability ensures continuous access to data without disruption. Authentication checks who is sending and receiving the data, ensuring the messages are legitimate. Cryptography plays a key role in achieving confidentiality by securing data through encryption, while the additional steps ensure the integrity and authentication.

Cryptology encompasses both cryptography and cryptanalysis. While cryptography focuses on securing data, cryptanalysis works to break cryptographic systems to access sensitive information. Cryptographic techniques are categorized into symmetric and asymmetric methods. Symmetric encryption uses a single key for both encryption and decryption, whereas asymmetric encryption relies on a pair of keys, a public key for encryption and a private key for decryption.

Symmetric encryption works in two modes: block encryption and stream encryption. Block encryption processes fixed-size data blocks, such as Advanced Encryption Standard (AES), Data Encryption Standard (DES), and Triple Data Encryption Standard (3DES). In contrast, stream encryption secures data in real-time at the bit level, such as Linear-feedback Shift Register (LFSR) and A5.

AES is a symmetric, block-based encryption algorithm (Arman *et al.*, 2024) and is considered one of the most secure and efficient methods for data protection. It operates using three key sizes: AES-128, AES-192, and AES-256 (Zodpe and Sapkal, 2020), with the strength of encryption directly depending on the key size. AES secures information by applying the principles of confusion and diffusion. To further

enhance its capabilities, it is often combined with various modes that provide additional features such as data integrity and authentication. Some of the well-known modes include Electronic Codebook (ECB), Cipher Block Chaining (CBC), Output Feedback (OFB), Cipher Feedback (CFB), and Galois/Counter Mode (GCM). These modes introduce extra steps surrounding the encryption process to meet specific security requirements.

Despite the wide adoption of AES due to its efficiency and security, it still faces challenges related to certain attack vectors and operational modes. For example, the Electronic Codebook (ECB) mode is vulnerable to pattern repetition, making encrypted data susceptible to analysis. Similarly, the Cipher Block Chaining (CBC) mode can be targeted by bit-flipping attacks, compromising data integrity. These challenges highlight the need for novel enhancements to AES that can reinforce its resistance against emerging threats without compromising performance. To address these issues, this paper proposes a new enhancement by introducing a pre-encryption layer. This layer is designed to strengthen AES by ensuring confidentiality, integrity, and authentication through several mechanisms:

- Generating a unique value from the encryption key using a hashing process and embedding parts of it into each block for authentication purposes.
- Incorporating the sender's unique identifier and merging it with the hashed unique number.
- Introducing a new shifting operation, based on a value derived from the key, to further obscure the data.

While previous research has contributed valuable modifications to AES, many have focused primarily on performance improvements or isolated security aspects. Few studies address a comprehensive enhancement that integrates confidentiality, integrity, and authentication simultaneously. Moreover, approaches that embed unique identifiers derived from the key into the encryption process remain underexplored. To address these gaps, this paper proposes a pre-encryption layer that incorporates multiple security mechanisms to strengthen AES comprehensively. Unlike existing enhancements that focus on modifying internal AES operations, the proposed layer introduces external manipulations that add structural unpredictability and enhance resilience against a wider range of attacks.

Contributions

This paper aims to introduce several pre-processing steps before the encryption process to achieve the following:

- Confidentiality: Ensured through the encryption steps of AES.
- Integrity: Achieved by embedding specific information into the block (e.g., a personal identifier, a secret number derived from the key, and the block number), enabling error detection during verification if any modifications occur
- Authentication: Accomplished by including a hidden secret identifier within the plaintext before encryption.
- Resolving the Bit-Flipping Issue in CBC Mode: Addressed by shuffling the blocks and applying a shift operation before encryption.
- Achieving Integrity without Using Modes: Enhancing AES to provide integrity without relying on traditional modes.
- Addressing Pattern Repetition in ECB Mode: Preventing the repetition of patterns that occur during block encryption in ECB mode.
- Mitigating Block Stability and Sequentially: By shuffling the blocks after encryption, the algorithm improves its resistance to duplicate code and statistical attacks.
- Enhancing AES Modes: Ensuring that all modes, when used with AES, inherently support both authentication and integrity.

The paper is organized as follows: it begins with a review of some related studies relevant to the proposed model. This is followed by the methodology and the steps involved in the model's development. Next, the Implementation section explains the construction process of the proposed model step by step. The Experiment Setup section follows, where the preparation of the proposed model and other algorithms for comparison is discussed. The Results section presents the findings and compares the algorithms. Finally, the paper concludes with a conclusion and future work.

2. Related works

In this section, we review various efforts by researchers to develop and modify the AES algorithm for better performance. We also highlight key modifications aimed at enhancing AES.

Over the past decade, a significant number of researchers have explored ways to boost AES performance, focusing particularly on speed and energy efficiency. For instance, Awan *et al.* (2020) worked on increasing the speed of AES and succeeded in achieving a processing rate of 1,000 blocks per second. Similarly, Zodpe and Sapkal (2020) introduced improvements by designing a more secure, modified S-box and optimizing the process of generating the initial key needed for encryption and decryption.

In another study, D'souza and Panchal (2017) proposed modifications to make AES more resistant to attacks, implementing techniques such as Dynamic Key Generation and Dynamic S-box Generation.

Efforts to enhance AES were not limited to data files; researchers also applied improvements to multimedia encryption. For image encryption, Mohammed *et al*. (2024) proposed a method based on fragmenting images into multiple pieces, aiming to maximize data confidentiality. On the other hand, enhancements to AES were extended to audio encryption as well. Mossa (2017) developed an improved version of AES for audio signals by applying XOR operations and embedding an authentication tag within the signal, thereby strengthening AES against brute-force attacks.

Additionally, some researchers have focused on making AES lighter and more suitable for resource-constrained devices. Mohammad and Abdullah (2022), for example, reduced the computational complexity of AES by replacing the MixColumns operation with a more cost-effective technique, specifically the continued fraction method.

Addressing security threats like bit-flipping attacks was another important area of focus. Alslman *et al*. (2023) tackled this by modifying the CBC mode: they proposed computing a hash of the ciphertext and using it as an Initial Vector (IV). This method helps mitigate bit-flipping issues and adds a layer of verification to the encryption process. Their results demonstrated the success and potential of this approach.

Finally, in an effort to enhance AES-CBC performance for Internet of Things (IoT) devices, Lee and Sim (2021) introduced a technique of periodically changing the IV. They also proposed combining two encryption keys to generate the main AES key, which led to further improvements in both performance and security.

Many studies have sought to improve AES in various ways, such as speeding up encryption (Awan *et al*., 2020), designing modified S-boxes (Zodpe and Sapkal, 2020), or enhancing resistance to attacks through dynamic key and S-box generation (D'souza and Panchal, 2017). Other works focused on specific data types like images (Mohammed *at al*., 2024) and audio (Mossa, 2017), or tailored AES for resource-constrained devices (Mohammad and Abdullah, 2022). Furthermore, efforts to mitigate bit-flipping attacks by modifying CBC mode (Alslman *et al*., 2023) or periodically changing the initialization vector for IoT applications (Lee and Sim, 2021) have shown promise.

However, the proposed approach distinguishes itself by combining unique key-derived identifiers, pre-encryption shifting operations, and block shuffling to simultaneously address confidentiality, integrity, authentication, and known mode-specific vulnerabilities. This holistic enhancement extends the security features of AES beyond what previous works have achieved.

2.1. Research gap

While various enhancements have improved AES in specific areas, a unified approach that simultaneously ensures confidentiality, integrity, and authentication, while also addressing vulnerabilities tied to specific operational modes, remains largely unexplored. This study addresses that gap by introducing a novel, structured, and lightweight pre-encryption mechanism designed to enhance AES across multiple security dimensions.

Most existing AES improvements tend to concentrate on one aspect of security or performance—for example, modifying the S-box to enhance confusion, or adjusting key scheduling to increase unpredictability. Only a few studies have embedded authentication and integrity directly into the encryption process itself. Even fewer have addressed structural issues related to specific encryption modes, such as pattern exposure or block rigidity. In contrast, the proposed model incorporates all of these features into a single, lightweight framework, eliminating the need for external tools or reliance on particular operation modes.

3. Methodology

This section outlines the full workflow of the proposed model, detailing each operation that enhances the AES algorithm through external pre-processing layers. The process ensures the integration of confidentiality, integrity, and authentication while improving resilience against pattern analysis and modification attacks. The methodology consists of the following sequential steps:

3.1. Key processing and hashing

The process begins by applying a SHA-256 hash function to the original AES (128) encryption key. This produces a fixed-size hash value 'x'. To increase entropy and add structural complexity, the hash is passed through the following polynomial transformation:

$$y = x2 \ + x + 5\,, \tag{1}$$

where *x* represents the hash value. This polynomial ensures both an increase in the extracted value's length (*x*) and its complexity. Other polynomials can also be used, but this particular one was sufficient based on experimental results.

This expanded value '*y*' is then converted into binary format to prepare for subsequent tagging and shifting operations.

3.2. Tag generation per block

Authentication and integrity are enforced through a dynamically changing tag, calculated individually for each block. The process for a single block is as follows:

- A sender-specific identifier (ID) is converted into binary format, referred to as ID_{bin}.
- The previously generated hash value y is converted into binary format, referred to as y_{bin}.
- For each block:
  - A different bit from the 'ID' and a different set of three bits from the binary representation of 'y' are extracted (e.g., block 0 uses bits 0–2 from 'y' and bit 0 from 'ID', block 1 uses bits 3–5 from 'y' and bit 1 from 'ID', and so on).
  - These four bits are concatenated to produce a "4-bit tag", which is appended to each block before encryption.

This tag ensures both authentication (via the ID bit) and integrity (via the bits derived from the key hash).

Additionally, the value:

$$W = y \ mod \ 128 \tag{2}$$

is computed at this stage. It determines the shift amount applied later in the process and changes with any slight modification to the original key.

This process increases encryption complexity, disrupts attackers, and addresses known vulnerabilities such as bit-flipping attacks, pattern analysis, and statistical analysis. An additional step is introduced: a unique number will be assigned to each block before encryption (counter). These blocks will then be shuffled, ensuring that even identical plaintexts result in distinct ciphertexts due to the differences in block numbers. As is well known, any variation in the plaintext necessarily leads to a difference in the ciphertext.

3.3. Block structuring

Each block is assigned a unique sequential number (0 to 999), which is encoded using 3 bytes. The structure of each block before encryption is:

Block Format: [Block Number (3 bytes)] + [Tag (1 byte)] + [Plaintext]

This structure allows integrity verification, authentication, and block identification after decryption.

3.4. Shifting operation

Each complete block is subjected to a **bitwise left shift** by `w` bits. This operation obfuscates the internal structure of the block and introduces additional complexity, making ciphertext analysis more difficult. This process helps to obscure the added information; row shifts will be applied to the plaintext based on values derived from the encryption key. This process ensures that extracting values from the key and modifying the text accordingly is nearly impossible for anyone without knowledge of the key, and thus achieving authentication or decrypting the text is also infeasible. Even if the key is known, the authentication mechanism guarantees that creating a valid plaintext or modifying it without the correct personal identifier remains unfeasible.

3.5. Padding

Two types of padding are applied:

- "Block-level padding": Ensures each block conforms to the fixed AES block size.
- "Full-text padding": Ensures that the total message length is divisible by the block size, avoiding data loss.

3.6. Block shuffling

To prevent attackers from correlating patterns between input and output, blocks are “shuffled” randomly (or using a key-derived sequence). This disrupts any sequential or repetitive patterns in the ciphertext.

3.7. AES encryption

Finally, each block, now tagged, numbered, shifted, padded, and shuffled, is passed through the “standard AES encryption” using the original key. The internal structure of AES remains unchanged; the enhancements are implemented entirely before the encryption phase. This layered approach ensures that any identical plaintext block encrypted multiple times will result in “different ciphertexts” due to variation in tags, block order, and shifting.

3.8. AES decryption

The decryption process involves reversing all the encryption steps:

- Compute the hash of the key to derive *y* and *W*.
- Decrypt the ciphertext using AES.
- Remove padding from the full text.
- Divide the text into blocks and apply a right shift by *W*.
- Extract the first three values (block number) and the fourth value (tag) from the plaintext.
- Rearrange the blocks according to their numbers.
- Convert the tag back to binary and compare it with y and ID.

A mismatch in y indicates that the text has been modified, while a mismatch in the ID signifies an untrusted sender. Figures 1 and 2 visually illustrate the encryption and decryption pipelines, respectively.

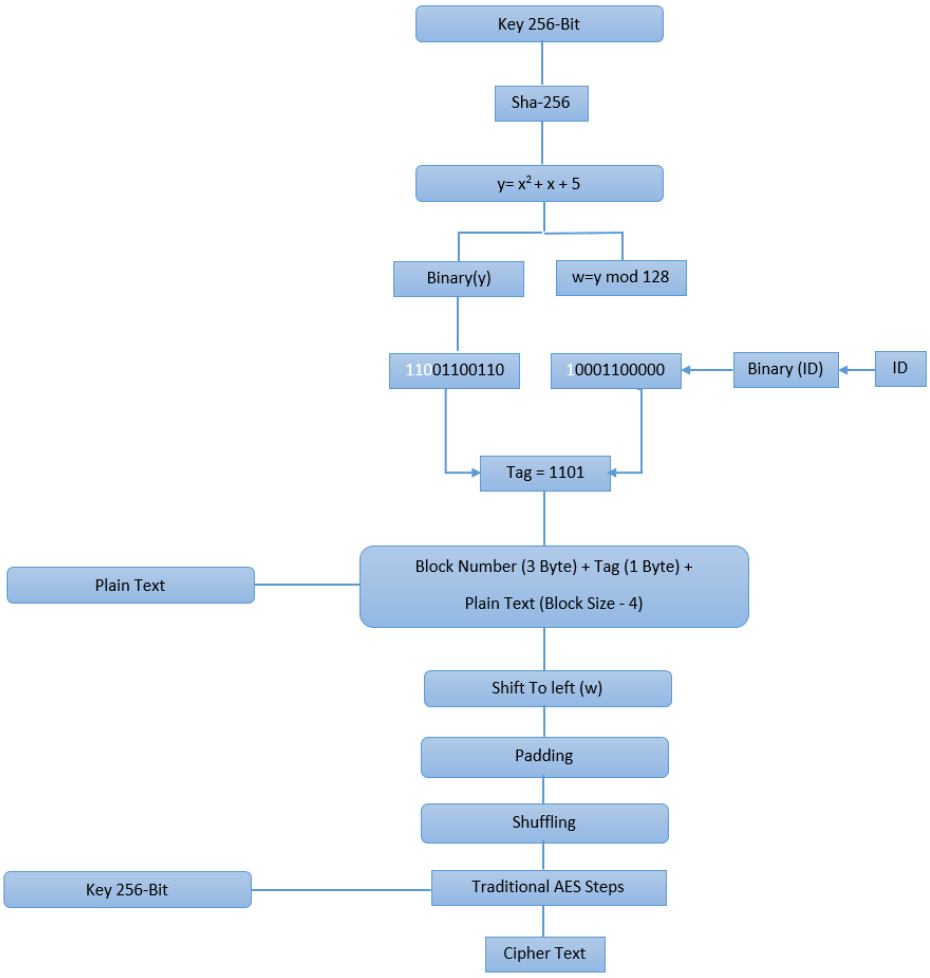

Figure 1. The encryption process in the proposed model

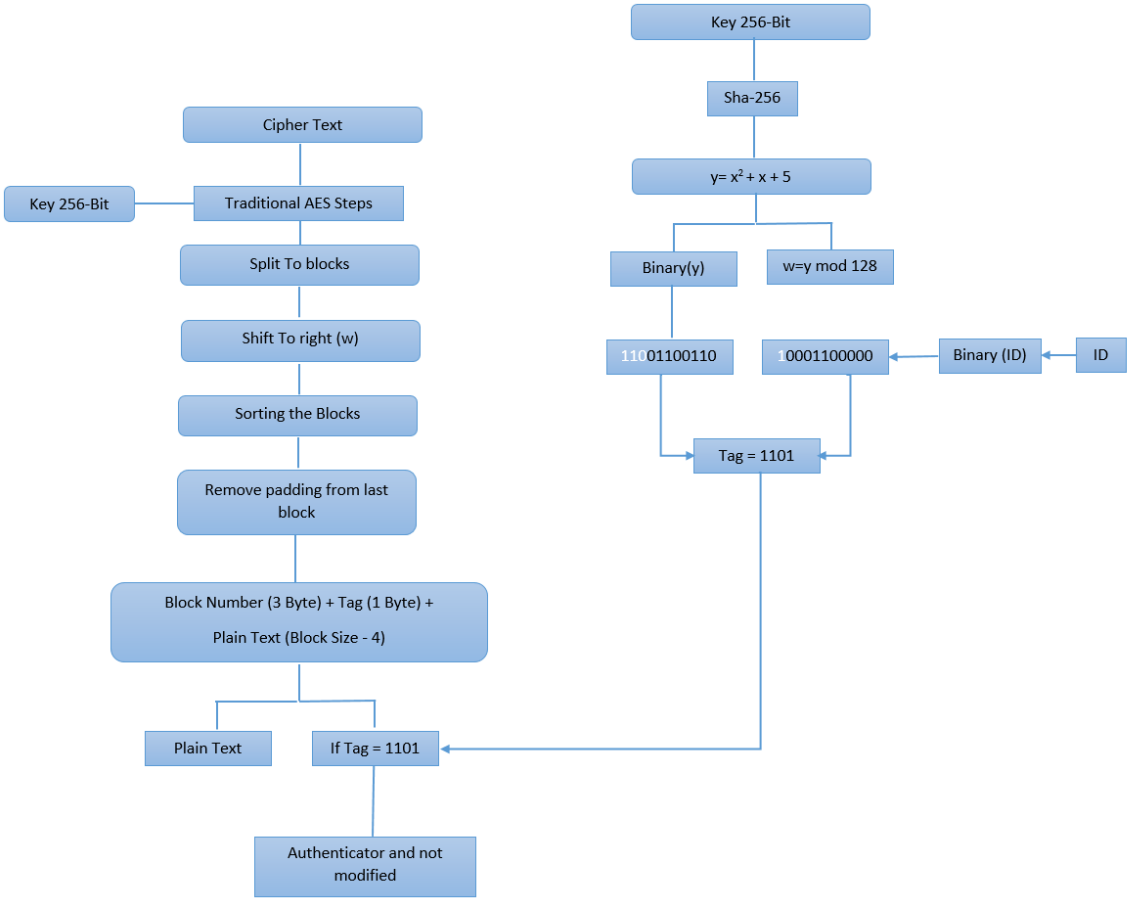

Figure 2. The Decryption process in the proposed model

4. Implementation

4.1. Key processing and hashing

The first step involves extracting a unique number from the key. To ensure that this number changes with at least a one-bit change in the key, we use the hash function. This function generates a unique hash value for any given textual or numeric input. In the proposed model, we utilize SHA-256 to create a unique hash value for the key.

Once the hash value is generated, it results in a relatively long numeric sequence. However, this length may not be sufficient for encryption purposes. To address this, we apply the polynomial in (1) to extend the hash value's size and simultaneously increase its complexity. After preparing the extracted value (y), it is converted into binary format to prepare it for the next stage.

4.2. ID encoding (authentication input)

To achieve authentication, a unique identifier specific to the sender is required, allowing the receiver to verify the sender's identity. For this purpose, the identifier value (ID) is used as the sender's unique identifier. The ID is transformed into its binary representation. If the lengths of ID or y are insufficient, they are repeated to ensure adequate size.

4.3. Tag generation per block

To generate the tag (TAG), we combine the two unique values (y and ID). Specifically, three bits from y are merged with one bit from ID. The resulting binary value is then converted into a decimal number, which is appended to the plaintext before encryption.

This step optimizes the space used to represent the tag while simultaneously obfuscating the values of both ID and y.

In general format:

- For each block i:
- Extract a single bit from ID_{bin} at position i mod len(ID_{bin}), and
- Extract 3 consecutive bits from y_{bin} starting at position 3i mod len(y_{bin}) (i.e., bits y_{bin}[3i mod L], y_{bin}[(3i+1) mod L], and y_{bin}[(3i+2) mod L], where L = len(y_{bin})).
- These four bits are concatenated to form the 4-bit tag for that block:
- TAG_i = concat (i mod len(ID_{bin}), y_{bin}[3i mod L], y_{bin}[(3i+1) mod L], y_{bin}[(3i+2) mod L])

This design ensures that even a small change in the key or identifier will propagate to different tags across all blocks. The tag is then appended at the beginning of each block prior to encryption.

4.4. Block number assignment

To enable block shuffling and reordering, a block number is added to each block. Blocks are numbered sequentially from 0 to 999, allowing up to 999 blocks to be processed at a time. Three bytes are required to represent the block number.

Increasing the block count is possible, but this reduces the size of each block. For instance, 4 bytes are needed for 9,999 blocks, 5 bytes for 99,999 blocks, and so on.

4.5. Block structuring and formatting

After appending both the block number and the tag at the start of the block, the block structure appears as follows:

Block Format: [Block Number (3 bytes)] + [Tag (1 byte)] + [Plaintext]

Here, the plaintext size is reduced to 16−4 bytes to accommodate the additional metadata.

4.6. Bitwise left shift (obfuscation layer)

To prevent the ciphertext from being structurally predictable, each block undergoes a bitwise left circular shift. The number of shift positions, denoted as w, is derived using the formula (2) above.

This operation is applied uniformly across all bits in the full block (including block number, tag, and plaintext). It significantly obfuscates the bit pattern of the original plaintext, and since w is directly dependent on the hash of the key, any slight key change will cause a completely different shift pattern.

4.7. Padding mechanism

Padding is applied in two stages:

Block Padding: Added when the final block's length is shorter than the standard block size.

Full Text Padding: Applied to the entire text before encryption to ensure alignment with the encryption algorithm.

4.8. Block shuffling (permutation layer)

After formatting and shifting, all blocks are shuffled. This can be done using a pseudo-random permutation derived from the key (e.g., using a deterministic key-based PRNG). The block numbers embedded in each block allow correct reordering during decryption. This step significantly mitigates known-mode attacks such as pattern repetition in ECB and bit-flipping in CBC.

4.9. Final AES encryption

The prepared text, with all necessary values added and transformations applied, is then encrypted using AES in its standard form without any additional modifications.

5. Experiment setup

In this step, the proposed algorithm and the other primary encryption algorithms will be initialized, and the required parameters and settings will be configured. Key components, such as the ID value, encryption key, and operational mode, will be defined.

To ensure a fair and unbiased comparison, we utilized the Hazmat module, which is part of the widely used Cryptography library (Cryptography.io., 2024). This module provides consistent and standardized implementations for cryptographic algorithms. By using Hazmat, we ensure that all algorithms follow the same coding practices and optimizations, allowing for a reliable and objective evaluation of their performance.

5.1. The proposed model

The proposed model is implemented using Python with several specialized libraries, including cryptography, hazmat, hashlib, random, and others. These libraries provide the necessary tools for hashing, encryption, and randomization operations. For hashing, the SHA-256 algorithm is used to compute the values of y and w, which play a key role in block manipulation and encryption. SHA-256 is chosen for its robustness and ability to produce unique and secure hash values. For key size, we use AES-256 with a 32-byte key: KISTFN5856NNDFLO0IWFTSNNVR698USR. This key size is selected to ensure high security and compliance with modern cryptographic standards. The ID value used in this model is 99871457.

To isolate the effect of the operational modes on the proposed model's performance, the ECB mode is employed. This mode works with the plaintext as separate, independent blocks, making it suitable for evaluating block-level transformations while minimizing inter-block dependencies.

5.2. Advanced Encryption Standard (AES)

To implement AES, we will use the Hazmat module, which is part of the Cryptography library. In this experiment, we will use AES-256 with a key size of 32 bytes. We will apply padding to ensure that the size of the plaintext matches the block size. The mode of operation will be set to ECB. The key used is "KISTFN5856NNDFLO0IWFTSNNVR698USR".

5.3. Triple Data Encryption Standard (Triple-DES)

To implement Triple-DES, we will use the Hazmat module. Triple DES consists of three sequential applications of the DES algorithm with three different keys. The first key is used to encrypt the plaintext, the second key decrypts the result of the first step, and the third key encrypts the output of the second step. These three keys are combined into a single key with a size of 24 bytes (192 bits). We will apply padding to ensure that the size of the plaintext matches the block size. The mode of operation will be set to ECB. The key used is " KISTFN5856NNDFLO0IWFTSNN".

5.4. Data Encryption Standard (DES)

In the Hazmat module, DES is not implemented directly, so we will implement DES in an indirect way. Triple DES (3DES) consists of three sequential applications of the DES algorithm using three keys. However, we can simulate DES by using Triple DES with the same key for all three steps. In this case, the first and second steps cancel each other out, and the third step encrypts the plaintext using the key. These three keys are combined into a single key with a size of 24 bytes (192 bits).

We will apply padding to ensure that the size of the plaintext matches the block size. The mode of operation will be set to ECB. The key used in this implementation is "KISTFN58KISTFN58KISTFN58".

5.5. X Data Encryption Standard (DES-X)

As with DES, XDES is not directly implemented in the Hazmat module, so we will use an indirect implementation with some modifications. XDES follows the same process as DES but introduces additional XOR operations for added security.

Before inputting the plaintext into the DES model, we perform an XOR operation between the plaintext and a key (key1). After completing the encryption process, another XOR operation is performed between the resulting ciphertext and key2. This ensures that the text is effectively scrambled, enhancing its security.

In XDES, we use three keys:

- Key0: A 24-byte key used for the DES encryption process.
- Key1: A key used for XOR before encryption.
- Key2: A key used for XOR after encryption.

Padding will be applied to ensure that the plaintext size matches the block size. The mode of operation will be set to ECB. The keys used in this implementation are:

- Key0: "KISTFN58KISTFN58KISTFN58" (24 bytes).
- Key1: b" ISTF3SAN" (8 bytes).
- Key2: b" S6S8FHKJ" (8 bytes).

5.6. Dataset

In this experiment, we used a substantial English text titled Moby Dick [8], which contains 213,674 words and over 1.2 million characters, spread across 135 chapters, along with an introduction, conclusion, and additional sections.

6. Results and discussion

Now that the algorithms have been implemented, we will evaluate their performance. In cryptography, evaluating algorithms is a critical matter because it is challenging to measure their performance comprehensively. A realistic evaluation often involves testing algorithms against popular attacks, such as brute force, or analyzing specific metrics like speed and correlation. While these metrics alone may not be sufficient for a complete evaluation, they are widely accepted as useful indicators.

In this paper, we will assess the speed of each algorithm by calculating the average time it takes to encrypt and decrypt the dataset. Additionally, we will measure the correlation between the plaintext and ciphertext (i.e., the text before and after encryption).

Table 1. Result for all images

No	Algorithm	Average Encryption Time	Average Decryption Time	Average Correlation	Total Processing Time
1	Proposed	0.0003s	0.0002s	0.0047	2.8383s
2	AES	**0.0001s**	**0.0000s**	0.0043	**1.9553s**
3	Triple-DES	**0.0001s**	0.0001s	0.0040	2.1287s
4	DES	**0.0001s**	0.0001s	0.0043	2.1218s
5	X-DES	0.0007s	0.0006s	**-0.0009**	3.6387s

Table 1 highlights the superiority of AES in terms of average decryption speed and overall encryption and decryption time. Although AES matched DES and Triple DES in encryption time, it exhibited a relatively higher correlation between plaintext and ciphertext, which is less desirable from a cryptographic perspective. On the other hand, DES and Triple DES demonstrated the best performance in encryption and decryption speed, along with a good correlation between the original text and the encrypted result.

As for XDES, it exhibited excellent performance in terms of correlation but required significantly more time for the encryption and decryption process compared to the other algorithms. The proposed algorithm demonstrated strong performance in encryption and decryption speed compared to X-DES. Furthermore, its correlation value (0.0047) suggests a very low similarity between plaintext and ciphertext, slightly higher than AES and DES, which reflects strong diffusion and enhances resistance to statistical attacks.

Despite a minor increase in processing time compared to classical AES (0.0003s vs. 0.0001s), the added security features, such as authentication and bit-flipping resistance, justify this computational overhead.

The slight increase in encryption and decryption time observed in the proposed model is reasonable and justified, considering the additional pre-processing operations involved. Despite this increase, the overall processing time remains lower than that of X-DES, which is already a validated and effective algorithm. Therefore, the additional time cost is acceptable given the significant security enhancements offered by the proposed model, including authentication, integrity, and robustness against certain types of attacks.

Table 2. Comparison between the algorithms

No	Features	AES	Triple-DES	DES	X-DES	Proposed
1	Confidentiality	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**
2	Integrity	No	No	No	No	**Yes**
3	Authentication	No	No	No	No	**Yes**
4	Bit-flipping resistant	No	No	No	No	**Yes**
5	Addressing Pattern Repetition	No	No	No	No	**Yes**
6	Mitigating Block Stability	No	No	No	No	**Yes**

Unlike traditional AES or DES implementations, which do not include any form of authentication or integrity verification unless operated in specific modes (such as GCM or CCM), the proposed model integrates these features directly into the preprocessing layer. This allows even ECB-mode AES to achieve enhanced security without requiring external modifications.

This native support for integrity and authentication addresses critical gaps in traditional encryption models, particularly in environments where mode switching or additional cryptographic primitives are undesirable or limited.

Table 2 presents a comparison of the algorithms and the features they offer without employing any auxiliary modes.

Overall, the results demonstrated that the proposed model's performance, in terms of speed and the correlation between the original and encrypted text, was good and closely aligned with other algorithms, although other algorithms outperformed it in many aspects. However, as we know, not all algorithms excel in every functionality; their performance varies. Some are better in terms of speed, while others excel in correlation. The proposed model stands out by integrating multiple security features natively that enhance its performance, making it one of the top-performing algorithms. Specifically, it offers:

- Confidentiality
- Integrity
- Authentication
- Bit-flipping resistance
- Addressing pattern repetition
- Mitigating block stability

Other algorithms typically focus solely on confidentiality. Some may support additional features, but only with the use of specific modes.

Overall, the results demonstrate that while the proposed model may not outperform existing algorithms in every individual metric, it offers a balanced and robust encryption approach that natively incorporates multiple security properties. Unlike traditional algorithms that focus solely on confidentiality, the proposed model integrates integrity, authentication, and resistance to several known attacks directly into the encryption process. These enhancements make it particularly suitable for applications where both security and performance are required without relying on specific encryption modes.

7. Conclusion and further works

Data protection is considered one of the most critical challenges faced by organizations and individuals. As a result, they seek encryption algorithms that help safeguard their information. However, not all of these algorithms provide comprehensive and complete features simultaneously, such as confidentiality, integrity, authentication, bit-flipping resistance, addressing pattern repetition, and mitigating block stability. In this paper, we present a model that enhances the AES algorithm to incorporate all of the above features without the use of any auxiliary modes. The performance of the proposed model was tested on a set of text data. The results demonstrated that the proposed model's performance, in terms of speed and the correlation between the original and encrypted text, was very close to that of XDES, DES, AES, and Triple DES, while providing the features of confidentiality, integrity, authentication, bit-flipping resistance, addressing pattern repetition, and mitigating block stability.

In the future, the performance of the proposed model can be tested on larger image datasets to evaluate its effectiveness. Additionally, its performance and speed may be improved by reducing the block number size by encoding the block number in bits rather than bytes.

Declaration of Generative AI and AI-assisted technologies in the writing process:

During the preparation of this manuscript, the authors used artificial intelligence tools for language proofreading only. After using these tools, the authors reviewed and edited the content as needed and take full responsibility for the content of the publication.

References

1. Alslman, Y.S., Ahmad, A. and AbuHour, Y. (2023) Enhanced and authenticated cipher block chaining mode. *Bulletin of Electrical Engineering and Informatics*, 12(4), 2357–2362. DOI:10.11591/eei.v12i4.5113.
2. Anyanwu, A., Olorunsogo, T., Abrahams, T., Akindote, O., Reis, O. (2024) Data confidentiality and integrity: a review of accounting and cybersecurity controls in superannuation organizations. *Computer Science & IT Research Journal*, 5(1), 237–253. DOI:10.51594/csitrj.v5i1.735.
3. Arman, M.S., Mamun, S.A. and Jannat, N. (2024) A modified AES based approach for data integrity and data origin authentication. In: *Proceedings of 2024 3rd International Conference on Advancement*

in Electrical and Electronic Engineering (ICAEEE), Gazipur, April 2024. IEEE, 106. DOI:10.1109/ICAEEE62219.2024.10561750.
4. Awan, I. A., Shiraz, M., Hashmi, M.U., Shaheen, Q., Akhtar, R., Ditta, A. (2020) Secure framework enhancing AES algorithm in cloud computing. *Security and Communication Networks*, 1-6, 8863345. DOI: 10.1155/2020/8863345.
5. Cryptography.io. (2024) Cryptography Hazmat Primitives: Symmetric Encryption. Available at: https://cryptography.io/en/latest/hazmat/primitives/symmetric-encryption/#cryptography.hazmat.primitives.ciphers.Cipher (Accessed: 18 November 2024).
6. D'souza, F.J. and Panchal, D. (2017) Advanced encryption standard (AES) security enhancement using hybrid approach. In: *Proceedings of International Conference on Computing, Communication and Automation (ICCCA),* Greater Noida, May 2017. IEEE, 647-652. DOI:10.1109/CCAA.2017.8229881.
7. Fatima, K., Nawaz, S. and Mehrban, S. (2019) Biometric authentication in health care sector: A survey. In: *Proceedings of International Conference on Innovative Computing (ICIC),* Lahore, Pakistan. IEEE, 1-10, DOI: 10.1109/ICIC48496.2019.8966699.
8. Kaggle. (2020) Moby Dick. Available at: https://www.kaggle.com/datasets/razetime/moby-dicktext.
9. Lee, S.W. and Sim, K.B. (2021) Design and hardware implementation of a simplified DAG-based blockchain and new AES-CBC algorithm for IoT security. *Electronics*, 10(9), 1127. DOI: 10.3390/electronics10091127.
10. Mohammad, H.M. and Abdullah, A.A. (2022) Enhancement process of AES: a lightweight cryptography algorithm-AES for constrained devices. *TELKOMNIKA (Telecommunication Computing Electronics and Control),* 20(3), 551–560. DOI:10.12928/telkomnika.v20i3.23297.
11. Mohammed, Z.A., Gheni, H.,l., Hussein, Z., Al-Qurabat, A. (2024) Advancing cloud image security via AES algorithm enhancement techniques. *Engineering, Technology & Applied Science Research*, 14(1), 12694–12701. DOI:10.48084/etasr.6601.
12. Mossa, E. (2017) Security enhancement for AES encrypted speech in communications. *International Journal of Speech Technology*, 20(1), 163–169. DOI:10.1007/s10772-017-9395-3.
13. Mukhopadhyay, D., Sonawane, G., Gupta, P., Bhavsar, S., Mittal, V. (2013) Enhanced security for cloud storage using file encryption. DOI:10.48550/arXiv.1303.7075.
14. Zodpe, H. and Sapkal, A. (2020) An efficient AES implementation using FPGA with enhanced security features. *Journal of King Saud University – Engineering Sciences*, 32(2), 115–122. DOI:10.1016/j.jksues.2018.07.002.