Technological Applied and Humanitarian Academic Journal
https://riyadahpublishers.com/index.php/tahaJ

TAHAJ

2025, Vol.2 No. 1, page No: 18-37

Research Article

Open Access

Enhancing Performance Through Dynamic AES Round Keys and Adaptive Data Shifting Techniques

Adel Mohammed Ali Al-Qashbari [1], Nabil Mohammed Ali Munassar [1], Mohammed Fadhl Abdullah[1] , Ahmed Saleh Khaled AL-Hurdi [1]

[1] IT Department, College of Engineering and Computing, University of Science and Technology, Aden, Yemen.

[*]*Corresponding author: a.alqashbari@ust.cedu*

Received: January 16, 2025 Accepted: February 15, 2025 Published: February 26, 2025

Cite this article as: Adel Mohammed Ali Al-Qashbari, Nabil Mohammed Ali Munassar, Mohammed Fadhl Abdullah, Ahmed Saleh Khaled AL-Hurdi. Enhancing Performance Through Dynamic AES Round Keys and Adaptive Data Shifting Techniques. Technological Applied Humanitarian Academic Journal (TAHAJ). 2025;2(1):18-37.

Abstract:
The query discusses two encryption methods: standard AES (acting as the control group) and a new hybrid encryption algorithm (the experimental group). Symmetric encryption, such as AES, uses a private key for both encryption and decryption, applying multiple constants and iterations throughout the process. The study proposes modifications to the AES algorithm by omitting certain steps—specifically, the substitution and mixcolumn processes—while still utilizing a single key derived from a random primary key. This approach is accompanied by dynamic data displacement to bolster security and improve execution time.
The research further indicates that traditional encryption methods struggle to find a balance between maintaining strong security, operational efficiency, and resistance to various forms of attacks. In the experiments conducted, the hybrid encryption algorithm demonstrated improved encryption and decryption times across different file sizes, showcasing its competitive edge over standard AES while ensuring data security against brute force attack testing.
Keywords: Performance, Dynamic AES , Data Shifting

INTRODUCTION

Transformation companies transitioning from traditional to digital operations are facing significant security challenges. Achieving a secure digital transformation necessitates the continuous enhancement of cybersecurity capabilities. Cybersecurity refers to the strategies employed by a country or organization to protect its products and information in cyberspace (Hussain, Mohamed, & Razali, 2020). One key component of cybersecurity is cryptography, which is defined as a method for safeguarding information and communications through the use of codes.

Cryptography encompasses two primary types: symmetric cryptography and asymmetric cryptography. Symmetric cryptography utilizes the same key or secret for both encrypting and decrypting messages. In contrast, asymmetric cryptography employs a different key for encryption than the one used for decryption. In this paper, the researchers provide an in-depth examination of symmetric cryptography and asymmetric cryptography, highlighting various

algorithms related to symmetric cryptography, such as DES, AES, and 3DES, as well as algorithms related to asymmetric cryptography, including RSA, ElGamal, and ECC (Patil & Bansode, 2020).

Additionally, hybrid cryptography emerges as a synthesis of the benefits of both asymmetric and symmetric cryptography, significantly enhancing the overall security level. The proposed hybrid method integrates dynamic shifting with a symmetric algorithm, specifically utilizing a dynamic AES key.

LITERATURE REVIEW

Verma & Dhiman, 2022 demonstrated implementation of improved cryptography algorithm in this study used two blocks of 128 bits in order to test the proposed algorithm, RSA and AES algorithms; the proposed algorithm achieved a lesser time than others.

Adeniyi, Falola, Maashi, Aljebreen, & Bharany, 2022 illustrated secure sensitive data sharing using RSA and ElGamal cryptographic algorithms with hash functions in this research used various text files: 10 kb, 15 kb, 20 kb, 25 kb, 30 kb, 35 kb, 40 kb, 50 kb, and 100 kb to examine both asymmetric algorithms, RSA and ElGamal. The RSA algorithm gets lower time in encryption; however, the ElGamal algorithm achieves a lesser time in decryption.

Ogundoyin, Ogunbiyi, Adebanji, & Okeyode, 2022 showed comparative analysis and performance evaluation of cryptographic algorithms. In this thesis, I input different text files into symmetric and asymmetric algorithms AES, DES, and RSA. The sizes of various files are 50b, 100b, 150b, 200b, and 500b. The RSA gets lower time in terms of encryption and decryption.

Ahmed & Ahmed, 2022 applied comparative analysis of cryptographic algorithms in the context of communication: A systematic review in this scientific paper was used to compare asymmetric and symmetric algorithms through the input of different text files into algorithms. After testing the algorithms, the ALGamal algorithm consumes less time in terms of encryption and decryption.

Mohammad & Abdullah, 2022, demonstrated the enhancement process of AES, a lightweight cryptography algorithm—AES for constrained devices. In this research, we examined the symmetric algorithm AES, LWC-AES, by inputting five multiple text files into algorithms 481b, 1.4kb, 2.82kb, 5.64kb, and 45.1kb. The LWC-AES achieved a lesser time for data encryption and decryption.

Kubba & Hoomod, 2020, illustrated developing a lightweight cryptographic algorithm based on DNA computing. In this paper, the researcher develops a novel algorithm for encryption and decryption data and compares it with another cryptography model through an input message of 128 bits in algorithms; the novel mode gets lower time.

Marqas, Almufti, & Ihsan, 2020 showed comparing symmetric and asymmetric cryptography in message encryption and decryption by using AES and RSA algorithms. This thesis used different tasks in the symmetric and asymmetric algorithms. The tasks represented through the input of various words were 100, 200, 300, and 400 words input into algorithms to determine execution time for all. In the ultimate result, the AES was better than the RSA in terms of encryption and decryption time.

Maqsood, Ahmed, Ali, & Shah, 2017 applied cryptography: a comparative analysis for modern techniques. In this study, the researcher compares asymmetric and symmetric algorithms such as AES, DES, 3DES, RSA, and ALGamal using multiple text files of 32 kb, 126 kb, 200 kb, 246 kb, and 280 kb. Finally, the RSA algorithm achieved a lesser time to encrypt and decrypt files.

Hossain, Hossain, Uddin, & Imtiaz, 2016, demonstrated performance analysis of different cryptography algorithms in this study tested various algorithms through using different text files: 329B, 778B, and 2048B. In the ultimate result, the AES got a lower time to process files. AES was better than DES, RSA, and others.

Applying a Hybrid Encryption Algorithm in Cloud Computing. This study explores and presents an advanced encryption strategy that combines three of the most powerful known algorithms, which are Blowfish, Paillier, and AES, to increase the level of security and privacy during uploading files to the cloud storage. This research aims to provide an overview of how this process can be implemented using these nested hybrid algorithms and the potential benefits of this multi-layered approach. 300b 10,35,1kb 25, 120,10kb 1100, 300, 50kb 3000, 6100 seconds.

METHODOLOGY

The methodology for this research follows a quantitative experimental approach and an applied approach and includes the following steps:

Design and Development

This case demonstrates a flow chart for encryption and decryption as shown in Figure 1 and Figure 2.

Key Expansion for AES Algorithm

This code demonstrates a key expansion process used in AES-like encryption, which expands an initial 128-bit key into 9 subkeys, each of 128 bits (16 bytes). The key expansion is based on the AES algorithm's principles but adapted for a limited number of subkeys.

RotWord

Rotates the word (shifts the bytes) to introduce further randomness into the key expansion.

S-Box (Substitution Box)

A fixed table is used for byte substitution during encryption to provide confusion. The table contains 256 hexadecimal values.

RCON (Round Constant)

A constant is used during key expansion to modify the expanded key values, ensuring cryptographic strength.

Key Expansion

Starting from the initial 128-bit key, it generates subkeys by expanding the key into columns.

Every 4th column is modified by applying the SubWord, RotWord, and XOR with the round constant (RCON).

The key expansion continues until 36 columns are produced (equivalent to 9 subkeys for this case), and these columns are flattened into a byte sequence.

Encryption Process Description

Figure 1 below illustrates the encryption process. The process begins with an XOR operation between the plaintext and a dynamic AES key, producing an initial ciphertext. This ciphertext then undergoes a series of rounds, each utilizing a unique AES key derived from an initial seed. During each round, a dynamic bit shift is applied to the data, where the bits in each byte are shifted to specific positions. This operation further obfuscates the data structure and enhances security. The shift pattern changes with each round, adding a layer of complexity that makes reverse engineering without the correct key sequence extremely challenging. This iterative process of XOR and dynamic bit-shifting continues across multiple rounds until the final ciphertext is generated, ensuring that each encryption step contributes to a fully obfuscated and secure output.

Encryption Process

The encryption process operates on 16-byte data blocks. Here's how the encryption works step by step:

Reading the Data
Python
Copy code
```
With open ("input_data.bin", "RB") as input_file:
   block = input_file.read(16)
   If not block:
      break
```
Purpose*:* Open the input file (input_data.bin) in binary read mode and read 16 bytes at a time.

Input_file.read (16) reads 16 bytes from the file. If the file ends, it returns an empty byte string, causing the loop to exit.

Padding Short Blocks
Python
Copy code
```
if len (block) < 16:
   block = block.ljust (16, b'\x00')
```

Purpose*:* If the block is smaller than 16 bytes (which can happen at the end of a file), pad it with zeros (b'\x00') to ensure it is exactly 16 bytes.

XOR Operation with Keys
Python
Copy code
```
xor_result = block
```

```
for i, key in enumerate (keys, 1):
    xor_result = bytes ([b ^ k for b, k in zip (xor_result, key)])
```

Purpose: This loop applies the XOR operation between the current xor_result (starting with the original block) and each of the 9 keys.

Zip (xor_result, key) pairs each byte of the xor_result with the corresponding byte of the key.
b ^ k performs the XOR operation on each byte. XOR is a reversible operation, meaning it can be undone during decryption.

Bitwise Shift

```
Python
Copy code
shift_amount = random.randint(1, 7)
xor_result = bytes([(b << shift_amount | b >> (8 - shift_amount)) & 0xFF for b in xor_result])
```

Purpose: After applying XOR with each key, the resulting bytes are shifted by a random amount (between 1 and 7 bits).

(b << shift_amount | b >> (8 - shift_amount)): This performs a left bitwise shift on each byte. The shifted-out bits are rotated back to the right.
The result is then masked with **& 0xFF** to ensure that each byte stays within the 8-bit range.

Storing Shift Values

```
Python
Copy code
shift_values.append(shift_amount)
```

Purpose: After each shift, the shift amount is stored in the shift_values list. These values are saved to a separate file (shift_values.txt) so they can be used for decryption.

Writing Encrypted Data

```
Python
Copy code
encrypted_file.write(xor_result)
```

Purpose: The final encrypted block (after XOR and shifting) is written to the output file (encrypted_data.bin).

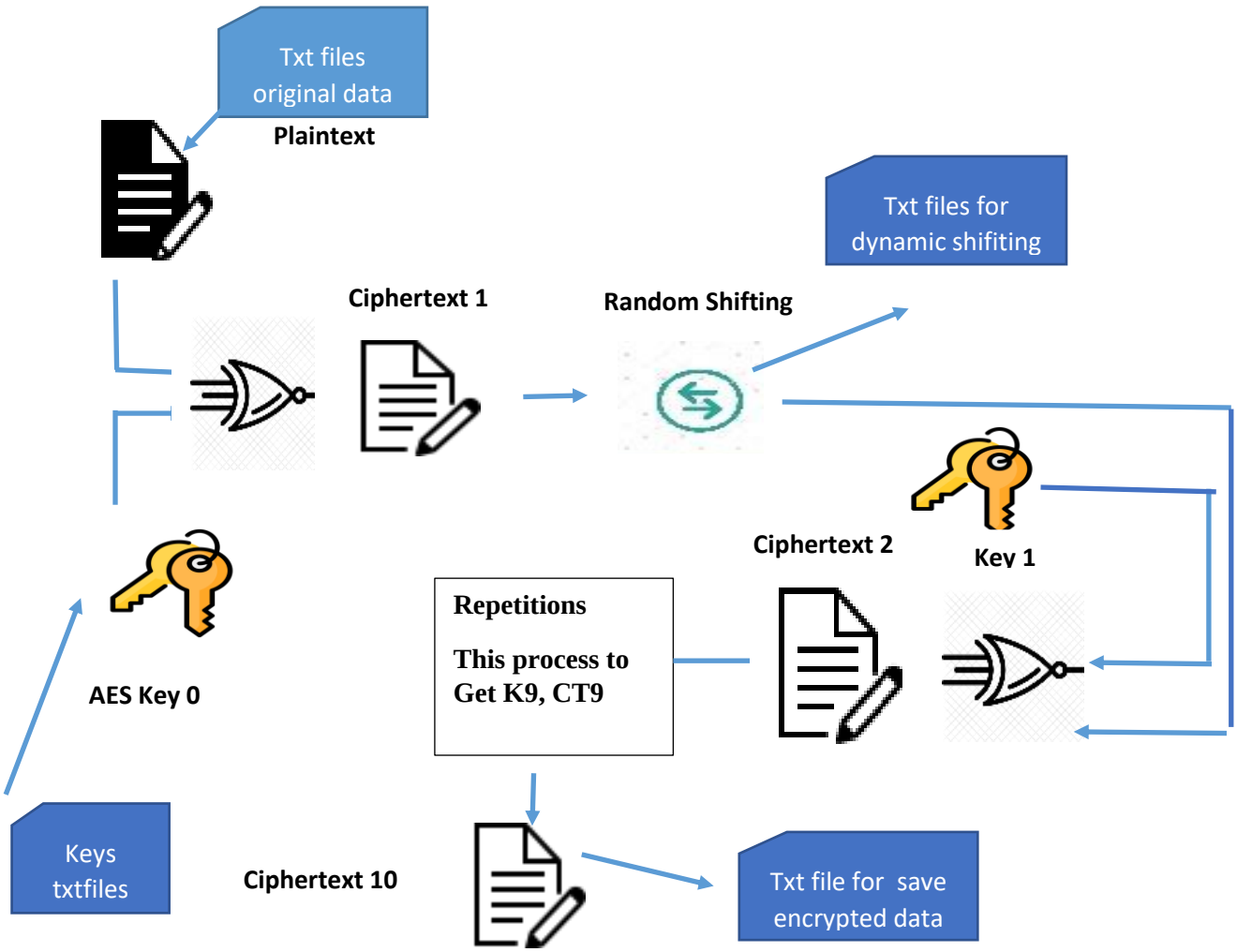

Figure 1 Demonstrates Data Encryption Cases

Decryption Process Description

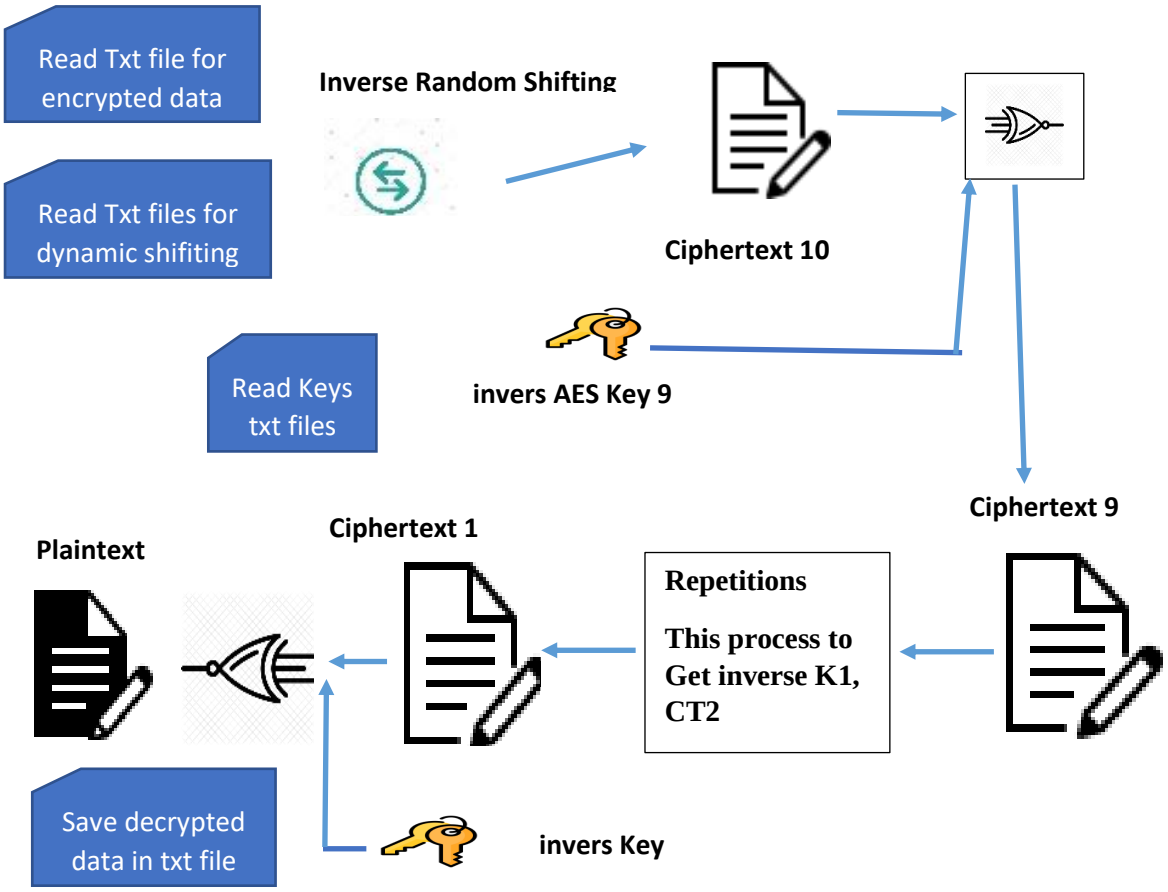

Figure 2 Demonstrates Decryption Data Cases

Figure 2 above illustrates the decryption process, which reverses the encryption steps by applying each AES dynamic key and dynamic data shift operation in the reverse order. Starting with the final ciphertext, the process begins with a reverse data shift to restore the original bit positions in each byte. This is followed by an XOR operation using the corresponding AES dynamic key. This sequence is repeated for each round in reverse, utilizing the original sequence of AES dynamic keys and dynamic shifts to accurately reconstruct the plaintext. Consequently, the original data is successfully recovered.

Decryption Process
The decryption process reverses the operations performed during encryption. Here's how the decryption works:

Reading Encrypted Data

```
Python
Copy code
With open("encrypted_data.bin", "RB") as encrypted_file:
    encrypted_block = encrypted_file.read(16)
    If not encrypted_block:
        Break
```

Purpose: Open the encrypted file (encrypted_data.bin) and read each 16-byte block for decryption.

Reading Shift Values
python
Copy code
shift_values = list(map(int, shift_file.readline().strip().split()))

Purpose: For each block, the corresponding shift values are read from the shift_values.txt file. These values were saved during encryption and will be used to reverse the shift operation during decryption.

Reversing Bitwise Shift
Python
Copy code
decrypted_result = encrypted_block
for i, (key, shift_amount) in enumerate(zip(reversed(keys), reversed(shift_values)), 1):
decrypted_result = bytes([(b >> shift_amount | b << (8 - shift_amount)) & 0xFF for b in decrypted_result])

Purpose: The decryption starts by reversing the bitwise shift. For each key, the shift operation is reversed:

b >> shift_amount | b << (8 - shift_amount): This is a right shift (opposite of the left shift in encryption).
The shift amount used is the one that was saved during encryption.

Reversing XOR with Keys
Python
Copy code
decrypted_result = bytes([b ^ k for b, k in zip(decrypted_result, key)])

Purpose: After reversing the shift, the XOR operation is applied again using the same key (in reverse order). XOR is its inverse, so applying it again will restore the original value of the block before encryption.

Writing Decrypted Data
Python
Copy code
decrypted_file.write(decrypted_result)

Purpose: After decrypting the block, the result is written to the output file (decrypted_output.bin)

Plain text
AHMEDSALEHKHALEDbbfrom republic of yemen1111

Cipher text
×ٹêپ×poCزI!o·d E»³#M‰ *XĪz‡UG²1ïé€ºgط

Decrypted text
AHMEDSALEHKHALEDbbfrom republic of yemen1111

Performance Evaluation

In this case, demonstrates the experimental setup and experimental result for the proposed algorithm.

Experimental Setup

The algorithms are implemented using Python Experiments are performed on a Device nameLAPTOP-CDTI2F87 Processor Intel(R) Core (TM) i7-1065G7 CPU @ 1.30GHz 1.50 GHz Installed RAM8.00 GB (7.77 GB usable) System type 64-bit operating system, x64-based processor. We used different sizes of text files in our experiments.

Experimental Result

Performance Evaluation through computational time of hybrid cryptography algorithm in terms of encryption and decryption time**.**

RESULT

Discussion of Literature Review One and Proposed Technique

A single experiment was used to examine the model through one block of 128-bit input data into algorithms. The purpose was to determine which algorithms achieved the lower time of encryption and decryption.

Objectives

The objective is to inspect the effect of multiple text file sizes used on the algorithm's performance during encryption and decryption.

Detailed Description of the Results

Table 1 below demonstrates carefully used single text files 128bit, ultimately when comparing the results of the (Verma & Dhiman, 2022) algorithm and the proposed algorithm achieved less time to transfer plain text to cipher text and return cipher text to plaintext.

Table 1 Demonstrate Execution Second Time for Proposed Algorithm and Rohit Algorithm

Algorithms	**Execution time Data1(128 bit)**	**Execution time Data2(128 bit)**
Rohit &Jyoti Algorithm	7.3546	9.2943
Proposed Algorithm	0.001994s	0.001994s

Discuss Literature Review Two and the Proposed Algorithm

A single experiment was used to test the algorithm concluded text file 50KB plaintext involvement data into algorithms. The purpose was to detect which algorithms achieved the lower time of encryption and decryption.

Objectives

The goal is to examine the effect of different file sizes on the algorithm's performance in terms of encryption and decryption time.

Detailed Description of the Results

Table 2 and Table 3 below demonstrate carefully used individual text files 50KB, finally when comparing the output of the (Adeniyi et al., 2022) algorithm and the proposed algorithm get the least time to transfer plain text to cipher text and return cipher text to plaintext.

Table 2 illustrates the Encryption Time Millisecond for the Proposed Algorithm and the Adeniyi Algorithm.

S_no	Size (Kb)	RSAEncryption (ms)	ALGamal Encryption (ms)	Proposed Encryption(ms)
1	50	1094	13215	197.3

Table 3 Explain the Decryption Time Millisecond for the Proposed Algorithm and the Adeniyi Algorithm

S_no	Size (Kb)	RSA Decryption (ms)	ALGamal Decryption(ms)	Proposed Decryption(ms)
1	50	23986	4525	158.7

Discuss Literature Review Three and Proposed Algorithm

experiment was used to evaluate the algorithm through various text files 50, 100, 150, 200, and 500 Bytes input data into algorithms. The purpose was to recognize which algorithms achieved the lower time of encryption and decryption.

Objectives

The objective is to investigate the impact of various text file sizes used by the algorithm during execution time.

Detailed Description of the Results

Table 4 and Table 5 below vividly used multiple text files 50, 100, 150, 200, 500Byte, and observed whenever increasing the size of text files the time increase also to encrypt and decrypt data for the Ogundoyin algorithm however proposed algorithm increased time in decryption data, eventually when equating the results of the (Ogundoyin et al., 2022) algorithm and the proposed algorithm achieved lower time to transfer plain text to cipher text and return cipher text to plaintext.

Table 4 Demonstrates the Encryption Time Millisecond Between the Proposed Algorithm and the AES, DES, and RSA Algorithm

Data size Byte	AES	DES Enc	RSA Encryption	Proposed encryption
50	40	38	36	0
100	50	46	47	0
150	60	53	61	0
200	62	56	65	0
500	65	60	69	0

Table 5 Demonstrates the Decryption Time Millisecond Between the Proposed Algorithm and the AES, DES, and RSA Algorithm

Data size Byte	AES	DES Dcr	RSA decryption	Proposed decryption
50	37	35	34	0
100	46	43	44	0
150	57	50	58	0
200	60	52	62	7
500	61	58	65	7

Discussion of Literature Review Four and Proposed Algorithm

experiment was used to examine the algorithm through a set of text files1KB, 2KB, 4KB, 10KB, 20KB, and 40KB input data into algorithms. The purpose was to identify which algorithms achieved the lower time of encryption and decryption.

Objectives

The objective is to explore the effect of text file sizes used on the algorithm's performance during encryption and decryption.

Detailed Description of the Results

Table 6 and Table 7 below show the carefully used several text files 1KB, 2KB, 4KB, 10KB, 20KB, and 40KB, monitor whenever increasing the size of the files the time increase also to encrypt and decrypt data, and ultimately when relating the results of the (Ahmed & Ahmed, 2022). The algorithm and the proposed algorithm achieved less time to transfer plain text to cipher text and return cipher text to plaintext

Table 6 Illustrates Encryption Time Second for Proposed Algorithm and Asymmetric Algorithm

Data size	**RSA encryption(s)**	**ALGamal encryption(s)**	**Proposed encryption (s)**
1KB	0.150	0.120	0.007
2KB	0.292	0.250	0.007
4KB	0.620	0.515	0.016
10KB	1.781	1.452	0.033
20KB	4.355	3.522	0.079
40KB	17.322	11.805	0.157

Table 7 Illustrates Decryption Time Second for the Proposed Algorithm and Asymmetric Algorithm

Data size	**RSA decryption(s)**	**ALGamal decryption(s)**	**Proposed decryption (s)**
1KB	0.128	0.120	0.007
2KB	0.262	0.245	0.007
4KB	0.570	0.509	0.016
10KB	1.671	1.441	0.054
20KB	4.254	3.509	0.058
40KB	22	11.711	0.116

Discussion of Literature Review Five and Proposed Algorithm

experiment was used to examine through many texts files size 481B, 2KB, 5KB, and 45KB input data into algorithms. The purpose was to determine which algorithms achieved the least time for encryption and decryption.

Objectives

The objective is to inspect the effect of multiple text file sizes used on the algorithm's performance during encryption and decryption.

Detailed Description of the Results

Table 8 below illustrates carefully used various text files 481B, 2.82KB, 5.64KB, and 45.1KB, observed whenever increasing the size of files the time increase also to encrypt and decrypt data, ultimately when comparing the results of the (Mohammad & Abdullah, 2022). The algorithm and the proposed algorithm obtained less time to transfer plain text to cipher text and return cipher text to plaintext

Table 8 Explain Encryption Time Millisecond for the Proposed Algorithm and Symmetric Algorithm

Data size	LWC encrypt(ms)	LWC decrypt(ms)	Proposed encrypt(ms)	Proposed decrypt(ms)
481byte	279	260	0	0
2.82KB	257	266	13	16
5.64KB	250	267	25	15
45.1KB	280	291	171	133

Discussion of Literature Review Six and Proposed Algorithm

The individual experiment was used to examine algorithms through one block of 128-bit input data into algorithms. The purpose was to determine which algorithms achieved the lower time of encryption and decryption.

Objectives

The objective is to inspect the effect of several text file sizes used on the algorithm's performance during encryption and decryption.

Detailed Description of the Results

Table 9 below demonstrates carefully used single text files 128bit, finally when comparing the results of the (Kubba & Hoomod, 2020). The algorithm and the proposed algorithm achieved less time to transfer plain text to cipher text and return cipher text to plaintext

Table 9 Demonstrates the Encryption Time Between the Kubba Algorithm and the Proposed Algorithm

Algorithms	Encryption time message (128-bit)
Kubba & Hoomod Algorithm	0.0070
Proposed Algorithm	0.000996

Discussion of Literature Review Seven and Proposed Algorithm

experiment was used to test many text file sizes 100, 200, 400, and 800 words through input data into algorithms. The purpose was to govern which algorithms achieved the least time for encryption and decryption.

Objectives

The objective is to explore the impact of multiple text file sizes used on the algorithm's performance during encryption and decryption.

Detailed Description of the Results

Table 10 and Table 11 below show the wise use of various text files 100, 200, 400, and 800 words, detected whenever increasing the size of files the time increase also to encrypt and decrypt data, ultimately when comparing the results of the (Marqas et al., 2020). The algorithm and the proposed algorithm achieved less time to transfer plain text to cipher text and return cipher text to plaintext

Table 10 illustrates the Encryption Time Between the Proposed Algorithm and AES, the RSA Algorithm

No of words	AES encryption	RSA encryption	Proposed Algorithm encryption
100	0.412	1.843	0.0
200	1.596	3.536	0.007
400	3.222	7.045	0.008
800	4.037	14.167	0.021

Table 11 Explains the Decryption Time Between the Proposed Algorithm and the AES, and RSA Algorithm

No of words	AES decryption	RSA decryption	Proposed Algorithm decryption
100	0.408	1.774	0.008
200	1.163	3.453	0.008
400	2.330	7.106	0.16
800	4.616	13.877	0.025

Discussion of Literature Review Eight and Proposed Algorithm

experiment were used to examine algorithms through two text files 32KB, and 126KB input data into algorithms. The purpose was to regulate which algorithms achieved the lower time of encryption and decryption.

Objectives

The objective is to consider the effect of multiple text file sizes used on the algorithm's performance during encryption and decryption.

Detailed Description of the Results

Table 12 and Table 13 below explain carefully used text files 32KB, and 126KB pragmatic whenever increasing the size of files the time increase also to encrypt and decrypt data, ultimately when comparing the results of the (Maqsood et al., 2017). The algorithm and the proposed algorithm achieved less time to transfer plain text to cipher text and return cipher text to plaintext.

Table 12 illustrates the Encryption Time Between the Proposed Algorithm and the AES, and RSA Algorithm

File size KB	AES encryption	RSA encryption	Proposed Algorithm encryption
32	0.15s	0.13s	0.13s
126	0.46s	0.52s	0.46s

Table 13 illustrates the Decryption Time Between the Proposed Algorithm and the AES, and RSA Algorithm

File size KB	AES decryption	RSA decryption	Proposed Algorithm decryption
32	0.15s	0.15s	0.09s
126	0.44s	0.43s	0.38s

Discussion of Literature Review Nine and Proposed Algorithm

experiment was used to inspect the algorithm through multiple text files329B, 778B, and 2048B input data into algorithms. The purpose was to determine which algorithms achieved the lower time of encryption and decryption.

Objectives

The objective is to investigate the effect of multiple text file sizes used on the algorithm's performance during encryption and decryption.

Detailed Description of the Results

Table14 and Table 15 below vividly used multiple text files329B, 778B, and 2048B, observed whenever increasing the size of files the time increase also to encrypt and decrypt data, ultimately when comparing the results of the (Verma & Dhiman, 2022) algorithm and the proposed algorithm achieved lesser time to transfer plain text to cipher text and return cipher text to plaintext.

Table 14 Explain the Encryption Time Between Proposed Algorithm and AES, RSA Algorithm

File size Bytes	**AES encryption(ms)**	**RSA encryption(ms)**	**Proposed Algorithm encryption(ms)**
329	287	462	7
778	299	541	8
2048	300	488	14

Table 15 Shows the Decryption Time Between the Proposed Algorithm and the AES, and RSA Algorithm

File size Bytes	**AES decryption(ms)**	**RSA decryption(ms)**	**Proposed decryption(ms)**
329	293	499	0
778	304	450	5
2048	297	491	16

Report on the Security of the Encryption Algorithm Against Brute Force Attack

This report evaluates the robustness of the implemented encryption algorithm against brute force attacks as shown in Figure 3 below. The experiment used a Python-based brute force script to test whether the encryption could be broken by systematically trying 1000 random keys of the same length as the encryption key. The results demonstrate the strength of the encryption algorithm, as no valid key was found to decrypt the file successfully.

Encryption algorithms are vital for securing sensitive data. To assess their strength, it is crucial to test their resistance against common attack methods, such as brute force. This method systematically tries possible keys to decrypt a file. In this experiment, we used a hybrid algorithm encryption scheme with a key length of 16 bytes (128 bits) and tested its resistance to brute force attacks.

Test Security for Hybrid Algorithm

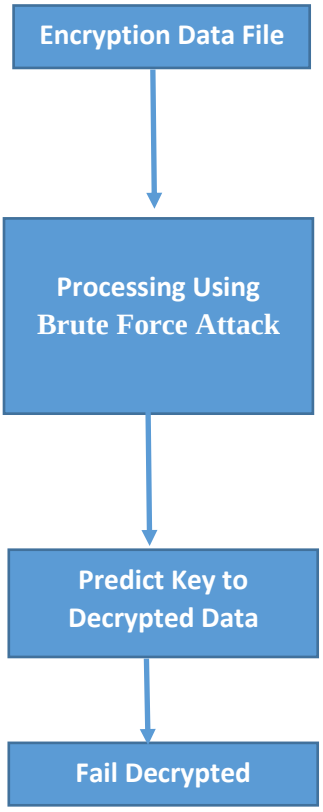

Figure 3 Demonstrates Diagram brute force attack

Encrypted File

The file (encrypted_data.bin) was encrypted using the hybrid algorithm in mode with a random 16-byte key.

Brute Force Script

A Python script was developed to attempt decryption using 1000 random keys.
The script utilized the pycryptodome library for AES operations and implemented the unpadding technique (unpad) to verify the integrity of the decrypted data.

Key Generation

Keys were generated randomly using the os.urandom function, ensuring high randomness and uniqueness for each key.

Decryption Attempt

Each generated key was used to attempt decryption. If the decryption was successful, the plaintext would have been logged alongside the key used.

RESULTS

Decryption Attempts:
A total of 1000 keys were tested against the encrypted file.
No valid key was able to decrypt the file successfully.

Output Analysis

The script verified the integrity of decrypted data using the hybrid algorithm block size and padding rules. All attempts failed to produce a valid plaintext, indicating that the correct key was not in terms of the tested keys.

Table 16: Test Hybrid Algorithm by Brute Force Attack

Files	***Technique of attack***	**Result**
Encrypted file(*encrypted_data.bin)*	Brute force attack	Fail to get the key to decrypt the file

CONCLUSION OF TEST BRUTE FORCE ATTACK

The results confirm the robustness of the encryption algorithm against brute force attacks with the current computational limits. The following key observations were made:
The large keyspace of AES-128 makes brute force attacks computationally infeasible without access to significantly more keys or time.
Proper randomization in key generation significantly enhances the algorithm's strength.
This experiment demonstrates the importance of using strong encryption standards and the difficulty of breaking them using brute force methods alone.

APPENDIX

Key Length Tested: 16 bytes (128 bits)
Encryption Mode: hybrid algorithm
Brute Force Attempts: 1000 keys
Results: No valid key was found.

Discussion

The proposed algorithm achieved more significant results than the symmetric algorithms in terms of execution time, due to the use of dynamic shifting in the proposed algorithm, while the symmetric algorithm relied on fixed, repeated, and complex steps.

CONCLUSION

After comparing the standard symmetric and asymmetric encryption methods (control group) with the hybrid encryption model (experimental group), the proposed model demonstrated notable differences. Specifically, the experimental group exhibited improved security performance and achieved optimal encryption and decryption times, making it particularly effective. This approach significantly enhances both performance and security.

And security without significant computational costs by using dynamic data shifting and a dynamic standard AES random key framework.

.

REFERENCES

1. Hussain, A., Mohamed, A., & Razali, S. (2020). A review on cybersecurity: Challenges & emerging threats. Paper presented at the Proceedings of the 3rd International Conference on networking, information systems & security.

2. Adeniyi, E. A., Falola, P. B., Maashi, M. S., Aljebreen, M., & Bharany, S. (2022). Secure sensitive data sharing using RSA and ElGamal cryptographic algorithms with hash functions. *Information, 13*(10), 442.

3. Ahmed, S., & Ahmed, T. (2022). Comparative analysis of cryptographic algorithms in the context of communication: A systematic review. *International Journal of Scientific and Research Publications, 12*(7), 161-173.

4. Hossain, M. A., Hossain, M. B., Uddin, M. S., & Imtiaz, S. M. (2016). Performance analysis of different cryptography algorithms. *International Journal of Advanced Research in Computer Science and Software Engineering, 6*(3).

5. Kubba, Z. M. J., & Hoomod, H. K. (2020). *Developing a lightweight cryptographic algorithm based on DNA computing.* Paper presented at the AIP Conference Proceedings.

6. Maqsood, F., Ahmed, M., Ali, M. M., & Shah, M. A. (2017). Cryptography: a comparative analysis for modern techniques. *International Journal of Advanced Computer Science and Applications, 8*(6).

7. Marqas, R. B., Almufti, S. M., & Ihsan, R. R. (2020). Comparing Symmetric and Asymmetric cryptography in message encryption and decryption by using AES and RSA algorithms. *Xi'an Jianzhu Keji Daxue Xuebao/Journal of Xi'an University of Architecture & Technology, 12*(3), 3110-3116.

8. Mohammad, H. M., & 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.

9. Mudge, K.(2018).What are the downsides of 128-bit encryption? Retrieved from https://www.quora.com/What-are-the-disadvantages-of-128-bit-encryption

10. Ogundoyin, I., Ogunbiyi, D., Adebanji, S., & Okeyode, Y. (2022). Comparative Analysis and Performance Evaluation of Cryptographic Algorithms. *UNIOSUN Journal of Engineering and Environmental Sciences, 4*(1).

11. Patil, P., & Bansode, R. (2020). Performance evaluation of hybrid cryptography algorithm for secure sharing of text & images. *International Research Journal of Engineering and Technology, 7*(9), 3773-3778.

12. Verma, R., & Dhiman, J. (2022). Implementation of Improved Cryptography Algorithm. *International Journal of Information Technology and Computer Science, 14*(2), 45-53.

AUTHORS BIOGRAPHY

Ahmed Saleh Khaled https://orcid.org/0009-0003-2295-6310, was born in May 1981 in Lahj Republic of Yemen completed secondary education in Taiz 2000 and earned his bachelor's degree in computer science from the University Science of Technology- in 2004- Taiz and completed Master from Arab Academy 2023. He is currently pursuing a Ph.D in Information Technology at the University of Science & Technology, Aden. He is contacted at: *aalhurdi@gmail.com,*

Associate Professor. Dr. Nabil Mohammed Ali Munassar was born in Saudi Arabia in September 1978. He received his B.S. degree in Computer Science from the University of Science & Technology, Hodeida branch, Yemen in 2001; his M.S. degree in Computer Information Systems from The Arabic Academy for Banking and Financial Sciences, Sana'a branch, Yemen, in 2007, and his Ph.D. degree in Computer Sciences from Jawaharlal Nehru Technology University, Hyderabad, India 2014-2015. From 2001 until now, he has been a lecturer at the Faculty of Computers & IT, University of Science & Technology, Yemen. He is the author of more than 20 articles and has many funded research Projects. His research interests include Information Technology, Software Engineering, Databases, System Analysis, and Artificial Intelligence. (nabil_monaser@hotmail.com, n.munassar@ust.edu).

Professor. Mohammed Fadhl Abdullah is currently a professor of computer engineering in the Faculty of Engineering at Aden University in Yemen. He received his master's and PhD degrees in computer engineering from the Indian Institute of Technology, Delhi, India, in 1993, and 1998. He was the editor-in-chief of Aden University Journal of Information Technology (AUJIT). He is a founding member of the International Center for Scientific Research and Studies (ICSRS). His main research interests are in the fields of machine learning, parallel algorithms, and cybersecurity. He can be contacted at email: m.albadwi@ust.edu

Name: Adel Mohammed Ali Al-Qashbari
Academic Degree: PhD in Pure Mathematics
Academic Title: Associate Professor
Tasks:
1) Vice Dean for Graduate Studies and Scientific Research - University of Aden
2) Head of the Mathematics Department, Faculty of Education, Aden - University of Aden
3) Academic Supervisor of the Preparatory Year Program at the Faculty of Engineering - University of Aden
4) Editor-in-Chief of the Journal of Faculties of Education, University of Aden
5) Reviewer of the Researcher Journal of the Scientific Research Association - University of Aden
6) General Coordinator of the Department of Basic Sciences at the University of Science and Technology - Aden
7) Faculty Member at the Faculty of Engineering and Computers at the University of Science and Technology

8) Editorial Board Member of the Book Journal at the University of Book in Iraq
Date of Appointment at the University: 4/21/2002 by Resolution No. (124)
General Specialization: Pure Mathematics
Specialization: Differential Geometry PhD + Special Functions Master's
Current Position: Associate Professor Doctor at the Faculty of Education / Aden - Department of Mathematics / University of Aden
Place and Date of Birth: Aden Governorate 1971/4/24 AD
Address: Aden Governorate / Khormaksar District / Al-Saada District / Next to Aden International Airport
Mobile phone: 733678130 – 700135938 – 776830823 – Home phone: 02-235628
E-mail: Adel_ma71@yahoo.com