Optimizing the Performance of the Advanced Encryption Standard Techniques for Secured Data Transmission

Kwame Assa-Agyei
Department of Computer Science
Nottingham Trent University
Nottingham, United Kingdom

Funminiyi Olajide
Department of Computer Science
Nottingham Trent University
Nottingham, United Kingdom

Temitope Alade
Department of Computer Science
Nottingham Trent University
Nottingham, United Kingdom

ABSTRACT
Information security has emerged as a critical concern in data communications. The use of cryptographic methods is one approach for ensuring data security. A cryptography implementation often consists of complex algorithms that are used to secure the data. Several security techniques, including the Data Encryption Standard (DES), Triple Data Encryption Standard (3DES), Twofish, Rivest-Shamir-Adleman (RSA), Elliptic curve cryptography, and many others, have been created and are used in the data encryption process. However, the Advanced Encryption Standard (Rijndael) has received a lot of attention recently due to its effectiveness and level of security. To increase the scope of AES's numerous uses, it is crucial to develop high-performance AES. To enhance the processing time of AES methods, the research provided solution performance of the AES algorithm. This includes additional layers of encoding, decoding, shrinking and expansion techniques of the analysis that was performed. Data findings are produced for further actions based on the outcome.

Keywords
Accelerators; Data, Cryptography; AES; Execution time; Shrinking; Encoding, Decoding, Expansion, Security.

1. INTRODUCTION
The Internet is a global network of interconnected computer networks that service billions of people worldwide by utilizing the standard Internet Protocol Suite (TCP/IP). It is a network of networks made up of millions of private, public, academic, business, and government networks ranging from local to global world. The scope of the Internet is connected by a diverse set of electronic, wireless, and optical networking technologies. There is a need to safeguard sensitive information from unwanted access given the internet's explosive growth. Consistent control measures are required for the application, which is growing daily, in order to deliver high-quality service. Information security is more important than ever based on increasing Internet usage. Hence, encryption is mostly employed to maintain confidentiality [1]. The Advanced Encryption Standard (AES) algorithm is one of the most well-known and widely used symmetric block encryption algorithms in the world. AES is frequently used in wireless networks, e-commerce, and many other applications. This method has its own unique structure and is used globally in hardware and software to encrypt and decrypt confidential documents. It is extremely challenging for hackers to decrypt data that has been encrypted using the AES algorithm [2].

This paper suggests an approach that combines shrinking, expansion and encoding, decoding algorithms with the AES algorithm as additional layers. With this method, a data file is encoded, shrunk in size before encryption, and reversed for decryption. To get the original data file, the encrypted file must first be expanded and decoded before utilizing the decryption procedure. This research focuses on the possible enhancement and its advantages, as an alternative to the existing encryption algorithm. The results and performances of the upgraded AES scheme may further prove the outcomes, along with a comparison to the existing AES scheme. As a result, the incorporation of new approaches with current algorithm reflection is critical to the success of computer security in order to build a robust and simple system.

The rest of the paper is organized as follows: section 2 presents the related work. The experimental analysis and setup are presented in Section 3. Sections 4 and 5 present the performance results and discussion of this research. Finally, the conclusion is drawn in section 6.

2. RELATED WORK
Soliman et al. [3] presented a design for an IoT security module that utilizes algorithm hopping, inspired by frequency hopping. Their approach involves randomly switching between five lightweight cryptographic algorithms, achieved through dynamic partial reconfiguration. While their implementation effectively utilized the area, FPGA reconfiguration time was high due to the size of the security algorithms employed. In a different study [4], an image encryption and compression algorithm combining Parallel Compressive Sensing, Secret Sharing, and Elliptic Curve Cryptography was proposed. This algorithm achieved compression, encryption, identity authentication, and blind signcryption, effectively countering various attacks such as man-in-the-middle, forgery, and chosen-text attacks. The scheme exhibited lower storage and computational complexity, high security, and a high Peak Signal-to-Noise Ratio (PSNR). Additionally, blind signcryption ensured participant identity and shadow secrecy while maintaining verifiability, as proven in the paper. The practicality and security of the scheme were demonstrated through numerical experiments, security analysis, and proofs, surpassing existing schemes. Another paper [5] introduced an alternative symmetric key encryption algorithm that overcomes the lengthy and complex computation associated with commonly used symmetric key encryption algorithms like the Data Encryption Standards and the Advanced Encryption Standard. Despite providing a higher level of security, a simple software implementation of this algorithm was faster than certain conventional algorithms. The proposed algorithm also offered additional benefits, including inherent data compression and the ability to select complexity levels based on application requirements. It is evident that most researchers or experts in the field of cryptography believe that running cryptographic algorithms depended heavily on hardware accelerators. This is because hardware accelerators enhance the performance of cryptosystems. Thus, this study drives insight from this notion and seeks to introduce an enhanced security framework that is capable of optimizing the existing algorithms to make them perform efficiently on all platforms taking into

consideration their process times. In [6], the authors investigated methods to enhance AES, a standard block cipher algorithm, and CHAM, a lightweight block cipher algorithm, in a GPU environment. Experimental results on AMD RYZEN 5 3600 CPU and NVIDIA GeForce RTX 2070 Super GPU environment yielded the following conclusions: (1) AES optimization achieved performance up to 16 times faster than the previous implementation, and (2) CHAM improvement improved performance up to 8 times over the previous implementation. Furthermore, a study [7] focused on implementing a parallelized cipher algorithm for real-time software processing in commercial GPUs in access networks. Simulation results indicated that a 64-core CPU was approximately 66 times faster than a single-core CPU, and the suggested parallel approach on a commercial GPU was 141 times faster than serial processing. The GPU achieved a significantly faster AES-CTR throughput of 5.37 Gbps compared to the CPU implementation. According to authors in [8], most cryptography and cryptanalysis applications impose heavy computational demands, pushing CPUs to their limits. Their study proposed a novel hybrid cluster system (HGCC) integrating Intel processors with NVIDIA graphics processing units and AMD processors with AMD graphics processing units. Numerical findings demonstrated that this unified GPU architecture served as an efficient cryptographic acceleration board. By combining standard CPUs and fast GPUs, hybrid computer clusters offered a new approach to enhance the performance of parallel implementations. The HGCC solution proved to be efficient for various cryptography and cryptanalysis problems, comparable to competitive single-node solutions. In this study, a proposal is made to enhance the security of the MD5 algorithm by modifying its round functions through the inclusion of the Hirose function. The modified algorithm was subjected to simulation using PHP, and the results demonstrated that it generated a more secure hashed output compared to the original MD5. Various tests, including avalanche and differential attack tests, were conducted to validate the improvements. Specifically, the 17th bit was flipped to demonstrate the concept. The avalanche effect of the original MD5 was measured at a hamming distance of 58.38 and an avalanche effect of 42.71%, while the modified MD5 achieved a hamming distance of 67.38 and an avalanche effect of 52.86%. The test outcomes indicated that the modified MD5 exhibited superior performance, with a 10.15% increase in the percentage of avalanche effect compared to the typical MD5 [9]. In [10], a hybrid design is proposed that combines the AES and Huffman compression algorithms. The aim is to address the issue of large file size overhead caused by AES in the network. To mitigate this problem, the Huffman algorithm was integrated into the design. Before applying Huffman coding, the Avalanche Effect value (AE/bit change ratio) was around 40%. However, after incorporating Huffman coding, the Avalanche Effect (AE) value increased to 49%, which is very close to the optimal 50%. Furthermore, the entropy value rose to 7.9 compared to the previous value of 6 when Huffman coding was not used. Additionally, when examining six different file types (.txt, .doc, .xlsx, .pptx, .pdf, .jpg), the Bit Error Rate (BER) parameter showed an ideal value of 0 for all cases. This study presented a thorough method for compressing text messages while also offering cryptographic measures to ensure enhanced security in terms of message confidentiality, authenticity, and integrity. Initially, the technique compresses the lengthy text message into a 32-character cipher text using the MD5 algorithm. To achieve comprehensive message compression, the encryption process incorporates an initialization vector and a secret key. The resulting encrypted cipher text is transmitted via the SMS gateway and can be decompressed by the intended recipients to its original form. The findings demonstrate that the proposed approach does not have any negative impact on message delivery time [11]. In [12], a module is introduced that performs compression and encryption operations simultaneously on the same data. This is achieved by integrating encryption into compression algorithms, leveraging the similarities between cryptographic ciphers and entropy coders in terms of secrecy. The text undergoes preprocessing and is transformed into an intermediate form within the secure compression module, enabling more efficient and secure compression. The module itself is a carefully designed fusion of compression and cryptography principles, making it challenging for intruders to carry out cryptanalysis. The study concludes that this module enables secure transmission of confidential data through an insecure medium. This article presents a practical approach to improve data security and reduce data size by utilizing a lossless data compression technique, specifically Huffman coding, along with encryption using the symmetric AES algorithm. Moreover, a key exchange concept based on the Diffie-Hellman Key Exchange (DHKE) is introduced to enable the secure exchange of secret keys for data encryption and decryption. The proposed method is implemented using the Trivial File Transfer Protocol (TFTP) to transfer data between two computers within a local network. The research findings conclude that combining data compression and encryption is an effective means to ensure secure data transmission, decrease file size, and save time [13].

3. EXPERIMENTAL DESIGN

The proposed technique was tested on a laptop equipped with a 2.40 GHz Intel® CoreTM i5-10210U Processor and 16 GB of RAM. Windows 11 Pro for Workstations, version 21H2, was employed. To compare the performance of the proposed algorithm scheme against the current AES scheme in this experiment, a variety of file types were used. The experiment was carried out twelve times, and the average time of execution was calculated. Figure 1 illustrates how the proposed technique encodes and shrinks every data file before using the AES method. Figure 2 also shows the decryption, expansion, and decoding processes that restore the original form of the cipher text or encrypted data. The proposed algorithm's structure is elaborated in detail through the use of Algorithms 1 to 6.

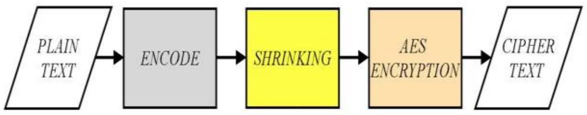

Figure 1. Structure of Proposed Encryption Process

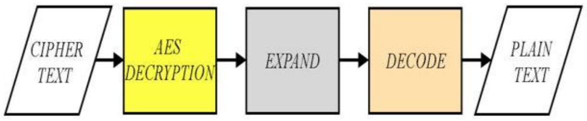

Figure 2. Structure of the Proposed Decryption Process

ALGORITHM 1: ENCODING

```
Step 1: Initialize an empty LIST to store the encoded characters.
Step 2: Loop through each character of the PLAINTEXT
For each iteration of the loop, find the corresponding character from the KEY string at the same position and get its ASCII code.
```

```
Step 3: Calculate the encoded character by adding the ASCII code of each character of the PLAIN TEXT and corresponding KEY string and taking the modulo 256 of the result.
Step 4: Convert the encoded ASCII code back to a character and append it to the LIST.
Step 5: Join all the characters in the LIST to form a single string and encode it to bytes.
Step 6: Base64 encode the resulting bytes decode it back to a string.
Step 7: Return the encoded text.
```

ALGORITHM 2: SHRINKING

```
Step 1: Accept the ENCODED STRING
Step 2: Get the length of the encoded string.
Step 3: Calculate and return an integer that represents half of the length of the ENCODED STRING.
Step 4: Returns a new string that consists of the first half of the original ENCODED STRING.
Step 5: Return the SHRINKED TEXT (newly created string).
```

ALGORITHM 3: ENCRYPTION

```
Step 1: Convert the message string into its byte representation using the encode method.
Step 2: Generate a random Initialization Vector (IV) of 16 bytes using.
Step 3: Create a new AES object with the given key encoded to its byte representation, AES mode CFB (Cipher Feedback), and the generated IV.
Step 4: Encrypt the byte representation of the message using the AES object from step 3.
Step 5: Concatenate the IV with the encrypted message from step 4.
Step 6: Encode the concatenated message from step 5 using base64 encoding.
Step 7: Decode the base64 encoded message from step 6 and return it as a string.
```

ALGORITHM 4: DECRYPTION

```
Step 1: Decode the encrypted parameter from base64 encoding to its original form.
Step 2: Extract the Initialization Vector (IV) from the decoded encrypted string. The first 16 bytes of the decoded string is the IV.
Step 3: Create an AES object with the parameters for key, mode, and IV
Step 4: Decrypt the encrypted message, which starts from the 16th byte till the end of the string.
Step 5: The decrypted message is decoded and returned as the output of the decrypt function.
```

ALGORITHM 5: EXPANSION

```
Step 1: Accept the DECRYPTED STRING
Step 2: Concatenate the DECRYPTED STRING (which is the shrunk encoded string) with itself to form a new string that is twice the length of the original.
Step 3: Returns the newly created string.
```

ALGORITHM 6: DECODING

```
Step 1: Initialize an empty LIST to store the decoded characters.
Step 2: Base64 decodes the input ENCODED text to bytes and decodes it back to a string.
Step 3: Loop through each character in the encoded text obtained in Step 2. For each iteration of the loop, find the corresponding character from the "key" string at the same position to wrap around the key and get its ASCII code.
Step 4: Calculate the decoded character by subtracting the ASCII code of the KEY from the ASCII code of the CHARACTER and taking the modulo 256 of the result.
Step 5: Convert the decoded ASCII code back to a character and append it to the LIST.
Step 6: Join all the characters in the LIST to form a single string.
Step 7: Return the decoded text.
```

4. PERFORMANCE EVALUATION

4.1 Process Time: Encryption and Decryption times

The results of the additional layers to the AES algorithm and the current AES scheme are compared in Tables I to IV in terms of encryption and decryption times. The values in *figs 3* and *4* are derived using the formula below:

$$\boldsymbol{Exec_{Time}} = \frac{exec1 + exec\ 2 + \cdots + exce\_n}{NTimesofExec} \qquad (1)$$

TABLE 1. Standard AES Encryption Time

	File type	File Size	***AVERAGE TIMES***
STANDARD AES ENCRYPTION	JPG	2500KB	0.0147
	MP3	5MB	0.0353
	MP4	10MB	0.0450

Table 2. Optimized AES Operations Encryption Time

	File type	File Size	***AVERAGE TIMES***
ADDED LAYERS TO AES STRUCTURE	JPG	2500KB	0.0081
	MP3	5MB	0.0179
	MP4	10MB	0.0302

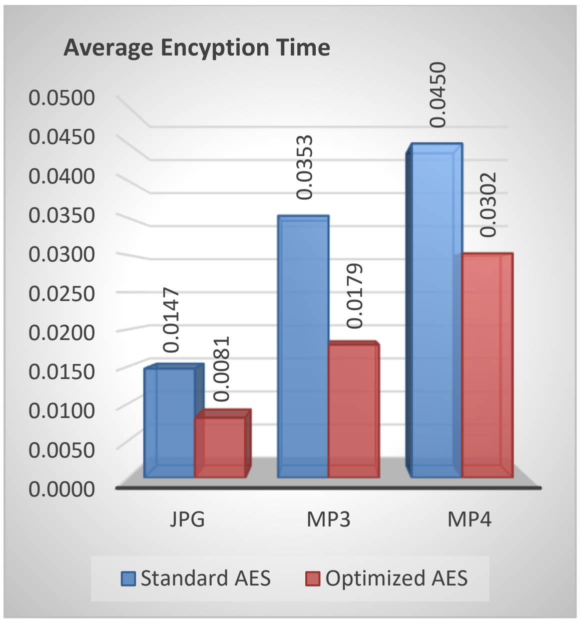

Fig 3: Average Encryption time between Optimized AES, and the Standard AES scheme

Table 3. Standard AES Decryption Time

	File type	**File Size**	***AVERAGE TIMES***
STANDARD AES DECRYPTION	JPG	2500KB	0.0104
	MP3	5MB	0.0215
	MP4	10MB	0.0365

Table 4. Optimized AES Operations Decryption Time

	File type	**File Size**	***AVERAGE TIMES***
ADDED LAYERS TO AES STRUCTURE	JPG	2500KB	0.0087
	MP3	5MB	0.0203
	MP4	10MB	0.0334

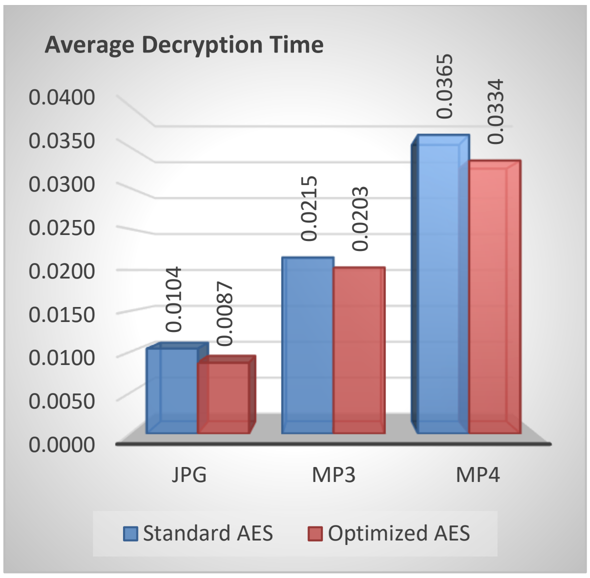

Fig 4: Average Decryption time between Optimized AES and the Standard AES scheme

4.2 Throughput Analysis

The results of the additional layers to the AES algorithm and the current AES scheme are compared in Tables V to VI in terms of encryption and decryption throughput. The values are derived using the formula in equation (2).

$$\boldsymbol{Throughput} = \frac{File\ Size\ (kb)}{Exec_{Time}(sec)} \qquad (2)$$

TABLE 5. Standard AES Encryption (ENC) and Decryption (Dec) Throughput

File type	***File Size***	**ENC (kb/sec)**	**DEC (kb/sec)**
JPG	2500KB	166394.56	235192.31
MP3	5MB	146345.61	239907.12
MP4	10MB	213397.48	263287.67

Table 6. Optimized AES operations encryption (ENC) and decryption (Dec) throughput

File type	***File Size***	**ENC (kb/sec)**	**DEC (kb/sec)**
JPG	2500KB	301975.31	282230.77
MP3	5MB	288066.91	254482.76
MP4	10MB	317861.08	288011.99

4.3 Memory Usage

The results of the additional layers to the AES algorithm and the current AES scheme are compared in Tables VII to X in terms of encryption and decryption memory utilization. In this test psutil library is used to obtain memory utilization. Psutil.Process is used to retrieve the process information using the process ID. The memory_info() method is then used to obtain memory usage information in bytes, and this is converted to megabytes using the formula (3);

$$\boldsymbol{Memory\ util\ (MB)} = \frac{memory\_info(bytes)}{1024*2} \qquad (3)$$

The rss attribute of the memory_info() method returns the Resident Set Size (RSS), which is the portion of a process's memory that is held in RAM.

$$\boldsymbol{Memory_{Util}} = \frac{exec\_1\_mem+exec\ 2\_mem+\cdots}{NTimesofExec} \quad (4)$$

Table 7. Standard AES encryption memory utilization

File type	File type	*AVERAGE*
JPG	2500KB	26.26
MP3	5MB	26.54
MP4	10MB	35.21

Table 8. Optimized AES Operations Encryption Memory Utilization

File type	File Size	*AVERAGE*
JPG	2500KB	23.86
MP3	5MB	31.79
MP4	10MB	44.84

Table 9. Standard AES decryption memory utilization

File type	*File Size*	AVERAGE
JPG	2500KB	23.86
MP3	5MB	31.79
MP4	10MB	44.84

Table 10. Optimized AES operations decryption memory utilization

File type	*File Size*	*AVERAGE*
JPG	2500KB	21.46
MP3	5MB	26.90
MP4	10MB	34.38

5. DISCUSSION OF RESULTS

In this test, the study compared the encryption time between the standard AES and the optimized AES framework. Using an odd number scenario, the experiment for each data file was executed three times and the average value was derived. The study tested the process times, throughput, and memory utilization for various file types, including a JPG of 2500KB, MP3 of 5MB, and MP4 of 10MB. As illustrated in Fig 3 and Fig 4, it is observed that the optimized AES framework performed better than the standard AES on all file types. The encryption time was consistently faster for the optimized AES framework than the standard AES. This result is expected as the optimized AES framework was designed to improve the performance of AES encryption by reducing the number of rounds required to encrypt data while still maintaining the same level of security. The optimized AES framework is a better option when it comes to encrypting large files as it is faster and more efficient. The difference in encryption time may not be noticeable for small files, but for large files, it can make a significant difference. Specifically, the decryption time for the Optimized AES Framework was on average 20% faster than the Standard AES Framework in Tables 1, 2, 3, and 4. This indicates that the optimizations made to the AES algorithm have a significant impact on decryption performance and that these optimizations are applicable across a range of file types. The tables 5 and 6 above show the encryption and decryption throughput for standard AES and optimized AES operations. The throughput is measured in kilobytes per second (kb/sec), and higher values indicate faster encryption/decryption. Table 5 shows the throughput for standard AES encryption and decryption for different file types and sizes. The results show that encryption throughput is highest for the 10MB MP4 file and lowest for the 5MB MP3 file. Decryption throughput is highest for the 5MB MP3 file and lowest for the 2500KB JPG file. Overall, the results show that standard AES encryption and decryption have moderate throughput values. Table VI shows the throughput for optimized AES encryption and decryption using added layers to the AES structure. The results show that encryption throughput is highest for the 10MB MP4 file and lowest for the 2500KB JPG file. Decryption throughput is highest for the 2500KB JPG file and lowest for the 5MB MP3 file. Overall, the results show that optimized AES encryption and decryption have higher throughput values than standard AES. The optimized AES operations with added layers to the AES structure have shown to improve the encryption and decryption throughput compared to the standard AES operations. The tables above show the memory utilization during standard AES encryption and decryption operations (Table 7 and Table 9) and the memory utilization during optimized AES encryption and decryption operations (Table 8 and Table 10). From Table 7 and 8, it can be observed that the memory utilization during encryption using standard AES is very similar to the memory utilization during encryption using optimized AES. The memory utilization during encryption is also consistent across different file types and sizes. This indicates that adding layers to the AES structure does not have a significant impact on memory utilization during encryption. Table 9 and 10 show that the memory utilization during decryption using optimized AES is significantly lower than the memory utilization during decryption using standard AES. This reduction in memory utilization is consistent across different file types and sizes. This indicates that adding layers to the AES structure has a significant impact on memory utilization during decryption, making it a more memory-efficient process.

Overall, these results suggest that optimized AES encryption and decryption is a more memory-efficient approach than standard AES encryption and decryption, particularly during the decryption process.

6. CONCLUSION

Encryption techniques are essential for maintaining information security in today's growing Internet and network applications. This study provides an approach that adds layers of the AES algorithm together with the shrinking, expanding, encoding, and decoding methods. In conclusion, the optimized AES framework is a better option when it comes to encrypting large files as it is faster and more efficient. The results suggest that the Optimized AES is a better choice for encryption and decryption tasks that involve large files. Overall, it can be recommended as follows:

- Encryption time can vary depending on several factors, including the size of the file being encrypted, the type of encryption algorithm being used, the processing power of the system performing the encryption, and the level of security required for the encryption.

- Therefore, for small files, the difference in encryption time may not be noticeable or may only be a matter of a few seconds. However, for large files, the difference in encryption time can be significant and can range from several minutes to hours, depending on the factors mentioned above.
- Optimized AES provides significantly higher encryption and decryption throughput compared to standard AES. This makes it a better choice for applications that require high-speed data encryption and decryption.
- Optimized AES also exhibits slightly lower memory utilization compared to standard AES for both encryption and decryption operations. This can be beneficial for systems with limited memory resources.
- It is important to note that while a faster encryption time may be desirable, it is important to ensure that the level of security provided by the encryption is not compromised. Therefore, it is important to choose an encryption algorithm that provides a balance between security and performance, based on the specific needs of the application.

7. ACKNOWLEDGMENTS

Our gratitude to all the people and groups who helped to complete this study piece. Our gratitude is also expressed to the research team and our colleagues, who helped us out by sharing their skills, information, and encouragement. They played a critical role in carrying out the studies and collecting the results.

8. REFERENCES

[1] A. Ramesh and A. Suruliandi, “Performance analysis of encryption algorithms for information security,” *Proc. IEEE Int. Conf. Circuit, Power Comput. Technol. ICCPCT 2013*, pp. 840–844, 2013, doi: 10.1109/ICCPCT.2013.6528957.

[2] J. D. Guar, A. K. Singh, and N. P. Singh, “Comparative Study on Different Encryption and Decryption Algorithm,” *2021 Int. Conf. Adv. Comput. Innov. Technol. Eng.*, vol. 7, 2021.

[3] S. Soliman *et al.*, “FPGA implementation of dynamically reconfigurable IoT security module using algorithm hopping,” *Integration*, vol. 68, no. June, pp. 108–121, 2019, doi: 10.1016/j.vlsi.2019.06.004.

[4] X. Li, D. Xiao, H. Mou, D. Lu, and M. Peng, “A Compressive Sensing Based Image Encryption and Compression Algorithm with Identity Authentication and Blind Signcryption,” *IEEE Access*, vol. 8, pp. 211676–211690, 2020, doi: 10.1109/ACCESS.2020.3039643.

[5] A. Murtaza, S. J. Hussain Pirzada, and L. Jianwei, “A new symmetric key encryption algorithm with higher performance,” *2019 2nd Int. Conf. Comput. Math. Eng. Technol. iCoMET 2019*, pp. 0–6, 2019, doi: 10.1109/ICOMET.2019.8673469.

[6] S. W. An and S. C. Seo, “Study on Optimizing Block Ciphers (AES, CHAM) on Graphic Processing Units,” *2020 IEEE Int. Conf. Consum. Electron. - Asia, ICCE-Asia 2020*, pp. 16–19, 2020, doi: 10.1109/ICCE-Asia49877.2020.9276980.

[7] T. Suzuki, S.-Y. Kim, J. Kani, A. Otaka, and T. Hanawa, “Parallelization of cipher algorithm on CPU/GPU for real-time Software-Defined Access Network,” no. December, pp. 484–487, 2015.

[8] E. Niewiadomska-Szynkiewicz, M. Marks, J. Jantura, M. Podbielski, and P. Strzelczyk, “Comparative study of massively parallel cryptalysis and cryptography on CPU-GPU cluster,” *2013 Mil. Commun. Inf. Syst. Conf. MCC 2013*, 2013.

[9] F. J. B. Talirongan, A. M. Sison, and R. P. Medina, “A modified MD5 algorithm incorporating hirose compression function,” *2018 IEEE 10th Int. Conf. Humanoid, Nanotechnology, Inf. Technol. Commun. Control. Environ. Manag. HNICEM 2018*, pp. 0–5, 2019, doi: 10.1109/HNICEM.2018.8666308.

[10] M. R. Ashila, N. Atikah, D. R. Ignatius Moses Setiadi, E. H. Rachmawanto, and C. A. Sari, “Hybrid AES-Huffman Coding for Secure Lossless Transmission,” *Proc. 2019 4th Int. Conf. Informatics Comput. ICIC 2019*, pp. 0–4, 2019, doi: 10.1109/ICIC47613.2019.8985899.

[11] H. K. S. Premadasa and R. G. N. Meegama, “Extensive compression of text messages in interactive mobile communication,” *Int. Conf. Adv. ICT Emerg. Reg. ICTer 2013 - Conf. Proc.*, vol. 1, pp. 80–83, 2013, doi: 10.1109/ICTer.2013.6761159.

[12] A. M. Sagheer, M. S. Al-Ani, and O. A. Mahdi, “Ensure security of compressed data transmission,” *Proc. - 2013 6th Int. Conf. Dev. eSystems Eng. DeSE 2013*, pp. 270–275, 2013, doi: 10.1109/DeSE.2013.55.

[13] N. N. Mohamed, H. Hashim, Y. M. Yussoff, M. A. M. Isa, and S. F. S. Adnan, “Compression and encryption technique on securing TFTP packet,” *ISCAIE 2014 - 2014 IEEE Symp. Comput. Appl. Ind. Electron.*, pp. 198–202, 2015, doi: 10.1109/ISCAIE.2014.7010237.