International Journal of Advanced Science and Technology
Vol. 29, No. 5, (2020), pp. 12520-12533

An Enhanced Encryption System Based on the Unison of Lossless Compression and a Tri-Hashing Algorithm

Adenle Samuel Adimu[1], Adewale Opeoluwa Ogunde[2,*], Bosede Oyenike Oguntunde[3], Mba Obasi Odim[4], Samson Afolabi Arekete[5]

[1,2,3,4,5]*Department of Computer Science, Redeemer's University, Nigeria*
[1]*adenles@run.edu.ng, ogundea@run.edu.ng[2,]*, oguntunden@run.edu.ng[3], odimm@run.edu.ng[4], areketes@run.edu.ng[5]*

Abstract

In recent times, technological advancement in communications has made it essential to protect the ever-increasing amount of data and ensure the privacy of users. There is a lot of sensitive data present on the web, and it needs to be protected. Using conventional security methods like RSA, DES, AES algorithms, and others alone have become inadequate in protecting data from any kind of potential abuse. Complex algorithms are required to not only to encrypt the data but also to compress and distribute the data. This is due to the fact that some existing encryption techniques can be cracked given enough time and resources. This work attempts to resolve these flaws by combining a lossless compression technique with encryption in a single process or selective encryption of data. The proposed algorithm, tagged Enhanced Encryption System (EES), uses three different keys. One hashed from PBKDF2, the other from the PI sequence and finally DNA sequence, where each one is invariant of the user's inputted key. Following this operation, there are infinite possible keys generated under every iteration. Lossless compression was applied to the test data based on the Lempel-Ziv-Welch (LZW) algorithm. Data encryption was implemented with the python programming language. The output produced different ciphertexts for the same plain text, thereby confusing any hacker that tries to brute-force the system. The experimental results showed that EES achieved better encryption and decryption run-time without losing data when compared to the AES, RSA and DES algorithms.

Keywords: *Ciphertext, Compression, Cryptography, Decryption, DNA, Encryption.*

1. Introduction

Digital communication over networks and different communication platforms are the order of the day in today's world. This mode of communication, though fast and effective, has opened many vulnerabilities to malicious people for perpetrating their evil intentions. Quite a number of stand-alone solutions had been proposed in the literature, but none is found to be perfect as these hackers always find a way around them most times. There is, therefore, an urgent need to come up with combined security models with very high potentials of providing a lasting solution to these problems. This research is targeted at improving the security of our communication systems by creating a tri-hashing algorithm that uses three different keys: one hashed from PBKDF2, the other from the PI sequence, and finally from DNA sequence to have stronger and better security characterised by lossless data compression technique and better encryption. The powers of the three methods are combined here to have a stronger security. The concept of data compression is about the minimisation of data representation by coding it. This process is tagged lossless compression if the master copy can be fully regenerated from the derived version. Whether a compression will be lossy or not is much determined by the kind, source and entropy of data to be compressed [1].

Huffman's lossless compression, which is the compression technique used in this work, is usually associated with no data loss [2]. Huffman coding and Shannon-Fano method for text compression are based on a similar algorithm which is based on variable-length encoding algorithms. Additionally, both techniques use a prefix code-based approach on a set of symbols along with their probabilities [2]. This work, therefore, aims at producing a single enhanced encryption algorithm, tagged EES, implemented with the adoption of Huffman's compression technique and the tri-hashing algorithm, generating keys from PBKDF2, PI and DNA sequence. The remaining section is organised as follows. Section two provides a brief review of literature tied to this work. The proposed architectural model detailing the process of combining the Huffman's compression and the tri-hashing algorithms was presented in section three. Section four describes the implementation details of the work. Section five provides a conclusion of the work and gave valuable recommendations and directions for future work.

2. Related Works

This section provides a review of some research efforts in the research area.

2.1. DNA-Based Cryptographic Systems

There have been significant research efforts that have resulted in an improved cryptographic system; nevertheless, recent studies are being directed towards Deoxyribose Nucleic Acid (DNA) Cryptography [3]. The authors in [4] presented a review of DNA encryption algorithms in collaboration with standard classical encryption algorithms. The study showed that incorporating DNA technology into symmetric-key cryptography methods became resilient against attacks. The result was in agreement with a theoretical analysis that has shown that including a biological element in computing has tremendously improved cryptosystems in addition to enhanced computing power. Narasingapuram and Ponnavaikko in [5] proposed a DNA cryptography-based user-level security for cloud computing and applications. The algorithm transforms each character of the input data to its ASCII form, generates its binary equivalent, which is used to generate the DNA sequence to form the strong key PK for encryption. The results were compared with the existing Asymmetric DNA algorithm discussed in [6] showed that the proposed system outperformed other cryptographic techniques in terms of time and computational complexity.

A method for implementing data hiding based on DNA sequence was proposed in [3]. The system employed the concept of binary coding and random values, and the sample DNA sequence was utilised to transform the message. Both the DNA sequence and transformed data are delivered to the recipient, which would then be extracted to get the original message. A common DNA sequence shared by both sender and receiver is used for encryption and decryption processes. The findings from the method showed that it was resistant against certain attacks, and it ensured data integrity and confidentiality over the transmission channel. The study in [7] presented a DNA cryptographic technique that uses the Symmetric Key Exchange to protect data travelling through an insecure channel. The technique uses XOR operation with a one-time-pad DNA sequence as the encryption technique. The method proves to be efficient in encrypting, transmitting and preventing attacks as it is difficult for intruders to generate a crack key. Cryptography and bioinformatics techniques were combined to secure information transmission over insecure channels in Siddaramappa and Ramesh [8]. The authors proposed a method of deoxyribonucleic (DNA) and RNA as key generated for secure data encryption and decryption methods over a communication channel. This method makes it harder for the cryptanalyst to break it as there are nearly billions of DNA sequences.

[9] proposed an Enhanced Encryption Algorithm based on Prime number, DNA sequence and PI sequence, which is used to encrypt the data in a more secure and complex manner. The PI and DNA sequences are used to generate two different keys to produce the ciphertext. This work, however, adopts the Huffman lossless compression technique to compress the file before encryption and uses the tri-hashing technique, which is a combination of PBKDF2, the PI and DNA sequences for the encryption.

2.2. Other Cryptographic Systems

TACIT, a symmetric key encryption technique for secure routing, was proposed in [10]. It adopted an independent approach with an appropriate mathematical model which was assumed to be computationally secured. The key distribution system was applied on a secure policy-based routing, and it was limited to conversion of a text file. Aswin and Aswathy in [11] proposed A-S Algorithm which takes plain text with the key from the user, converts each letter of plaint text and key into ASCII code and then performs XOR operation between them. The obtained code is converted back to equivalent characters which are ciphertext. The algorithm is simple and efficient; however, it is easy to crack. [12] developed an Enhanced Encryption Algorithm which is a symmetric algorithm. The algorithm generates two keys using the Key Generation Service. It initially converts the plain text into binary blocks and divides them based on a random prime number generated based on their length and rotates the bits with the help of the first key. Then the resultant of XOR operation performed between the second key and the rotated bits merged in sequential order eventually yields the ciphertext. This algorithm makes the encryption process very complex and difficult to break. [13] solves the problem of existing techniques of content-based double encryption algorithm using symmetric key cryptography. It uses binary addition, folding method and logical XOR operation to encrypt the plain text and the key to produce cipher text. The secret key is encrypted and shared through a secure network; the enhanced algorithm gives superior performance compared to existing encrypted algorithms.

A double layer encryption method to ensure security in the cloud was presented in [14]. It is based on the RSA cryptography algorithm. This scheme resolves key escrow difficulty and data expose problem by RSA algorithm of the public key cryptography approach. In this proposed double-layer encryption schemes, the data will be extremely secured while protecting and sharing in the cloud environment. This scheme not only makes full use of the great processing skill of cloud computing but also can efficiently ensure cloud data privacy and security. [15] assessed the performance of some traditional and modern cryptographical techniques based on block size, key length, encryption time, among other criteria. RSA has the longest encryption time with approximately 490ms, and blowfish with the least encryption time of approximately 290ms, RSA and MD-5 has the largest block size of 512 bits while DES and blowfish have the least block size of 56 bits. In [16], three encryption algorithms, AES, DES and RSA were implemented, and their performances were compared based on their simulated time at encryption and decryption. The findings showed that the AES algorithm consumed the least time of encryption while RSA consumed longest encryption time. The decryption of the AES algorithm outperformed other algorithms. From the simulation result, it was found that the AES algorithm offers superior performance compared to DES and RSA algorithm.

Studies in [3]. [4], [5], [6], among others, have shown that the cryptographic system that incorporated DNA scheme had demonstrated some improved performance over the conventional approaches. In the current study, we proposed an enhanced encryption method that uses three different keys, one hashed from PBKDF2, the other from PI sequence and finally DNA sequence, where each one is invariant of the user's inputted key.

3. Methodology

This study combines the Huffman's lossless compression technique and encryption in a single indivisible process for data encryption. The block diagram of the system is depicted in Figure 1. The enhancement is provided by the combination of the PI sequence and DNA sequence, which will act as salt in a tri-hashing algorithm generated by the password-based key derivation function to (PBKDF2). The input file is passed to the Huffman's compression session, whose output is worked upon by the ESS core to generate the output file.

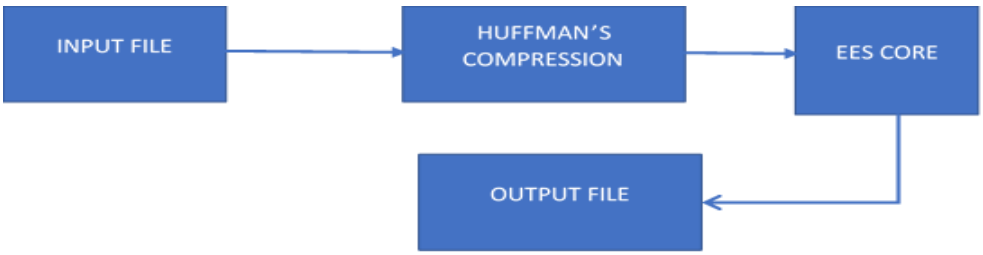

Figure 1. System's block diagram

3.1. Tri-hashing

The flowchart and aarchitectural design of the EES system is presented in Figure 2.

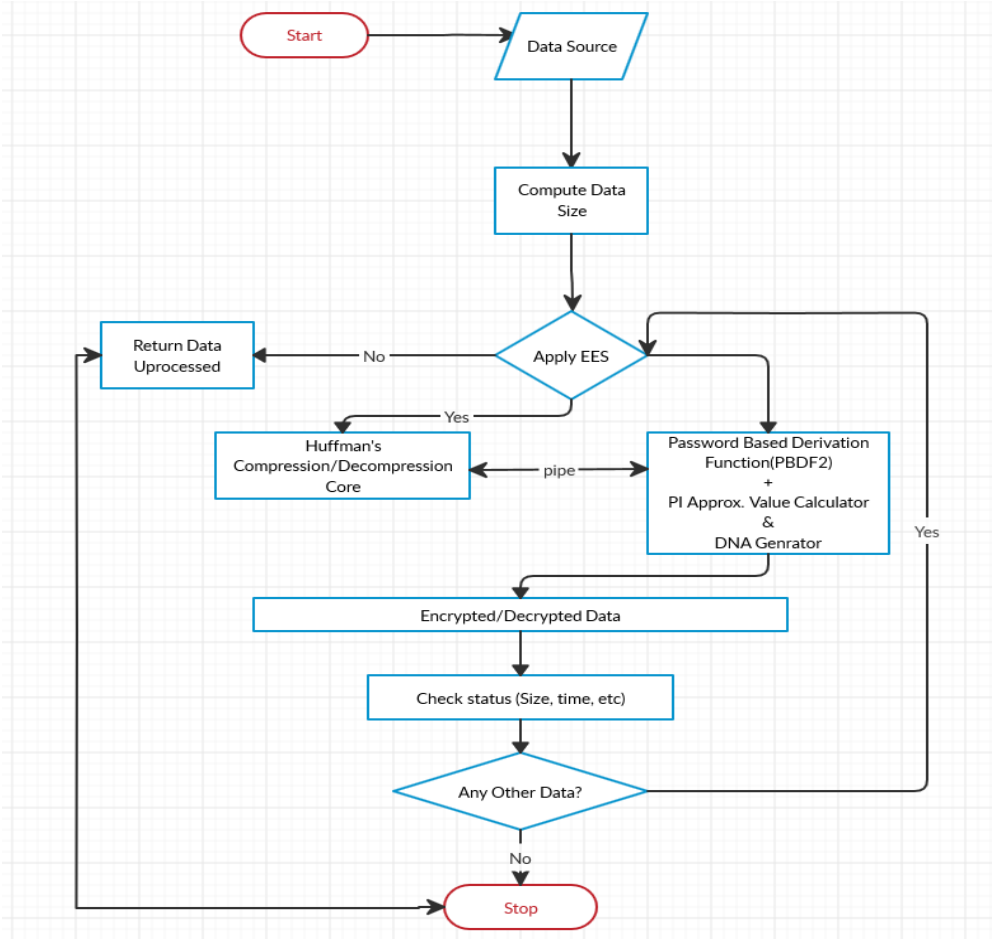

Figure 2. Flowchart and Architectural design of the EES system

The proposed method presents an algorithm which generates keys from the PBKDF2, while the PI value and the DNA sequence act as salt to make sure the encryption is unbreakable while generating a unique non-repeating key. The process is initiated by the START and data obtained from Data source. The data size is computed, and the EES applied to the data. This involves the first stage of compression/decompression and then encryption. This returns unprocessed if EES is not applied to the data.

3.2. Compression and Decompression Core

Huffman's compression/decompression is applied to the data. This ensures that there would be no loss of data. The original copy of the file can be completely recovered when the file is decrypted. A source symbol like a character is coded based on the variable-length coding scheme. The code is computed on the likelihood of occurrence of a value. [17]. There are two main parts involved: (1) create a Huffman tree and (2) traverse the tree to find codes [18].

Algorithm

huffmanCoding(string)
Input: A chatcter_string
Output: generated charater codes

Start:
specify a Huffman tree node with the following parts: character(c), frequency(freq), left(lc) and right(rc) child of the nodes.
Initialise a character frequency (cfreq) to 0
//and generate a list, "freq" for storing them.
cfreq = 0
for each c
cfreq = cfreq+1
end of for-loop

for all character ch
if freq ch ≠ 0 then
priority_queue_Q = ch + freq
end of for-loop
while Q ≠ Φ
take away item from Q and assign it to lc
take away item from Q and assign to the rc
visit each node to find the assigned code
end of while-loop
End

traverseNode(n: node, code)
Input: Huffman tree node, and the previously assigned code
Output: each assigned character code

if a node's lc n ≠φ then
traverseNode(lc(n), code+'0') // traverses the left_child
traverseNode(rc(n), code+'1') // traverses the right_child
else
show the character and data of current node.

The complexity of code assignment based on frequency is *O(n log n)* [18].

3.3. The Password-Based Derivation Function (PBKDF2)

A pseudo-random function is applied in PBKDF2 to the input password or passphrase, like hash-based message authentication code (HMAC), together with a salt value, and the process is repeated several times to generate a derived key. This, in turn, provides a 64-byte key from PBKDF2 with SHA-256 within 100,000 iterations which can then be used as a cryptographic key in succeeding procedures.

There are five input parameters for the PBKDF2 key derivation [19]:

$$DK = PBKDF2(PRF, Password, Salt, c, dkLen) \quad (1)$$

where PRF denotes a pseudo-random function of two parameters with output length hLen (e.g., a keyed HMAC). Password denotes the principal password for generating a derived key. Salt denotes an ordered collection of bits; c specifies the number of repetitions. dkLen denotes the bit-length of the generated key. DK is the generated key.

We calculate each *hLen*-bit block *Ti* of *DK*, as ("+" denote string concatenation):

$$DK = T1 + T2 + ... + T_{dklen/hlen} \quad (2)$$
$$Ti = F(Password, Salt, c, i) \quad (3)$$

The function *F* denotes the *XOR (^)* of *c* iterations of chained PRFs. *PRF, at the first iteration,* uses *a password* as the *PRF* key and *salt* concatenated with *i* encoded as a big-endian 32-bit integer as the input. (*i* is a 1-based index.) Succeeding iterations of PRF use *password* as the PRF key and the output of the previous PRF computation as the input:

$$F(Password, Salt, c, i) = N1 \wedge N2 \wedge ... \wedge Nc \quad (4)$$

where:

$$N1 = PRF(Password, Salt + INT\_32\_BE(i)) \quad (5)$$
$$N2 = PRF(Password, N1) \quad (6)$$
$$...$$
$$Nc = PRF(Password, Nc\text{-}1) \quad (7)$$

3.4 PI approx. Value and DNA Sequence Generator

The cryptographic generator works on the generation of random keys from the DNA and PI values within the script, DNA hashes are embedded during either the encryption and decryption phases, translated by XORs into a RAW format that is used in the processes defined. Then, it sends the generated keys ready to be hashed by the encryption core, upon completion, the compressed data is filtered to remove unnecessary data accumulated from the encryption, the data is finally sent as output for transfer to the destination, a stand-alone executor is also redefined within it, hence, while being an importable library, it is also inherently an executable script. After defining the input file for reading, the streams are piped into transformative middleware with the help of the `node-io` library built for this purpose and after that piped out to a writable stream.

3.5 Encrypted/Decrypted of Data

The processed data are stored here shortly before the status is checked; here, processed data can be extracted. Check status (size, time, etc.): various parameters are checked and documented.

3.6 Any other Data

At this stage, the algorithm checks for any other data on queue ready to undergo the same process. The data is finally sent as output for transfer to the destination folder.

3.7 Stop

This ends the process of EES.

4. Implementation and Results

This section covers the data description, experimental procedures and results.

4.1. Data description

Data samples of various types and sizes used for the experimentation was collected from the web and divided into different sizes, as shown in Table 1. The performance metric used is the time spent for compressing and encrypting the data samples, which comprises of various data types. Various data types were used to test the flexibility of the algorithm to work well on different data types. The experiment was repeated eleven times, and the values were recorded and later represented graphically.

Table 1. Different sizes of Data samples

Data Sample	Sizes (KB)	Data Sample Type
SAMPLE1	62.95	PNG
SAMPLE2	110.93	PNG
SAMPLE3	153	TXT
SAMPLE4	196	TXT
SAMPLE5	312	TXT
SAMPLE6	824.21	PPT
SAMPLE7	868	TXT
SAMPLE8	875.07	PPTX
SAMPLE9	1040	PDF
SAMPLE10	3110	PDF
SAMPLE1	5660	EXE

4.2. System Implementation

The implementation process and details are described in this section. Patterns that could result in cryptanalysis are usually removed by lossless compression to increase the unicity distance. Deflate, a lossless data compression file format, was adopted in the implementation. Deflate combines Huffman coding and LZSS [20]. It was used in compressing any data types in the EES implementation. Password-Based Key Derivation Function 2 (PBKDF2) typically derive a cryptographic key from the PI and DNA sequences, also used for key storage, aids in the creating an encryption key from a defined password, and where it is not possible to reverse the password from the hashed value. DNA Computing secures data using the biological structure of DNA A-T and G-C combinations which are more efficient in terms of space complexity. PI Sequence has an

infinite sequence of digits, due to its statistical randomness, it is used in encryption techniques for key generation in the EES core.

The EES algorithm was implemented and tested in Microsoft windows environment. The system was implemented with the NodeJS for the integration with several dependencies, packages of the existing lossless compression. Phython3 was adopted for the encryption process. Software needed to implement the EES algorithm must be downloaded from the libeaes with the installation procedure highlighted below.

On windows,
Run
> npm install libeaes -g
Open open powershell ([[shift] + <right click>] > 'Open Powershell window here')
Then
To encrypt:
> libeaes encrypt <input_file> <output.enc>
To decrypt:
> libeaes decrypt <output.enc> <input_file>

The first interface that is displayed by the application is the database verification interface, which helps to load the sample data into the database. The EES interface contains the compression and encryption interface. It shows the run-time, input file, output file, input size, output size and deflation. It also contains the compression and encryption interface, shown in Figure 3.

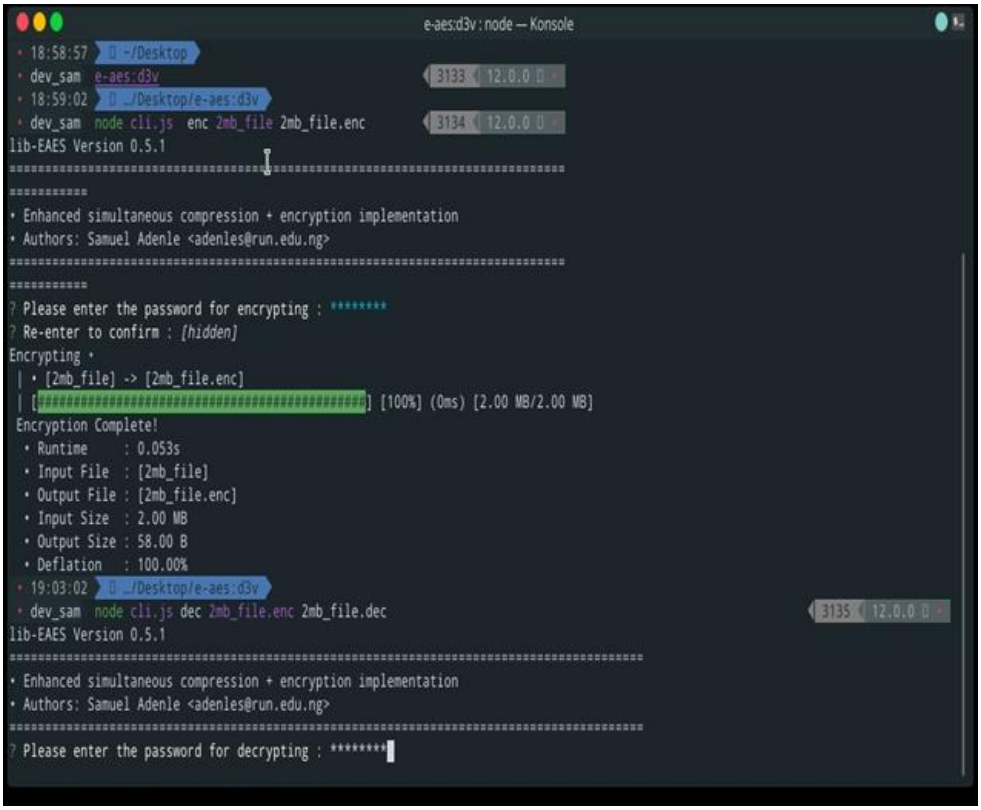

Figure 3. EES Interface

Figure 3 shows the runtime, input file, output file, input size, output size and inflation where Runtime = 0.053s, Input file = [2mb_file], Output file = [2mb_file.encrypted], Input size = 2.00MB. Output size = 58.00B and the result is then displayed as shown in the parent directory.

4.3. System Testing

In this section, the EES was tested using visual studio. Key generation, key management infrastructure, decryption, encryption, authorisation and authentication were all provided by Python's security features. EES and the three other benchmarking algorithms were used to encrypt the same input file at different times. The output files, which are the encrypted files, were saved for each algorithm used. These encrypted outputs were also used as input for the decryption processes by each algorithm tested. Additionally, all experiments were conducted on the same system, which implied the same environment and the same processing capabilities. Some of the evaluation parameters used in this work are described in the next section.

- **Encryption time**: this is a crucial factor that influences the performance of the system. It is the time spent in the conversion from plaintext to ciphertext. Plaintext block size and mode, key size are some of the factors that determine the encryption time. The goal of the system is to reduce the encryption time as the best system is the one with the least or fastest encryption time. In all the experiments conducted in this work, encryption time was measured in seconds (s).
- **Decryption time**: this is another crucial factor that influences the performance of the system. It is the time spent in recovering the plaintext from the ciphertext. The goal of the system is also to reduce the encryption time as the best system is the one with the least or fastest decryption time. In all the experiments conducted in this work, decryption time was measured in seconds (s).
- **Data size:** different encryption techniques to encrypt and decrypt various data sizes. In all experiments conducted in this work, data size was measured in kb.
- **Inflation:** this is an important property in lossless compression processes to maintain the completeness of the data without adding unwanted computations. In all experiments conducted in this work, inflation was measured in percentage (%).
- **Deflation**: this is an important property in lossless compression processes to maintain the completeness of the data without adding unwanted computations. In all experiments conducted in this work, deflation was measured in percentage (%).

The system was tested with a 2mb_file for the compression/encryption. Executing the EES decompression/decryption process displays the data input size first, and the running time is displayed upon completion. The process did not terminate until the process was completed and the status displayed. The results, displayed in Table 2, showed that the time spent for compressing and encrypting seven data samples was proportional to the data sizes, i.e. the bigger the size, the more time it consumed. Performance results also showed that when the size of the sample files was increased, the compression and encryption time also increased.

Table 2. Experiment conducted for compressing and encrypting different data sizes using EES.

Input File	Input Size (KB)	Deflation	Runtime (s)	Output File	Output Size (KB)
Flowchar	62.95	15.76%	0.158	flowchart.enc	53.03
sam.png	110.93	0.30%	0.121	Sam.enc	110.59
finalproje	824.21	0.04%	0.166	finalproject.en	823.87
cmp802.	875.07	4.58%	0.134	cmp802.enc	834.96
Doc13.pd	1040	1.15%	1.193	doc13.enc	1030
wls.pdf	3110	2.54%	0.224	wls.enc	3030
setup.exe	5660	53.88%	0.535	setup.enc	2610

Results of experiments conducted to decompress and decrypt using the EES Application is shown in Table 3.

Table 3. Experiment conducted by decompressing and decrypting different data sizes using EES.

Input File	Input Size (KB)	Inflation	Runtime(s)	Output File	Output Size (KB)
flowchart.enc	53.03	18.71%	0.713	flowchart.png	62.95
sam.enc	110.59	0.30%	0.054	sam.png	110.93
finalproject.enc	823.87	0.04%	0.398	finalproject.ppt	823.21
cmp802.enc	834.96	4.80%	1.281	cmp802.pptx	875.07
doc13.enc	1030	1.16%	0.418	doc13.pdf	1040
wls.enc	3030	2.60%	0.107	wls.pdf	3110
setup.enc	2610	116.83%	2.78	setup.exe	5660

Table 3 showed that the time spent for decompressing and decrypting the sample data was proportional to the data size, i.e. the bigger the data size, the more time it consumed. Performance results showed that when the size of the sample files was increased, the decompression and decryption time also increased.

4.4. Benchmarking with Existing Algorithms

In order to validate the efficacy of the EES algorithm, a comparative analysis was carried out against existing algorithms like AES, DES and RSA [16], which was implemented without compression. Four text files of different sizes were used for the experiments to compare the four algorithms. Performance results showed that when the size of text files was increased, the encryption and decryption time also increased. The performance metrics used was the time spent for compressing and encrypting the data samples. The experiment was repeated four (4) times, and the results are presented in Table 4.

Table 4. Experiments conducted on four different datasets using symmetric and asymmetric algorithms [21]

S/NO	Algorithm	File Size (Kb)	Time Taken to Encrypt (s)	Time Taken to Decrypt (s)
1	EES	153	0.103	0.054
	DES		3.000	1.100
	RSA		7.300	4.900
	AES		1.600	1.000
2	EES	192	0.056	0.100
	DES		2.000	1.240
	RSA		8.500	5.900
	AES		1.700	1.400
3	EES	312	0.094	0.077
	DES		3.000	1.300
	RSA		7.800	5.100
	AES		1.800	1.600
4	EES	868	0.116	0.067
	DES		4.000	1.200
	RSA		8.200	5.100
	AES		2.000	1.800

When EES was compared with existing AES, DES and RSA on sample3 data, performance results showed that the size of sample3 data was directly proportional to the encryption time, that is, when the size of data increased, the encryption time also increased. It was also discovered that the encryption time of EES was the lowest compared to existing algorithms, but the highest encryption time was recorded in RSA. The same trend was discovered when sample4, sample5 and sample 7 dataset was used for the experiments. For the decryption, it was discovered that the decryption time was directly proportional to the sample data size. The trend was similar on the four datasets used, that is sample3, sample4, sample5 and sample7. EES had the best decryption time while RSA had the worst decryption time out of the four algorithms compared.

5. Conclusion

Complex algorithms are required to not only encrypt but also to compress and distribute data. A study on the enhanced protection of such sensitive data was conducted in this work. The general situation with many existing encryption techniques is that they can be cracked given enough time and resources, and some of them also come with unwanted additional computations. This flaw was addressed in this study by combining the lossless compression technique with encryption in a single process for selective encryption of data. The developed enhanced encryption algorithm, EES, combined three different keys; one hashed from PBKDF2, the other from PI sequence and finally from the DNA sequence, where each one is invariant of the user's inputted key. This operation helped to ensure that infinite possible keys were generated under every iteration. EES produced different ciphertexts for the same plain text, thereby confusing the hacker in the process of brute-forcing the operation. The EES is platform-independent as regards the operating system as importing it would provide the API for manipulating the process. Otherwise, a stand-alone executor is also predefined within it, which means it is an importable library and inherently an executable script. The performance of EES was tested against established encryption methods, which are RSA, DES and AES algorithms, using different experimental dataset. The experimental results showed that the EES

algorithm outperformed other algorithms in terms of encryption time by returning the fastest encryption time while the RSA algorithm had the worst encryption time. In terms of decryption run-time, EES also performed better than the other three existing algorithms. In similar experiments conducted using the image and audio test dataset, EES algorithm performed better than RSA, AES and DES algorithms in terms of decryption and encryption times. Without knowledge of the encoding table, it is impossible to decode the compressed files; the data are encrypted with the enhanced encryption algorithm, which further makes unauthorised access to the files almost impossible. This is considered a principal strength of the EES algorithm developed in this work. Conclusively, EES is a more efficient cryptographic algorithm with better data-at-rest options. Due to rapid advancements in encryption, there is a need to extend newer possibilities and focus on introducing further enhancement with quantum cryptography. This shall be our focus in a future study.

Acknowledgments

The authors appreciate the Department of Computer Science's laboratory for providing a convenient environment for the analysis.

References

[1] B. Carpentieri, "Efficient Compression and Encryption for Digital Data Transmission," International Journal of Security and Communication Networks, vol. 2018, no. 9591768, p. 9, 2018.
[2] S. S and R. L, "Text Compression Algorithms - A Comparative Study," ICTACT Journal on Communication Technology, vol. 2, no. 4, pp. 2-8, 2011.
[3] B. R. Pushpa, "A New Technique for Data Encryption using DNA," in 2017 International Conference on Intelligent Computing and Control (I2C2), Coimbatore, India, 2017.
[4] A. Roy and A. Nath, "DNA Encryption Algorithms: Scope and Challenges in Symmetric Key Cryptography," International Journal of Innovative Research in Advanced Engineering (IJIRAE), vol. 11, no. 3, pp. 48-52, 2016.
[5] P. B. Narasingapuram and M. Ponnavaikko, "DNA Cryptography Based User Level Security for Cloud Computing and Applications," International Journal of Recent Technology and Engineering (IJRTE), vol. 8, no. 5, pp. 3738-3745, 2020.
[6] W. Stallings, Network security essentials, 4th ed., Prentice-Hall, 2011.
[7] P. Sanchita, A. Tausif, Abhishek. and K., "An Innovative DNA Cryptography Technique for Secure Data Transmission," International Journal of Bioinformatics Research and Applications, vol. 12, no. 3, pp. 238-262, 2016.
[8] V. Siddaramappa and K. Ramesh, "Cryptography and Bioinformatics techniques for Secure Information transmission over Insecure Channels," in 2015 International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT), Davangere, 2015.
[9] K. Sasipreetham, "Enhanced Encryption Algorithm Based on Prime Number, DNA Sequence and PI Sequence," International Journal of Computational Engineering Research (IJCER), no. 8, pp. 22-26, 2018.
[10] G. P., A. Singh, S. A. and N. Pahwa, "An Efficient Cryptographic Approach for Secure Policy Based Routing," IEEE Journal on Selected Areas in Communications, vol. 1, pp. 359-363, 2013.
[11] A. Aswin and A. Aswathy, "A novel symmetric cryptography algorithm for fast and secure encryption," in 2015 IEEE 9th International Conference on Intelligent Systems and Control (ISCO), Coimbatore, 2015.

[12] N. Veeraragavan, L. Arockiam and S. S. Manikandasaran, “Enhanced Encryption Algorithm (EEA) for Protecting Users’ Credentials in Public Cloud,” in 2017 International Conference on Algorithms, Methodology, Models, and Applications in Emerging Technologies (ICAMMAET), Chennai, 2017.
[13] J. K. Lyngdoh and D. V. Jose, "Enhanced Content-Based Double Encryption Algorithm using Symmetric Key Cryptography," Oriental Journal of Computer Science and Technology, vol. 10, no. 2, pp. 345-354, 2017.
[14] D. Usha and M. Subbbulakshmi, “Double Layer Encryption Algorithm Key Cryptography for Secure Data Sharing in Cloud,” International Journal of Scientific & Engineering Research, vol. 9, no. 5, pp. 91 - 94, 2018.
[15] B. O. Oguntunde, S. A. Arekete, M. O. Odim and O. I. Olakanmi, "A comparative study of some traditional and modern cryptographic techniques," International Journal of Engineering and Management Research, vol. 7, no. 6, pp. 76-82, 2017.
[16] P. Mahajan and A. Sachdeva, “A Study of Encryption Algorithms AES, DES and RSA for Security,” Global Journal of Computer Science and Technology Network, Web & Security, vol. 13, no. 15, pp. 14-22, 2013.
[17] G. Ruchi, K. Mukesh and B. Rohit, “Data Compression -Lossless and Lossy Techniques,” International Journal of Application or Innovation in Engineering & Management (IJAIEM), vol. 5, no. 7, pp. 120-125, 2016.
[18] K. Boyini, "Huffman Coding Algorithm," Tutorialspoint, Hyderabad, 2018.
[19] A. Nath, Python Cryptography, 2018.
[20] P. Deutsch, "RFC EDITOR-DEFLATE Compressed Data Format Specification version 1.," May 1996. [Online]. Available: <https://www.rfc-editor.org/info/rfc1951>. [Accessed 9 April, 2019].
[21] P. Mahajan and A. Sachdeva, “A Study of Encryption Algorithms AES, DES and RSA for Security,” Global Journal of Computer Science and Technology Network, Web & Security, vol. 13, no. 15, pp. 14-22, 2013.

Authors

S. A. Adenle, is a MSc student in Computer Science. He holds a BSc. in Computer Science. His research interests broadly revolve around the area of Cyber Security, Data Security and Artificial Intelligence.

A. O. Ogunde, is a Reader in Computer Science. He holds a PhD degree in Computer Science and is a registered member of the Computer Professionals Registration Council of Nigeria, Nigeria Computer Society, IAENG and several other professional bodies. He has published widely in both local and international journals. His research interests are in the area of Artificial Intelligence: data mining, machine learning, big data with other intelligent and knowledge-based systems.

B. O. Oguntunde, holds a PhD degree in Computer Science at the Redeemer's University, Ede, Nigeria. She is a Senior Lecturer in Computer Science. Her research interests are in Data Communication, Networking, and Bioinformatics. She has published articles in learned journals and presented papers at conferences.

M. O. Odim, is a Senior Lecturer in The Department of Computer Science, Redeemer's University. He holds a PhD degree in Computer Science and has published widely both local and Internationally. He is a member of the Nigeria Computer Society (NCS) the Computer Registration Council of Nigeria (CPN). He holds the Artificial Intelligence Analyst - Mastery and Explorer Awards of IBM. His research interests include but not limited to Machine Learning, Deep Learning, Data Mining, Database Systems and Information Security

S. A. Arekete, holds a PhD degree in Computer Science. He is a Reader (Associate Professor) in Computer Science at the Redeemer's University, Ede, Nigeria. His research interests are in Artificial Intelligence, Intelligent Agents Systems and Modelling. He has published articles in learned journals and attended conferences.