Journal of King Saud University - Computer and Information Sciences 36 (2024) 101999

HOSTED BY

Contents lists available at ScienceDirect

Journal of King Saud University - Computer and Information Sciences

journal homepage: www.sciencedirect.com

A hybrid approach to secure and compress data streams in cloud computing environment

A.Abdo [a], Taghreed S. Karamany [b,c,*], Ahmed Yakoub [b]

[a] Arab Open University, Faculty of Computing, Cairo 11837, Egypt
[b] Department of Information Systems, Faculty of Computers and Artificial Intelligence, Helwan University, Cairo 11795, Egypt
[c] Future Academy - Higher Future Institute for Specialized Technological Studies, 29 Ismailia Desert Rd, El Shorouk - Cairo 6363040, Egypt

ARTICLE INFO

Keywords:
Cloud Computing
Information security
Symmetric ciphers
Data Compression
Data streams
LZMA

ABSTRACT

Cloud computing has revolutionized the way businesses and individuals manage and utilize computing resources, providing significant benefits such as scalability, flexibility, and cost-effectiveness. However, the increasing use of cloud computing to store and transmit sensitive data has raised concerns about data security. Ensuring confidentiality and data integrity while enabling effective data transmission is a significant challenge. To overcome this issue, the proposed approach integrates encryption and compression methods to enhance transmission performance in the cloud and prevent unauthorized access to confidential information. This approach applies multiple layers of robust encryption algorithms, followed by LZMA, which aims to compress data size while ensuring data security. The proposed approach is well-suited for real-time implementation and delivers high encryption quality and compression capabilities. Various performance metrics evaluate its performance, including space-saving percentage, processing time, and the NIST randomness test. It provides a substantial improvement in space-saving percentage from 58.63% to 81.8%, ensuring efficiency. The security analysis confirms that ciphertexts generated by this approach pass all NIST tests, generating a 99% confidence level regarding the randomness of the ciphertext. The proposed hybrid approach offers an effective solution for addressing the challenges of securing sensitive data while taking advantage of the benefits of cloud computing.

1. INTRODUCTION

Cloud computing has become an increasingly popular way in the networking and computer science field. It has given small and medium-scale businesses with a cost-saving opportunity to avoid the expense of purchasing hardware resources. It gives us an equal opportunity to shine as it enables users with access to a powerful computing resource without having to invest in expensive hardware or software. Cloud computing operates on the principle of virtualization, wherein a single large machine is utilized by multiple clients with a view that they have their dedicated resources. Due to the significant benefits of managing resources in unlimited storage in the most inexpensive way, business continuity, and scalability, cloud computing has had rapid growth in the IT industry, therefore, this technology has emerged as one of the most powerful innovations that has attracted the interest of technologists globally (Sajay, 2019). Cloud computing services can be classified into three primary categories: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). SaaS provides software packages, system software, and application-related server storage networks to end users such as Gmail. PaaS is used by application developers to create applications that are hosted in the cloud such as Google App Engine. IaaS deals with hardware services, virtual machines, and network architecture infrastructure such as Microsoft Azure (Mahesh et al., 2023). The deployment models include private cloud, public cloud, community cloud and hybrid cloud. Public clouds are available to everyone while private clouds are used by organizations. Community clouds are shared by multiple organizations and hybrid clouds combine both private and public clouds (El-Booz and Attiya, 2017; Thabit et al., 2021).

Although Cloud computing has become a magical target for numerous benefits, but these benefits are limited due to several issues concerning security (Thabit et al., 2021). Availability, integrity, and accessibility are the three main cloud computing's barriers; Cryptography techniques could be utilized to provide confidentiality, integrity, and availability to the data stored or accessed through the cloud (Alemami et al., 2023). Compression, in addition to cryptography, is a

* Corresponding author.
E-mail address: taghreed.salem@fa-hists.edu.eg (T.S. Karamany).

https://doi.org/10.1016/j.jksuci.2024.101999
Received 31 July 2023; Received in revised form 14 February 2024; Accepted 27 February 2024
Available online 2 March 2024

crucial technique for cloud computing since it helps reduce the size of data that needs to be stored and transferred. Compression can help to reduce storage costs, improve network performance, and reduce bandwidth usage.

1.1. Cryptography

Data security is crucial for preserving all of the features and advantages offered by cloud computing. In order to ensure data confidentiality through the network, cryptography can be used. This is done by using both encryption and decryption methods. The main goal of cryptography is to provide a set of security features that ensure the confidentiality of the system is protected. These goals can be categorized into the five categories listed below (Sullivan, 2014).

- Authentication: Before sending the message, the sender and recipients' identities must be verified.
- Confidentiality: The message can only be interpreted by authorized users, and no one else can use it.
- Integrity: ensuring that the content of the transmitted data does not contain any kind of modification.
- Service reliability and availability: since intruders can disrupt the availability of services to users, the technology should be capable of providing users with the expected quality of service.
- Non-repudiation: This function indicates that neither the sender nor the recipient can deny that a particular message has been sent.

Cryptography is the technique of information security used to protect information by converting it (encrypting it) into an unreadable format known as encrypted text (cipher text). The encryption and decryption technologies are available in three types: symmetric, asymmetric, hybrid algorithms which may be utilized in cloud computing environment to encrypt and decrypt data (Arora and Parashar, 2013).

- Symmetric Encryption: is a cryptographic method that encrypts and decrypts data using a single secret key shared by the sender and recipient (Sherief, 2022) as shown in Fig. 1.
- Asymmetric Encryption: is a type of cryptographic method that employs a pair of related keys, consisting of a public key and a private key, to encrypt and decrypt data. As shown in Fig. 2. the authorized receiver only is able to decode the message using the private key (Sherief, 2022).
- Hybrid Encryption:is a type of encryption that combines multiple or more encryption algorithms, using a combination of symmetric and asymmetric encryption to take use of each type's capability. This hybrid cryptography method was created in order to provide an efficient and secure encryption algorithm capable of encrypting and decrypting data fast and safely (Akashdeep Bhardwaj, 2016).

1.2. Data Compression

The way of reducing the size of the original data by using specific encoding techniques, which saves storage capacity, increases file transfer speed, and lowers the cost of storage hardware and network bandwidth, can be classified into two types: lossy compression techniques and lossless compression techniques (Ignatoski and Lerga, 2020).

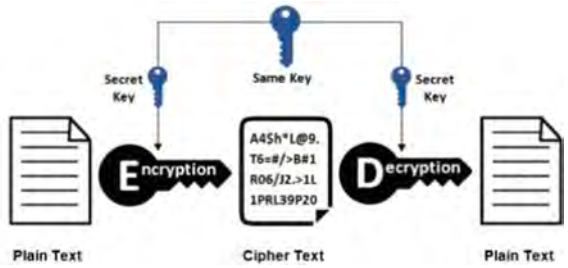

Fig. 1. Symmetric encryption.

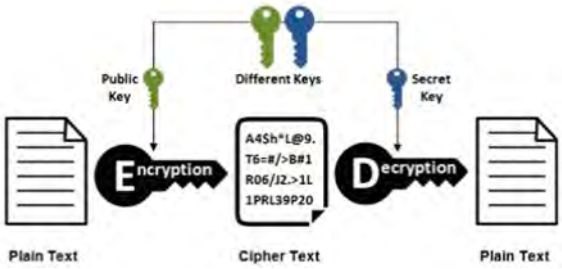

Fig. 2. Asymmetric encryption.

- lossy compression: discards some of the original data during compression. Although this results in a smaller file size, the decompressed data is not an exact replica of the original data. Lossy compression is often used for image and audio files include JPEG, MP3 and MPEG.
- Lossless compression: allows for the original data to be restored perfectly after decompressing, meaning that no data is lost in the compression process. Examples of lossless compression techniques include LZMA, Lempel-Ziv-Welch (LZW), Huffman Encoding, Run Length Encoding, and Arithmetic Coding (Adedeji, 2020).

This study aims to experimentally investigate the impact of applying compression and encryption to streams of text data in a cloud computing environment. Specifically, this research aims to provide robust encryption quality and acceptable compression capability, by identifying Execution time, calculating the space-saving percentage %, and comparing the result with the previous work, Furthermore, the resulting security level is evaluated using the NIST randomness test.

The subsequent sections of this paper are structured as follows. The related work will be discussed in Section 2. Section 3 presents the proposed technique and its various parts. Section 4 illustrates the simulation setup and the experimental results. Finally, Section 5 concludes the paper and sketches out the future work.

2. Related work

This section reviews and examines previous work in cryptography and compression to offer a better knowledge of the effectiveness of encryption algorithms by using less storage space and therefore improving performance and security level.

Nazia Shoukat et al. (2022) have implemented a new technique to improve the effectiveness and security of the traditional Vigenère cipher which is typically vulnerable to attacks. Their approach involves utilizing the Relative Frequency of Alphabetic Letters, arranging the sequence of the letters according to the relative frequency, and arranging them in increasing order to improve the data confidentiality of cipher text security features. Further improvements were made by modifying the Vigenère cipher using the relative letter frequency technique, and by compressing the text data using a lossless Huffman technique. The result was shorter, and more secure than what was previously achievable. Overall, this study significantly enhanced text message confidentiality by using multiple layers of security procedures, including encryption, decryption, and compression.

Sunil Kumar P et al. proposed a method (Kumar et al., 2023) that employs a two-factor data encryption protection mechanism based on Identity-based Encryption (IBE) algorithm that integrates unique customer users. To ensure maximum security, the data and documents are compressed and protected using the LZ4 algorithms, which offer both encryption and compression. This compression, in particular,

reduces storage space requirements as well as ensuring efficiency, as evidenced by a compression time of 0.15 s and a compression ratio of 0.949. These password-protected files will be kept on the cloud and the key will be issued.

In another study, Padmapriya et al. (Padmapriya and Eric, 2023) conducted an experiment to examine how data compression affects an amino acid encrypted text while maintaining security. They applied dictionary-based and entropy coding methods to compress ciphertexts of varying sizes using two popular algorithms: Huffman (based on entropy coding) and LZMA (a dictionary-based compression algorithm). The compression ratio was calculated for each file size. The researchers found that the entropy coding method saved 47 % of storage space, while the dictionary-based method saved 60 %, resulting in improved storage efficiency. These findings illustrate the benefits of this technique and highlight its potential for enhanced data storage.

Usama et al. in (Usama et al., 2021) conducted a comprehensive investigation into issues associated with combining data compression and encryption techniques without compromising either. The study involved utilizing an effective and secure data compression algorithm with cryptographic capabilities that merged adaptive Huffman coding, a pseudorandom keystream generator, S-Box, and a chaotic logistic map to integrate key control. The resulting secure adaptive Huffman coding can perform simultaneous secure compression and decompression, with data substitution applied via a chaotic S-Box and masked pseudorandom keystreams used to enhance encryption quality. The analysis revealed that the proposed technique enables faster processing times compared to separately running encryption and compression algorithms. The approach successfully compresses secure data, making it suitable for real-time applications, while the security evaluation demonstrates its sensitivity to plaintext and key. Moreover, the generated ciphertexts successfully pass all NIST tests with 99 % confidence in their randomness. The compression efficiency analysis indicates that the proposed method provides similar space-saving capabilities to standard techniques, while still ensuring adequate security.

Based on the one-time pad algorithm, the authors in (Al-Smadi et al., 2021) proposed a practical file cryptography mechanism by controlling the data type in the encryption key. This methodology overcomes the management issues with Vernam algorithm's encryption keys, where the output file size was reduced by applying the Huffman algorithm. The output file was both password-protected and AES-encrypted, resulting in improved security against potential attacks. The study utilized different file types, including (txt, pdf, doc, bmp, mp4, and exe). The results also showed that using a cryptographic key produced from integers reduced the time required for file encryption and decryption when compared to using an ASCII table for the key. However, the file size had little impact on the encryption time without compression.

A hybrid approach for securing SMS is presented in (Mahmoud et al., 2009). This method combines the compression and encryption processes. The SMS data is compressed using a lossless method. After this, the compressed SMS data is encrypted with the RSA technique. The benefit of this approach is to achieve protection criteria such as confidentiality and authenticity between two communication parties while also reducing message lengths. When compared to a technique that only uses the RSA encryption algorithm to protect SMS, the results show that the technique does not exceed the standard SMS length.

Makala Et al. (Makala et al., 2017) proposed another mechanism for compressing and encrypting data at the same time, achieving a minimum of 12 % compression. Their symmetric key encryption method uses a table of characters and their order. The table itself becomes the key and doesn't have to be sent every time. The 81 positions in the table make it difficult for intruders to break their mechanism. This method is well-suited for compressing SMS messages and provides a speedy approach for compressing, encrypting, and creating a cipher file.

The combination of RLE and knapsack was proposed by Marto, et al. in (2020). According to tests, the Run Length Encoding approach can successfully compress text if it has a large number of repeated letters. This method performs poorly when compressing text with minimal to no letter repetition due to the final file size being larger than the original. When used for text encryption, the Knapsack algorithm can successfully secure messages. The combination takes precedence Encryption followed by text compression is preferable since the combination of the two successfully compresses data better than prioritizing compression followed by encryption, even though the time to execute plaintext is longer. If there is a lot of character repetition in plaintext (the original message), the combination of Knapsack and RLE Algorithms in the Text file is appropriate.

N Sangwan (Sangwan, 2012) introduced a different mechanism to find out a method of making text data or messages highly secured and smaller in size than the original. To accomplish this goal, it combined an existing most effective Huffman Compression Technique on text data (to reduce file size) with a newly developed Block type Symmetric Key Algorithm (to ensure security). Two private keys have been used, which are known to both the sender and the receiver but are unknown to the outside world, that is why it is known as Secret Key Cryptography. The entire system achieves the goals of cryptography, is simple, and does not overlook security concerns. Because of its large key domain, it is not vulnerable to brute-force attacks.

Although Text File Services is still viable for sending messages, it lacks certain security features present in Instant Messengers, such as cryptography. Therefore, Kuswanto, D (Kuswanto, 2020) proposed the addition of a cryptography feature using the RSA algorithm to Text File Services. However, this algorithm can increase the plaintext's character count, so a compression algorithm called Shannon Fano was also added. The RSA and Shannon Fano methods were implemented in Text File Services, and the length of the key used was positively correlated with an average Test Compression Ratio of 1.41 and a space-saving of 29.37 %. The study concluded that as the key length increased, the decryption process took longer, and the average value of the avalanche effect ciphertext and the Compression result remained relatively the same.

Ruchita et al. (Sharma and Bollavarapu, 2015) used compression and encryption techniques to secure data, employing various cryptography algorithms in the process. Run-length encoding with RC4 and Caesar Cipher was used on text files of different sizes, but this combination is less effective when there are fewer consecutive characters present. Huffman with RC4 and DES had the best compression ratio among all compression algorithms for these five different text file sizes, reducing file size by about half. While LZW and Arithmetic techniques produced good results, Huffman was still much better. Compression was followed by encryption to improve security, and the authors discovered that Huffman provided a higher Compression Ratio than all other approaches by almost 50–80 %. In conclusion, Huffman compression algorithm is the best for text compression, followed by LZW, Arithmetic, and run length.

B. Carpentieri (Carpentieri, 2018) conducted a study that combined compression and encryption techniques on various digital data. The Calgary corpus was used as input, along with four encryption algorithms (DES, 3DES, AES, RC4), and four standard compression algorithms (Huffman coding, Arithmetic coding, Lempel-Ziv-welch coding, and run length encoding) were used. Two rounds of testing were performed. The first involved compression followed by encryption, while the second involved encryption followed by compression. The study demonstrated that the cost of encryption after compression for text data is negligible. Also, compressing after encryption does not help and even increases both the file and encrypted sizes. Furthermore, due to the randomness introduced by the encryption algorithms, the results revealed that arithmetic coding nearly doubled the original file size.

M. R. Ashila et al, (Ashila et al., 2019) proposed a combination of AES - Huffman code method to produce secure file encryption and minimize file size to reduce storage space and expedite the transmission of file transfers. Whereas encryption is performed first, followed by compression. To measure the level of file security, the avalanche effect (AE) and entropy after encryption and compression were measured. The

findings of this study showed that AES encryption increases the file size by around 25 % of the original, but the encrypted file size shrinks by roughly 30 % after implementing Huffman compression, and results show that the compressed file was lower than the original file size. Where it has been noticed that Huffman compression has a positive effect on AES encryption, Huffman can also increase file security based on measurements of AE and entropy values.

Based on what was discussed in the previous sections, the most effective way for cloud computing is to combine cryptography and compression algorithms. Table 1 demonstrates the main features and limitations of all previous techniques.

From the comparative study, the study in (Usama et al., 2021) showcased an efficient technique that combined data compression and encryption without compromising either process. By integrating adaptive Huffman coding, a pseudorandom keystream generator, and S-Box, this approach reduced processing time and achieved comparable compression efficiency to standard techniques, the Security analysis confirmed its sensitivity to the plaintext, and the generated ciphertexts passed all NIST tests for randomness.

3. The proposed approach

This Research focuses on incorporating existing encryption and data compression techniques to enhance the security and efficiency of storing and transferring data streams within the cloud computing environment, utilizing multiple layers of security through the use of Rc4 Encryption Algorithm, AES Encryption Algorithm, and Vigenère polyalphabetic substitution cipher with utilizing secret keys for them, as each key is different from the other. To securely share the secret keys for encryption and decryption processes, the sender and recipient should generate a random, complex key. They then exchange this key via a secure communication channel, such as a private network or secure messaging platform, to prevent interception or tampering during transmission. While the keys must be kept secret and must not be disclosed to the public or given to any unauthorized parties to ensure confidentiality. While LZMA compression algorithm is used to minimize storage requirements, it also facilitates the efficient uploading and transferring of data to the cloud. As demonstrated below in Fig. 3, the proposed framework comprises two components: the sender side and the receiver side. Each side comprises seven phases. On the sender side, these phases include initial encryption, decomposition, substitution, data size treatment, compression, encryption 2, and uploading encrypted-compressed data in the cloud. The receiver side follows the inverse processes of downloading encrypted-compressed data from the cloud, decryption 2, decompression, data size un-treatment, substitution decipher, composition, and decryption 1. The step-by-step processes involved in the proposed approach are illustrated in detail in the following subsections. As Fig. 4 illustrates the sender side implementation.

3.1. Sender Side

3.1.1. Initial Encryption

In step 1, the sender first encrypts the data using the RC4 algorithm, which is commonly used as a fast encryption algorithm due to its lightweight and results in low memory usage, power consumption, lower complexity, and higher throughput compared to several other cryptographic methods (Abdulameer, 2023). RC4 (Jindal and Singh, 2015) operates byte by byte with a secret key agreed upon by the sender and the receiver. RC4 comprises two main components: the Key Scheduling Algorithm (KSA) and the Pseudo-Random Generation Algorithm (PRGA) (Jindal and Singh, 2015). The KSA generates a 256-byte initial state known as the permutation, which is used by the PRGA to generate a keystream. To generate the ciphertext, the keystream is XORed with the plaintext message one byte at a time as shown in Fig. 5. This process is repeated until encrypting the entire plaintext message and the size of ciphertext becomes same as the original plaintext in binary format.

3.1.2. Decomposition phase

During **Step 2** of the methodology, the encrypted data obtained from $Cipher_1$, each byte (8 bits) is divided into four equal parts, each part consisting of 2-bits binary combination, with the possibilities being '00′, '01′, '10′, and '11′. After that replacing each set of 2- bits binary with a corresponding character, for instance, '00′ is replaced with 'A', '01′ with 'B', '10′ with 'C', and '11′ with 'D' as illustrated in (Fig. 6 and pseudocode 1), but this replacement led to a 4-fold increase (400 %) in its original size.

Table 1
Comparison between related studies.

References	Data used	Methodology used	Results	Limitations
(Padmapriya and Eric; Kumar et al., 2023)	Text	LZMA, Huffman coding	The entropy coding method saves 47 % of storage space, while the dictionary-based coding method saves 60 %. storage efficiency is also doubled	The security level is not clear.
(Usama et al., 2021)	Text, image	Adaptive Huffman coding, chaotic logistic map.	This method proved useful for real-time implementation as it minimized data storage and transmission consumption while successfully passing all NIST tests, demonstrating its reliability and effectiveness.	————
(Al-Smadi et al., 2021)	Text, Image, Audio	one-time pad, Vernam algorithm, Huffman, and AES	The approach made the output file more difficult to attack and reduced the amount of time to encrypt and decrypt a file.	Not taking into account the potential impact of data compression on the accuracy of the encrypted text
(Marto Hasugian et al., 2020)	Text	RLE then Knapsack	Text Compression using the Run Length Encoding algorithm can be compressed well if there are many sequential letters in the text.	RLE performs poorly when compressing text with little to no letter repetition.
(Sharma and Bollavarapu, 2015)	Text	Run length with RC4 and Caesar, Huffman with RC4 and DES, LZW & Arithmetic with RC4 & DES.	The best compression algorithm is Huffman then LZW then Arithmetic then run length give better result for text compression.	A text file with fewer consecutive characters may result in lower quality output.
(Carpentieri, 2018)	Text	Huffman coding, Arithmetic coding, Lempel-Ziv-welch coding, and run length encoding. DES, 3DES, AES, and RC4	The cost of encryption after compression was negligible for text data. However, the opposite case, where files were compressed after encryption, did not help.	Encrypting data after compression had minimal cost, while compressing data after encryption did not provide any benefit.
(Ashila et al., 2019)	Text, images	AES, Huffman code	AES encryption increases file size by 25%, but Huffman compression shrinks the encrypted file code by 30%.	AES encryption increases file size by 25%

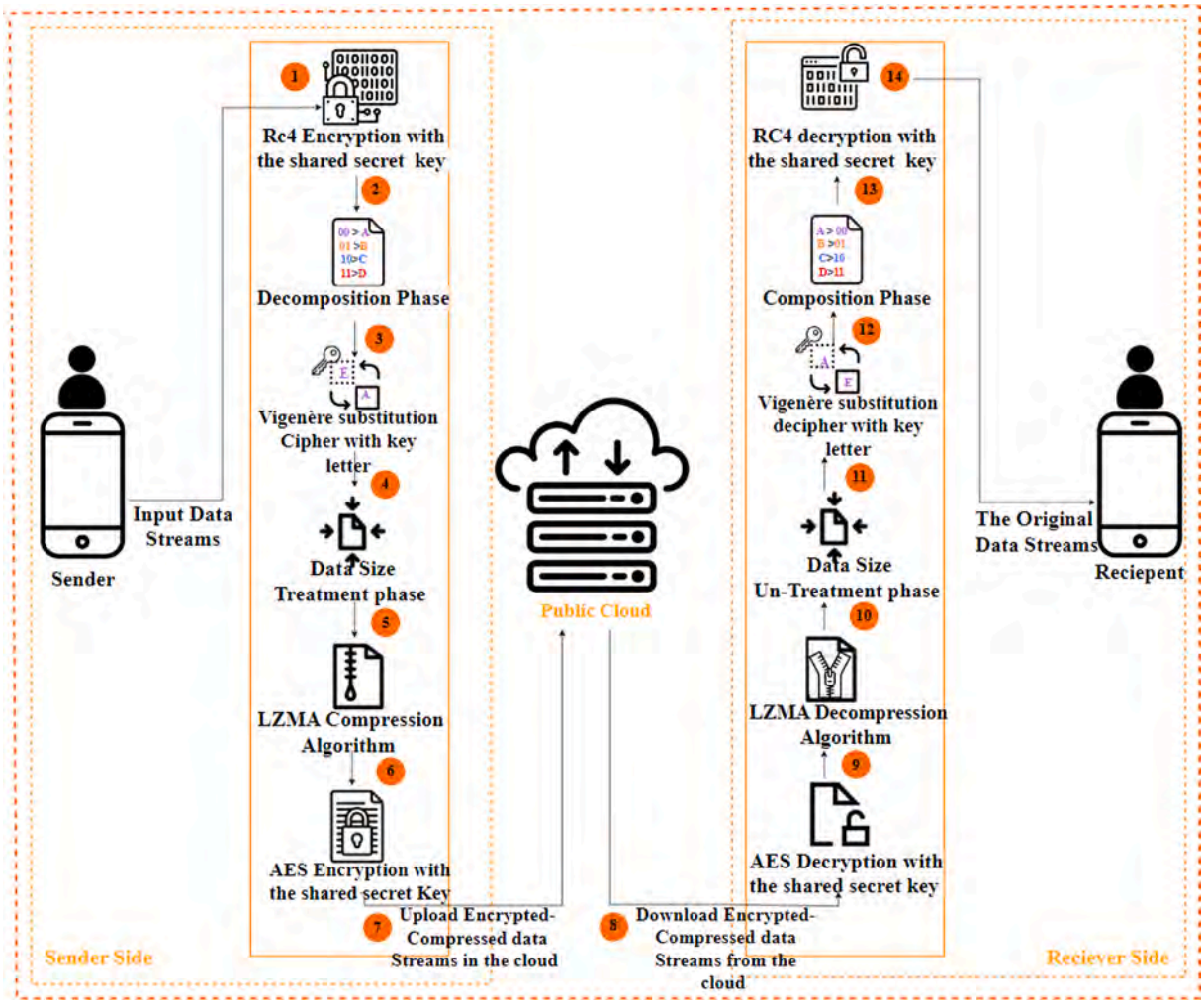

Fig. 3. The framework of the proposed approach to encrypt and compress data streams in the cloud.

Algorithm to replace each 2-bits with 1 letter (stream of binaries)

```
# Input: Sequence of 8 bits in its binary format (stream of zeros and ones)
  # Output: Sequence of letters (A, B, C, D)
  # step 1: Initialization phase
  start = 0
  end = 2
  array1 = []
  array2 = []
  # step 2: Divide each 8-bit binary into four equal part of 2-bits
  for i in range (0, len(input_str), 2):
  array1.append(input_str[start:end])
  start += 2
  end += 2
  # step 3: Replace each 2-bit binary with a letter
  for j in array1:
  if j == '00':
  array2.append('A')
  else if j == '01':
  array2.append('B')
  else if j == '10':
  array2.append('C')
  else if j == '11':
  array2.append('D')
  return array2
```

Pseudocode for replacing each 2-bit binary with 1 letter.

3.1.3. Substitution cipher phase

In **step 3** of the methodology, a character sequence consisting of letters A, B, C, and D undergoes substitution cipher using the Vigenère polyalphabetic substitution cipher. This cipher involves the use of a key letter to determine the shift applied to each letter in plaintext with the agreement of both the sender and the receiver on the shared secret key. This results in a more robust form of encryption when compared to simple substitution ciphers (Imanda et al., 2023). According to the secret key, this substitution process produces a sequence of four distinct characters as output. These new letters could be any combination of four letters in a sequence, such as E, F, G, and H. as shown in Fig. 7,based on the presumption that the key is the letter "E," though it could be any other letter in the alphabet, and therefore the sequence generated will differ.

3.1.4. Data size treatment

In **Step 4**, convert each character in the streams of E, F, G, and H into a 2-bit binary format of '01′, '10′, '11′, or '00′, as depicted in (Fig. 8 and pseudocode 2). This action returns the data size to its original size and resolves the issue that occurred in **Step 2** thus the data is 25 % compressed. Following that, replace each 8-bit binary with its corresponding ASCII character. as shown in (Fig. 9 and pseudocode 3). But it has been noticed that various ASCII codes signify nulls or white spaces. It is crucial to determine precisely which byte corresponds to these values to accurately reverse binary code into its original textual form while maintaining the data's original size. As per the ASCII table (ASCII Table), the initial 32 characters are comprised of unprintable control

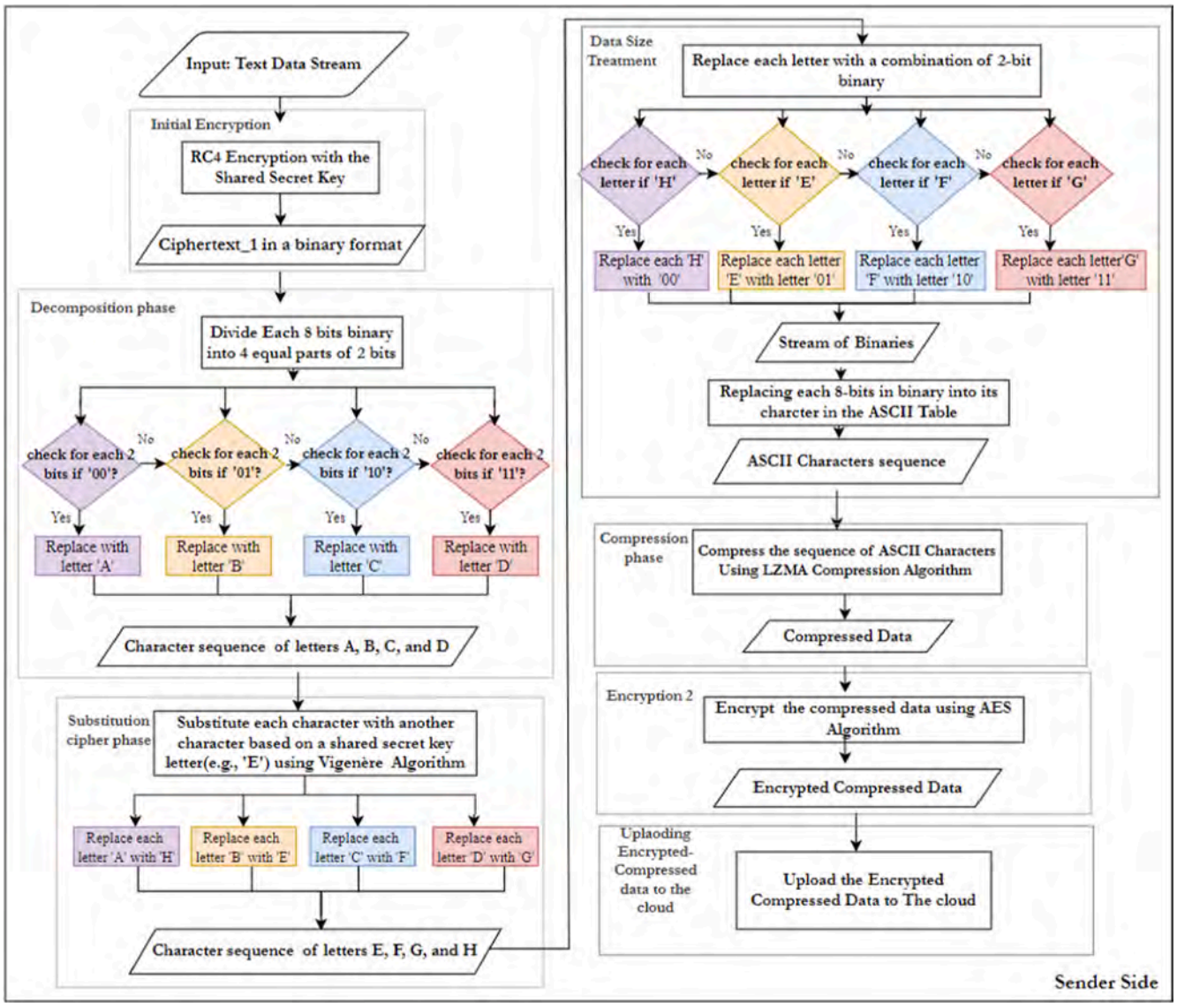

Fig. 4. The Sender Side implementation.

Fig. 5. RC4 Encryption Algorithm.

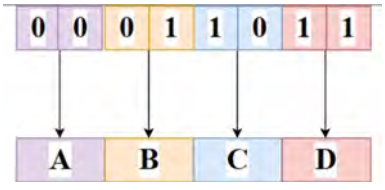

Fig. 6. Replacing Each 2Bits of Binaries With 1 Letter.

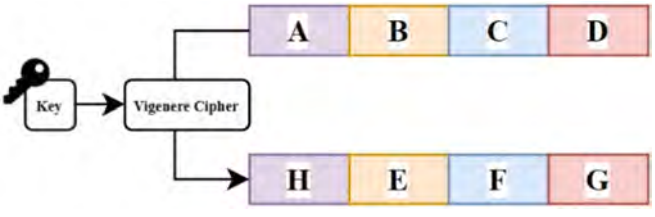

Fig. 7. Substitution cipher phase using Vigenère with a key letter 'E'.

Fig. 8. Replacing each letter with a combination of 2-bit binary.

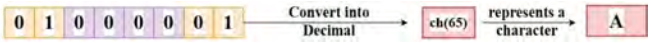

Fig. 9. Replacing each 8-bit in binary into its letter in ASCII Table.

codes that can manage input/output devices like printers, keyboards, etc. Identifying these values is crucial to ensuring correct data handling. Therefore, we have developed a Pseudocode algorithm for converting binary code into its corresponding characters, including specific conditions for these exceptional cases. This algorithm is used in the proposed technique to effectively reverse the binary code into its original textual form without any scrambling or increasing the original size.

Algorithm to convert each letter in the sequence of E, F, G, and H into a 2-bit of binaries

```
# Input: Sequence of characters (input_str)
  # Output: Sequence of the corresponding 2-bit of binaries
  # Step 1: Initialization phase
  output_str = ""
  indices = {"E": "01", "F": "10", "G": "11", "H": "00"}
  # Step 2: Loop through each letter in the input string and replace with 2-bit binary
  representation
  for char in input_str:
  output_str += indices[char]
  # Step 3: Return a Sequence of the combination of 2-bit of binaries
  return output_str
```

Pseudocode 2 for converting each letter into a combination of a 2-bit binary.

Algorithm to convert each 8-bit of binaries into its character in ASCII

```
Input: Sequence of Zeros and ones
  Output: Sequence of Strings
  # Step 1: Initialization.
  start = 0
  end = 8
  array1 = []
  array2 = []
  # Step 2: Loop to divide each 8-bit of binaries
  for i in range (0, len(input_sequence), 8):
  array1.append(input_sequence[start:end])
  start += 8
  end += 8
  # Step 3: Convert each set of 8-bit binary into its corresponding letter based on the
  ASCII table.
  Unprintable= {'00000000':'⓪','00000001':'①','00000010':'②','00000011':'③',
  '00000100':'④', '00000101':'⑤','00000110':'⑥','00000111':'⑦','00001000':'⑧',
  '00001001':'⑨','00001010':'⑩','00001011':'⑪','00001100':'⑫','00001101':'⑬',
  '00001110':'⑭','00001111':'⑮','00010000':'⑯','00010001':'⑰', '00010011':'⑲',
  '00010100':'⑳','00010101':'21','00010110': 22','00010111':'23 ', '00010010':'
  18',
  '00011000':'24','00011001': 25','00011010': 26','00011011':'27','00011100':'28',
  '00011101':'29', '00011110':'30','00011111': 31'}
  for r in array1:
  if r in unprintable:
  array2.append(unprintable[r])
  else:
  ascii_value = ''. join([chr(int(r[i:i + 8], 2)) for i in range(0, len(r), 8)])
  array2.append(ascii_value)
  # Step 4: Return the final sequence of ASCII values as Strings
  return array2
```

Pseudocode 3 for replacing each 8-bit of binaries into its character in ASCII.

3.1.5. Compression phase

Step 5 involved compressing a sequence of ASCII characters using the LZMA compression algorithm. LZMA (Lempel-Ziv-Markov chain algorithm) (Pavlov, 2024) is a type of lossless data compression algorithm. It is generally considered to be one of the most effective compression algorithms available, achieving higher compression ratios. The LZMA algorithm uses a dictionary-based compression system (Stecuła et al., 2022). It finds matches using dictionary data structures and produces a stream of literal symbols and phrase references, where the range encoder encodes one bit at a time. Range encoding is implemented in binary and divides integers using shift operations rather than slow division methods; allowing it to efficiently compress both repetitive and non-repetitive data. It is commonly used to compress large amounts of data. It has become one of the most widely used compression algorithms due to its efficiency and versatility.

3.1.6. Encryption 2

In step 6, the compressed data is encrypted using AES (Advanced Encryption Standard) encryption algorithm after being compressed with LZMA with the agreement of both the sender and the receiver on the shared secret key. AES (Abdullah, 2017) is a widely used symmetric block cipher encryption algorithm, known for its effectiveness in securing sensitive data. AES supports key lengths of 128, 192, or 256 bits and operates on plaintext in blocks of 128 bits (Al-Amri et al., 2023). Accordingly, AES applies a series of substitution and permutation operations to encrypt and decrypt data (Matta et al., 2021). The security of AES encryption relies on the difficulty of determining the key from the ciphertext. This step ensures that the compressed data is securely protected from unauthorized access and any malicious activities such as hacking and data theft using the AES algorithm with a 128-key length.

The combination takes precedence Data Compression followed by Encryption_2 is preferable. The reason is that the combination of LZMA then AES is more effective at compressing data and securing it more than prioritizing encryption then compression and this has been proven by the results of the experiment.

3.1.7. Uploading Encrypted-compressed data TO the cloud

In step 7, After completing the previous steps, the compressed and encrypted data can now be securely uploaded to a cloud storage provider. Whereas, these steps provide a reliable way of securing the integrity of valuable data while it's stored remotely.

3.2. Receiver Side

To decode the incoming data streams and retrieve the original text content, the receiver needs to carry out certain steps. These steps involve downloading and analysing the encoded text data and converting it into its original unencoded form using the shared secret key agreed upon by the sender. As Fig. 10 illustrates the receiver side implementation in below.

3.2.1. Downloading Encrypted-compressed data from the cloud

Step 1, the recipient should download the compressed, encrypted data while also taking care to ensure the data's integrity. Because the data is encrypted and compressed, it must be decoded to be understandable using the steps outlined below.

3.2.2. Decryption 2

Step 2, the downloaded encoded text data will be first decrypted by the receiver by utilizing the AES algorithm (Abdullah, 2017) and the shared secret key that was provided during the Encryption_2.

3.2.3. Decompression phase

Step 3, involves reconstructing the original data that was compressed using the LZMA algorithm by extracting the compressed data and transforming it back to its original uncompressed form. The

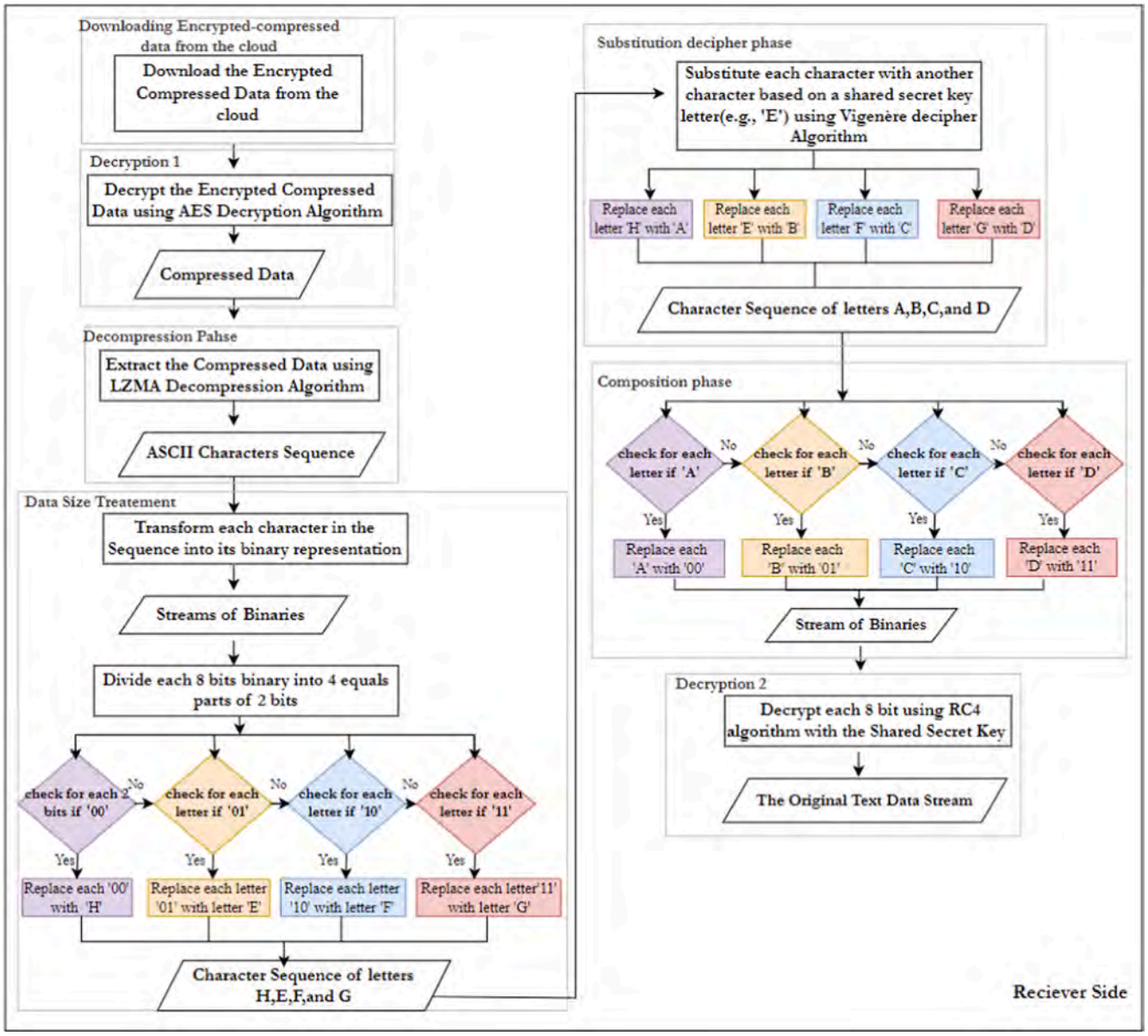

Fig. 10. The Receiver Side implementation.

resulting data will be a sequence of characters represented in the ASCII character set.

3.2.4. *Data Size un-treatment*

Step 4, the receiver should transform every character in a sequence that belongs to the ASCII character set into its corresponding binary representation (ASCII Table), as shown in (Fig. 11 and pseudocode 4). Each pair of binary bits in the sequence will be changed with a corresponding character to complete the transformation.'01′ will be replaced with the character 'E', '10′ with 'F', '11′ with 'G', and '00′ with 'H' as shown in (Fig. 12, pseudocode 5).

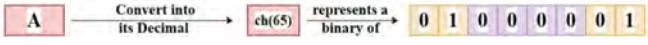

Fig. 11. Converting each character into its binary representation in ASCII table.

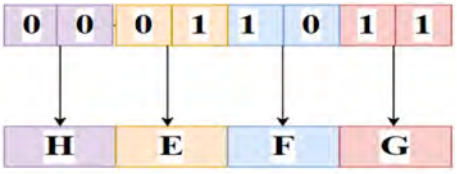

Fig. 12. Replacing each 2-bit of binaries with 1 letter.

Algorithm to convert each character into its binary format in the ASCII table (input sequence)

```
# Step 1: Initialization
  start = 0
  end = 1
  array1 = []
  array2 = []
  # Step 2: Loop to divide each ASCII character
  for i in range(len(input_sequence)):
  array1.append(input_sequence[start:end])
  start += 1
  end += 1
  # Step 3: Replace each character with its binary representation in the ASCII table
  for r in array1:
  result = ''.join(format(ord(i), '08b') for i in r)
  array2.append(result)
  # Step 4: Return sequence of zeros and ones (binary format)
  return array2
```

Pseudocode for converting each character into its binary representation in ASCII table.

Algorithm to replace each 2-bits binary with 1 letter (input sequence)

```
# Step 1: Initialization
  start = 0
  end = 2
  array1 = []
  array2 = []
  # Step 2: Loop to divide each 1-byte(8-bit) into four equal parts of 2-bits
  for i in range (0, len(input_sequence), 2):
  array1.append(input_sequence[start:end])
  start += 2
  end += 2
  # Step 3: Substitute each 2 bits with a letter ('E', 'F', 'G', 'H')
  for r in array1:
  if r == '00':
  array2.append('H')
  else if r == '01':
  array2.append('E')
  else if r == '10':
  array2.append('F')
  else if r == '11':
  array2.append('G')
  # Step 4: Return the final sequence of letters
  return array2
```

Pseudocode 5 for replacing each 2-bit of binaries with 1 letter.

3.2.5. Substitution decipher phase

Step 5, the receiver should decrypt a character sequence using the Vigenère substitution cipher algorithm with the same key letter as the one used for Step 3, result in each letter in the sequence of letters 'E', 'F', 'G', and 'H' will be switched to another sequence of 'A', 'B', 'C', and 'D' as shown in Fig. 13.

3.2.6. Composition phase

Step 6, After decrypting the sequence of 'E',' F', 'G', and 'H' into the sequence of 'A', 'B',' C', and 'D' using a Vigenère cipher algorithm, each resulting character should be transformed into a binary format consisting of 2-bits. Specifically, characters that were 'A', 'B', 'C', and 'D' will be converted into their corresponding binary values '00′, '01′, '10′, and '11′ as shown in Fig. 14 and pseudocode 6. As a result of this transformation, the size of the sequence will be reduced by approximately 25 %.

Algorithm to replace each letter into 2-bit binary (String)

```
# Input: a string of characters (input_str)
  # Output: stream of the corresponding 2-bit of binaries
  # Step 1: Initialization phase
  output_str = ""
  indices = {"A": "00″, "B": "01″, "C": "10″, "D": "11″}
  # Step 2: Iterate through each character in the input string and replace with 2-bit
  binary representation
  for char in input_str:
  output_str += indices[char]
  # Step 3: Return the output string
  return output_str
```

Pseudocode 6 for replacing each letter into a binary combination of 2-bits.

3.2.7. Decryption 1

Step 7 The receiver should use the RC4 decryption algorithm to restore the original text with the same shared key to decrypt the streams of zeros and ones and return the original message.

4. Experiment analysis

The experiment setup, datasets and software specifications will be discussed in detailed in the following sections. The proposed technique's performance is evaluated to demonstrate its superiority over other similar existing techniques. Additionally, the paper analyses the proposed technique's strength as well as its various parts concerning relevant parameters, as defined in the study.

4.1. Experiment design

The proposed approach is implemented using Python integrated with the ctypes library, a foreign function library that offers C compatible data types to allocate characters in 1 byte of memory. A Personal Computer running Windows 11 with 8.00 GB RAM and 11th Gen Intel (R) Core (TM) i7-1165G7 @ 2.80 GHz processor is used. As the standard benchmark input data, the Calgary Corpuses dataset is commonly used to assess the effectiveness of any compression technique. The proposed approach was evaluated using standard Calgary Corpuses (Witten et al., 1990) to assess its performance and various aspects including 14 files use ASCII encoding. AC, LZW, and AHC are data compression techniques known for achieving high compression efficiencies (Gupta and Nigam, 2021) and were thus used as a benchmark to compare the proposed technique's performance. Furthermore, the proposed technique's

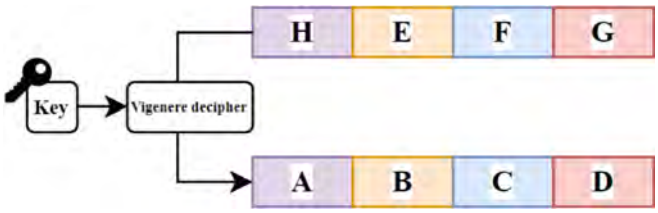

Fig. 13. Substitution decipher phase using Vigenère cipher with a key letter.

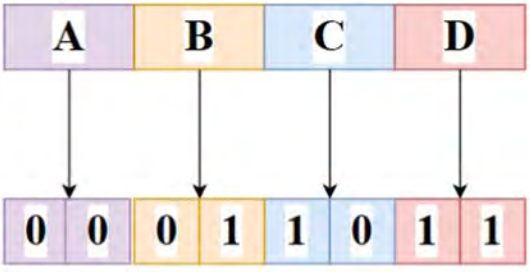

Fig. 14. Replacing each letter into a binary combination of 2-bits.

performance was evaluated by comparing it to previous data compression and encryption methods such as Chaotic Mutated Adaptive Huffman Tree (CMAHT) (Zhu et al., 2012), Chaotic Huffman Tree (CHT) (Hermassi, 2010), Chaos and Adaptive Huffman Coding (CAHC) (Usama et al., 2021), and Simultaneous Arithmetic Coding and Encryption (SACE) (Wong et al., 2010).

4.2. COMPRESSION EFFIECIENCY

The evaluation of compression techniques involves assessing their capability to reduce the size of data, using metrics such as compression efficiency, compression ratio, or space-saving capability. This evaluation approach has been documented in previous literature studies (Klein and Shapira, 2014; Usama et al., 2021; Wenfa Zhan and Aiman, 2012). According to research studies, compression techniques that result in a greater percentage of space savings are generally considered effective and efficient in terms of saving disk space and reducing transmission overheads, and this is supported by previous literature (Klein and Shapira, 2014; Usama et al., 2021; Wenfa Zhan and Aiman, 2012). In this section, we will discuss the space-saving (%) metrics to evaluate and demonstrate the efficiency of the proposed method compared to well-known compression methods and previous techniques for performing compression and encryption together using Equation. (1) (Fauzan et al., 2023).

$$\textbf{Space Saving}\ (\%) = \left(1 - \frac{\text{Compressed size}}{\text{Un} - \text{compressed size}}\right) \times 100 \quad (1)$$

In Table 2, The proposed technique's performance was compared to that of several other techniques discussed in the literature (Usama et al., 2021), including AC, LZW, AHC, CHT, CMAHT, SACE, and CAHC. Our results show that the proposed method achieves a higher range of space-saving percentage (58.63 % to 81.8 %), demonstrating a superior compression efficiency while also offering sufficient security. The security analysis section contains a more detailed analysis of the security implications.

4.3. PROECESSING TIME ANALYSIS

The processing efficiency provided by various techniques, including the proposed method is discussed in this section. In any data storage and communication system, processing time is critical and important factor. To demonstrate the performance efficiency of the proposed technique, the processing time must be measured. Processing time is calculated using Equation (2):

Processing time = Process end time − Process start time(2).

In this analysis, the effectiveness of the proposed method is compared to other existing data compression and security techniques, such as CHT, CMAHT, SACE, and CAHC, using processing times. When tested on standard Calgary Corpus input files, the proposed technique adeptly outperformed the other techniques by delivering the fastest data compression and encryption performance simultaneously as shown in Tables 3 and 4.

The standard AC and AHC methods were used to compress and encrypt the test files for the Calgary Corpus with AES.The processing time for the proposed technique, AC, and AHC with AES techniques are shown in Tables 5 and 6. The results show that the proposed method was the fastest and required the least amount of processing time when compared to performing the two operations separately.

As presented in Table 7, the proposed technique outperforms the CAHC technique discussed in literature (Usama et al., 2021), which applies compression-then-encryption. Besides, the proposed technique displays promising time efficiency as compared to the existing methods performing simultaneous compression-then-encryption or encryption-then-compression. These observations imply that the proposed method is a superior solution for reducing Overheads in terms of space and time for performing encryption-then-compression and compression-then-encryption procedures.

4.4. SECURITY ANALYSIS

The design of the proposed technique focuses on achieving high-security measures while reducing the associated costs of storing and transmitting data. In this section, through randomness tests, the security analysis of the proposed technique and its various components is

Table 3
Processing time (seconds) provided by proposed and existing methods for performing secure data compression.

File	Existing simultaneous compression and encryption techniques				Proposed Technique
	CHT	CMAHT	SACE	CAHC	
bib	393.73	988.26	218.43	137.32	16.77
Book 1	998.53	1593.58	375.88	192.33	**78.27**
Book 2	303.96	744.48	272.00	125.12	**65.96**
news	155.90	342.90	158.66	115.19	**42.9**
paper1	25.89	33.82	18.56	14.38	**11.56**
paper2	31.01	46.69	18.37	18.59	**13.4**
Paper3	20.95	23.50	33.64	12.81	**6.45**
Paper4	9.27	12.53	13.29	7.71	**3.6**
Paper5	8.17	26.89	5.16	7.48	**3.11**
Paper6	22.94	68.14	4.25	12.73	**6.31**
progc	19.01	29.80	13.71	12.83	**8.87**
progl	29.75	86.85	26.51	17.57	**12.7**
progp	22.26	26.54	20.56	14.57	**8.89**
trans	87.45	281.30	147.87	26.83	**14.9**
Average	159.27	319.57	97.03	715.96	**20.97**

Table 2
Comparing proposed and existing methods in terms of space savings (%).

	Compression techniques			Existing simultaneous compression and encryption techniques				
	AC	LZW	HC	CHT	CMAHT	SACE	CAHC	
bib	69.91	51.64	31.96	31.96	31.86	31.99	31.85	72.64
Book 1	65.87	49.15	40.46	40.46	40.41	40.03	40.41	**66.02**
Book 2	63.47	43.28	37.42	37.42	37.23	36.70	37.23	**72.18**
news	60.96	38.27	32.68	32.68	32.47	31.81	32.47	**68.58**
paper1	62.64	41.32	34.65	34.65	33.79	33.57	33.82	**67.4**
paper2	64.54	49.29	39.41	39.41	39.10	38.79	39.10	**66.83**
Paper3	37.20	48.60	38.51	38.51	38.19	37.57	38.19	**63.23**
Paper4	31.06	43.97	36.47	36.47	36.47	36.19	36.02	**58.92**
Paper5	27.46	41.24	33.46	33.46	32.42	32.86	32.42	**58.63**
Paper6	32.50	38.83	34.10	34.10	32.60	33.31	32.69	**67.07**
progc	61.39	38.21	32.12	32.12	31.30	31.41	31.35	**68.16**
progl	73.29	51.29	37.01	37.01	36.01	36.01	36.03	**79.02**
progp	72.22	52.81	35.70	35.70	34.98	36.85	34.98	**78.93**
trans	49.68	46.06	27.71	27.71	26.69	27.59	26.73	**81.8**
Average	56.58	43.92	35.05	35.12	32.82	34.45	32.87	**66.87**

Table 4
Processing time (seconds) provided by proposed and existing methods for performing secure data decompression.

File	Compression techniques		Encryption technique	Sequential compression and encryption		Proposed Technique
	AC	HC	AES	AC + AES	HC + AES	
bib	115.38	74.98	405.93	290.44	286.42	12.27
Book 1	298.13	90.40	1806.12	1424.10	1169.79	**73.02**
Book 2	197.64	47.19	1480.98	1101.37	947.99	**61.64**
news	127.66	63.98	852.26	729.25	669.72	**37.68**
paper1	17.65	4.32	121.88	100.96	81.18	**5.5**
paper2	26.03	6.79	224.20	140.99	135.36	**8.56**
Paper3	15.24	3.80	99.81	90.41	71.07	**5.27**
Paper4	7.53	3.25	30.73	27.33	25.58	**1.34**
Paper5	4.14	1.39	27.42	32.04	25.82	**1.24**
Paper6	21.11	4.54	131.73	90.22	110.73	**4.27**
progc	17.99	4.24	90.51	87.13	73.97	**4.048**
progl	26.52	7.11	171.33	157.86	130.67	**8.47**
progp	18.02	4.40	129.87	109.88	80.33	**5.37**
trans	50.98	7.55	216.06	208.55	179.55	**10.13**
Average	54.86	23.07	341.35	320.75	292.03	**17.06**

Table 5
Processing time (seconds) provided by compression, encryption, compression and encryption in a sequence technique and proposed for performing compression and encryption.

File	Compression techniques		Encryption technique	Sequential compression and encryption		Proposed technique
	AC	HC	AES	AC + AES	HC + AES	
bib	135.39	126.76	332.54	281.71	319.84	16.77
Book 1	220.51	143.80	1127.45	1007.68	823.21	**78.27**
Book 2	165.81	92.53	894.76	747.89	703.13	**65.96**
news	93.83	65.73	537.14	467.30	458.55	**42.9**
paper1	11.62	9.84	70.72	92.02	53.46	**11.56**
paper2	19.93	15.94	128.47	121.14	85.22	**13.4**
Paper3	13.64	8.46	58.85	64.05	46.89	**6.45**
Paper4	5.08	4.28	17.68	23.04	16.55	**3.6**
Paper5	3.21	3.15	15.50	26.99	20.39	**3.11**
Paper6	12.62	8.64	67.23	91.38	84.36	**6.31**
progc	9.14	7.38	48.79	50.89	40.33	**8.87**
progl	16.75	13.07	100.10	110.94	77.74	**12.7**
progp	11.90	8.94	87.32	75.79	47.95	**8.89**
trans	92.16	14.19	132.46	179.57	102.63	**14.9**
Average	58.63	37.39	254.91	255.73	212.08	**20.96**

Table 6
Processing time (seconds) provided by decompression, decryption, decompression and decryption in a sequence technique and proposed for performing decompression and decryption.

File	Compression techniques		Encryption technique	Sequential compression and encryption		Proposed technique
	AC	HC	AES	AC + AES	HC + AES	
bib	115.38	74.98	405.93	290.44	286.42	12.27
Book 1	298.13	90.40	1806.12	1424.10	1169.79	**73.02**
Book 2	197.64	47.19	1480.98	1101.37	947.99	**61.64**
news	127.66	63.98	852.26	729.25	669.72	**37.68**
paper1	17.65	4.32	121.88	100.96	81.18	**5.5**
paper2	26.03	6.79	224.20	140.99	135.36	**8.56**
Paper3	15.24	3.80	99.81	90.41	71.07	**5.27**
Paper4	7.53	3.25	30.73	27.33	25.58	**1.34**
Paper5	4.14	1.39	27.42	32.04	25.82	**1.24**
Paper6	21.11	4.54	131.73	90.22	110.73	**4.27**
progc	17.99	4.24	90.51	87.13	73.97	**4.048**
progl	26.52	7.11	171.33	157.86	130.67	**8.47**
progp	18.02	4.40	129.87	109.88	80.33	**5.37**
trans	50.98	7.55	216.06	208.55	179.55	**10.13**
Average	54.86	23.07	341.35	320.75	292.03	**17.06**

assessed.

RANDOMNESS ANALYSIS OF THE PROPOSED TECHNIQUE.

The NIST SP800-22 (AndrewRukhin, 2010) test suite is widely used to analyse the randomness of the data. This research used the NIST test suite to analyse the randomness of the proposed technique. It consists of 15 statistical tests with p-values between 0 and 1. By determining a significance level, these p-values were used to assess the randomness of the ciphertext. When $p < \alpha$, this means the ciphertext is considered non-random; otherwise, it is considered random. For this study, a significance level of $\alpha = 0.01$ was chosen to confirm a 99 % confidence level for the randomness analysis of the proposed technique. These 15 statistical tests were performed using Python, utilizing the numpy, base64, and scipy.stats libraries. numpy played a crucial role in the analysis by generating random numbers, performing various statistical operations, and creating arrays to store and manipulate the data. The base64 library was used to decode a base64-encoded string, which was then converted

Table 7
Processing time (seconds) offered by CAHC and the proposed techniques.

File	CAHC	Proposed technique
bib	137.32	**16.77**
Book 1	192.33	**78.27**
Book 2	125.12	**65.96**
news	115.19	**42.9**
paper1	14.38	**11.56**
paper2	18.59	**13.4**
Paper3	12.81	**6.45**
Paper4	7.71	**3.6**
Paper5	7.48	**3.11**
Paper6	12.73	**6.31**
progc	12.83	**8.87**
progl	17.57	**12.7**
progp	14.57	**8.89**
trans	26.83	**14.9**
Average	44.10	**20.38**

into a numpy array for further processing. This conversion was necessary to transform the encoded data into a format suitable for statistical testing. The scipy.stats library was specifically designed to perform statistical tests on the generated random data and calculate p-values. These p-values are essential in assessing the statistical significance of the results and determining if the null hypothesis should be rejected or not for each statistical test. These libraries provided the necessary functions and tools to automate the calculation of the 15 statistical tests.

The p-value calculation equations are displayed in Table 8. Table 9 shows the computed p-values for all tests using the default input parameter settings defined in the NIST statistic test suite. The results that are obtained show that the proposed technique's pass rate is 100 %, all

Table 9
NIST randomness test results.

Statistical test	Proposed technique	
	p-value	Result
Frequence	0.59875	Success
Block frequency	0.04293	Success
Cumulative sums	0.57648	Success
Runs	0.40941	Success
Long runs of one's	0.84309	Success
Binary Matrix Rank	0.65343	Success
Spectral DFT	0.56009	Success
No overlapping templates	0.29228	Success
Overlapping templates	0.65388	Success
Universal	0.69592	Success
Approximate entropy	0.85641	Success
Random excursions	0.298142	Success
Random excursions variant	0.26856	Success
Serial	0.19449	Success
Linear complexity	0.21214	Success

NIST tests were passed and thus considered secure with a 99 % confidence level.

5. Conclusion and futurework

This research utilized a multi-layered security approach with compression to secure and compress text data streams while preserving the original data sequence without any changes. This research focused on developing a secure and efficient way of storing and transferring data streams in the cloud computing environment by incorporating existing encryption and data compression techniques, including symmetric key

Table 8
P-value of NIST statistical tests equations.

Statistical test	P-Value equation	Description
Frequency	$erfc\left(\frac{s_{obs}}{\sqrt{2}}\right)$	where *erfc* is the complementary error function, s_{obs} is its test statistics.
Block frequency	igamc ($N/2x^2$(obs)/2	igamc is the incomplete gamma function, N is number of blocks, $x^2(0bs)$ is its test statistics.
Cumulative sums	$1-\sum_{k=\frac{-n}{4}+1}^{\frac{n}{4}-1}\left[\varnothing\left(\frac{(4k+1)z}{\sqrt{n}}\right)-\varnothing\left(\frac{(4k-1)z}{\sqrt{n}}\right)\right]+\sum_{k=(\frac{-n}{4}-3)/4}^{(\frac{n}{4}-1)/4}\left[\varnothing\left(\frac{(4k+3)z}{\sqrt{n}}\right)-\varnothing\left(\frac{(4k+1)z}{\sqrt{n}}\right)\right]$	Where n is the length of the bit string, z is its test statistics, and ∅ represents the cumulative distribution function (CDF)of the standard normal distribution and calculated by scipy.stats.norm. cdf() in python.
Runs	$erfc\left(\frac{\lvert V(obs)-2n\pi(1-\pi)\rvert}{2\sqrt{2n}\,\pi(1-\pi)}\right)$	$V(obs)$ is test statistics, n is the length of the bit string, π is the number of ones in the input sequence.
Long runs of one's	$\text{igamc}\left(\frac{K}{2},\frac{x^2(0bs)}{2}\right)$	Where K represents a parameter used in the computation of its test statistics, which is the maximum value of the max run for each subblock.
Binary Matrix Rank	$e^{-x^2(obs)/2}$	Where $x^2(0bS)$ is its test statistics.
Spectral DFT	$\text{erfc}\left(\frac{\lvert d\rvert}{\sqrt{2}}\right)$	d is its test statistics.
No overlapping templates	$\text{igamc}\left(\frac{N}{2},\frac{x^2(0bs)}{2}\right)$	$x^2(0bs)$ is its test statistics, N refers to the number of independent blocks into which the sequence is partitioned.
Overlapping templates	$\text{igamc}\left(\frac{5}{2},\frac{x^2(0bs)}{2}\right)$	$x^2(0bs)$ is test statistics.
Universal	$erfc\left(\frac{\lvert f_n-expectedValue(L)\rvert}{\sqrt{2}\,\sigma}\right)$	f_n is its test statistics,$expectedValue(L)$ refers to the expected value of average of the corresponding block numbers for each L-bit, σ represents the deviation of the computed statistic for the given expected value (L)
Approximate entropy	$\text{igamc}\left(2^{m-1},\frac{x^2}{2}\right)$	Where m is the length of each block,x^2 is its test statistics.
Random excursions	Igamc 5/2,x^2(Obs)/2	X^2(Obs)is its test statistics.
Random excursions variant	$erfc\left(\frac{\lvert\xi(x)-J\rvert}{2J(4\lvert x\rvert-2)}\right)$	$\xi(x)$ is its test statistics, J represents the number of cycles or repetitions of the test, x represents a state or value of $\xi(x)$
Serial	$\text{igamc}\left(2^{m-2},\nabla\psi_m^2\right)$	m is the length in bits of each block,$\nabla\psi_m^2$ is its test statistics.
Linear complexity	$\text{igamc}\left(\frac{K}{2},\frac{x^2(0bs)}{2}\right)$	$x^2(0bs)$ is its test statistics, K is a number of degrees of freedom (typically a fixed value of 6).

encryption techniques Rc4, AES, and the substitution cipher Vigenère, while AES is a more advanced encryption standard. Combining these techniques provided a more robust and comprehensive solution for data security while maintaining low storage of text data through LZMA compression. The proposed approach consists of seven stages in the encoding phase (sender Side) Initial Encryption, Decomposition, Substitution, Data size treatment, Compression, Encryption$_2$, and Uploading Encrypted-compressed data in the cloud and the inverse on the (receiver side). Experimental results proved that the proposed approach achieved faster processing time compared to performing the encryption and compression techniques as separate steps. Security analysis demonstrated that the proposed approach for generating ciphertexts passed all NIST tests with a high level of confidence (99 %) in terms of ciphertext randomness. Additionally, compression efficiency analysis demonstrates that the proposed approach produced a significantly higher range of space-saving percentages (58.63 % to 81.8 %) compared to standard techniques while also providing adequate security.

Future work will further focus on improving the proposed method and apply hybrid data encryption and compression techniques for data streams. Future research and advancements of the proposed work will not be restricted to text data but also can be applied to other data formats; including images, audio, and videos. In the future, we would like to design more sophisticated adaptive algorithms that are robust enough to secure and compress data streams. We would like to explore privacy and security schemes in the same context of data streams and experiment with several stream applications operating under different environmental constraints and having different security requirements.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Abdulameer, S.a.A., 2023. A Cryptosystem for Database Security Based on RC4 Algorithm. Journal of Al-Qadisiyah for Computer Science and Mathematics 15 (1), 189–196. https://doi.org/10.29304/jqcm.2023.15.1.1195.

Abdullah, A.M., 2017. Advanced encryption standard (AES) algorithm to encrypt and decrypt data. Cryptography and Network Security 16, 11.

Adedeji, K.B., 2020. Performance Evaluation of Data Compression Algorithms for IoT-Based Smart Water Network Management Applications. Journal of Applied Science & Process Engineering 7 (2), 554–563. https://doi.org/10.33736/jaspe.2272.2020.

Akashdeep Bhardwaj, G.V.B. Subrahmanyam, Vinay Avasthi and Hanumat Sastry "Security Algorithms for Cloud Computing, "*International Conference on Computational Modeling and Security (CMS)*, vol.85, pp. 535-542, 2016.doi: 10.1016/j.procs.2016.05.215.

Al-Amri, R. M., Hamood, D. N., & Farhan, A. K. "Theoretical Background of Cryptography, "*Mesopotamian Journal of Cyber Security*, Vol.2023, pp. 7–15, 2023. doi:10.58496/MJCS/2023/002.

Alemami, Y., Al-Ghonmein, A.M., Al-Moghrabi, K.G., Mohamed, M.A., 2023. Cloud data security and various cryptographic algorithms. International Journal of Electrical and Computer Engineering (IJECE) 13 (2), 1867–1879. https://doi.org/10.11591/ijece. v13i2.pp1867-1879.

A. M. Al-Smadi, A. Al-Smadi, R. M. Ali Aloglah, N. Abu-Darwish and A. Abugabah, "Files cryptography based on one-time pad algorithm,"International Journal of Electrical and Computer Engineering, vol. 11, no. 3, pp. 2335-2342, Jun. 2021. doi:10.11591/ijece. v11i3.pp2335-2342.

AndrewRukhin, JuanSoto,JamesNechvatal,Miles Smid,ElaineBarker,Stefan Leigh, MarkLevenson,Mark Vangel,DavidBanks,AlanHeckert,JamesDray and SanVo , "A statistical test suite for random and pseudorandom number generators for cryptographic applications, " *National Institute of Standards & Technology*,Apr.2010.

Rachna Arora and Anshu Parashar, " Secure User Data in Cloud Computing Using Encryption Algorithms," *International Journal of Engineering Research and Applications (IJERA)*, Vol. 3, no.4, pp.1922-1926, Aug.2013. doi: 9799a9f9bec6cf85715ca236035b5d89204b326a.

M. R. Ashila, N. Atikah, D. R. I. Moses Setiadi, E. H. Rachmawanto and C. A. Sari, "Hybrid AES-Huffman Coding for Secure Lossless Transmission," in 2019 Fourth International Conference on Informatics and Computing (ICIC), Semarang, Indonesia, pp. 1-5, 2019. doi:10.1109/ICIC47613.2019.8985899.

Carpentieri, B., 2018. "Efficient compression and encryption for digital data transmission," Security and Communication. Networks 2018. https://doi.org/10.1155/2018/9591768.

El-Booz, S.A., Attiya, G.M., 2017. Nawal El-Fishawy,"New Hybrid Approach for Secure Data Storage in Cloud Computing Environment,". Menoufia Journal of Electronic Engineering Research 26 (1), 193–212. https://doi.org/10.21608/mjeer.2017.63449.

Fauzan, M.N., Alif, M., Prianto3C., 2023. Comparison of Huffman Algorithm and Lempel Ziv Welch Algorithm in Text File Compression. IT Journal Research and Development 7 (2), 184–197. https://doi.org/10.25299/itjrd.2023.10437.

Gupta, A., Nigam, S., 2021. A Review on Different Types of Lossless Data Compression Techniques. International Journal of Scientific Research in Computer Science, Engineering and Information Technology 7 (1), 50–56. https://doi.org/10.32628/cseit217113.

Hermassi, H., 2010. Rhouma Rhouma and Safya Belghith,"Joint compression and encryption using chaotically mutated Huffman trees, "Communications in Nonlinear Science and Numerical. Simulation 15 (10), 2987–12299. https://doi.org/10.1016/j.cnsns.2009.11.022.

https://www.ascii-code.com/.

Ignatoski, M., Lerga, J., 2020. Ljubiša Stanković And Miloš Daković, "Comparison of Entropy and Dictionary Based Text Compression in English, German, French, Italian, Czech, Hungarian, Finnish, and Croatian,". Mathematics 8 (7), 1059–1075. https://doi.org/10.3390/MATH8071059.

R. Imanda, H. Nasution, U. Gumanti, R. G. Sinambela, B. A. Sima, C. Arista, B. Bangun, B. K. Sitepu, and A. Fauzi, "Development of Hybrid Encryption Method Using Affine Cipher, Vigenere Cipher, And Elgamal Algorithm to Secure Text Messages in Data Communication System, "*in Journal of Artificial Intelligence and Engineering Applications*, vol. 2, no. 2, pp. 51-58. Feb 2023.

Jindal, P., Singh, B., 2015. RC4 encryption-A literature survey. Procedia Computer Science 46, 697–705. https://doi.org/10.1016/j.procs.2015.02.129.

Klein, S.T., Shapira, D., 2014. "Practical fixed length Lempel-Ziv coding, "in. Discrete Applied Mathematics 163 (3), 326–333. https://doi.org/10.1016/j.dam.2013.08.022.

Kondi Uma Mahesh, Yaddanapudi Pavan, Burugapalli Kusuma Priya, Palapati Vamsi, Dr. Mohammad Shameem, Dr.M Kavitha, " A Review of the Challenges and Opportunities in Cloud Computing Services ," *Proceedings of the International Conference on Innovative Computing & Communication (ICICC)*, 2023. https://doi.org/10.2139/ssrn.4381946.

Kumar, S., Sundaresan, P., Logith, R. and Mathivanan, N., " A Data Security-based Efficient Compression and Encryption for Cloud Computing,". *In 2023 7th International Conference on Trends in Electronics and Informatics (ICOEI) IEEE*, pp. 647-653. Apr.2023.

Kuswanto, D., 2020. Cryptograph Rsa and Compression Shannon Fano Text File Services at Mobile Devices. Journal of Physics: Conference Series 1569 (2), pp. https://doi.org/10.1088/1742-6596/1569/2/022079.

Mahmoud, T.M., Abdel-latef, B.A., Ahmed, A.A., Mahfouz, A.M., 2009. Hybrid Compression Encryption Technique for Securing SMS. International Journal of Computer Science and Security (IJCSS) 3 (6), 473–478.

R. Makala, V. Bezawada, and R. Ponnaboyina, "A fast encryption and compression technique on SMS data," *International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*, pp. 1213-1217, 2017. doi:10.1109/WiSPNET.2017.8300308.

Marto Hasugian, P., Barita Nauli Simangunsong, P., Iqbal Panjaitan, M., Wahyuni, D., Fadillah Rezky, S., 2020. Combination of Cryptography Algorithm Knapsack and Run Length Enconding (RLE) Compression in Treatment of Text File. Journal of Physics: Conference Series 1573 (1). https://doi.org/10.1088/1742-6596/1573/1/012017.

Matta, P., Arora, M., Sharma, D., 2021. A comparative survey on data encryption Techniques: Big data perspective. Materials Today: Proceedings 46, 11035–11039. https://doi.org/10.1016/j.matpr.2021.02.153.

Padmapriya, M.K., Eric, P.V., 2023. Effect of Data Compression on Cipher Text Aiming Secure and Improved Data Storage. Lecture Notes in Networks and Systems Book Series 400, 195–201. https://doi.org/10.1007/978-981-19-0095-2_20.

Pavlov, I.: '7z format', [Online] Available at http://www.7-zip.org/7z.html.

Sajay, K.R., 2019. Suvanam Sasidhar Babu and Yellepeddi Vijayalakshmi, "Enhancing the security of cloud data using hybrid encryption algorithm,". Journal of Ambient Intelligence and Humanized Computing 10 (6), 2213–2221. https://doi.org/10.1007/s12652-019-01403-1.

Sangwan, N., 2012. Text Encryption with Huffman Compression. International Journal of Computer Applications 54 (6).

Sharma, R., Bollavarapu, S., 2015. Data Security using Compression and Cryptography Techniques. International Journal of Computer Applications 117, 22–25. https://doi.org/10.1007/978-981-19-0095-2_20.

Sherief, H., 2022. Murad and Kamel Hussein Rahouma,"Hybrid Cryptography for Cloud Security: Methodologies and Designs,". Digital Transformation Technology, Lecture Notes in Networks and Systems 224, 129–140. https://doi.org/10.1007/978-981-16-2275-5_7.

Shoukat, N., Azam, M., Khan, I., 2022. An Improved Method of Vigenere Cipher to Securely Compress the Text by using Relative Frequency. International Journal of Innovative Science and Research Technology 7 (10), 2456–12165.

B. Stecuła, K. Stecuła, and A. Kapczyński, "Compression of Text in Selected Languages—Efficiency, Volume, and Time Comparison, "*Sensors*, vol. 22, no. 17, pp. 6393, Aug. 2022, doi: 10.3390/s22176393.

Brandon A Sullivan , " Securing the cloud: Cloud computer security techniques and tactics," , pp.338–340, Jan.2014..

Thabit, F., Alhomdy, A.P.S., Al-Ahdal, A.H.A., Jagtap, P.D.S., 2021. A new lightweight cryptographic algorithm for enhancing data security in cloud computing. Global Transitions Proceedings 2 (1), 91–99. https://doi.org/10.1016/j.gltp.2021.01.013.

Usama, M., Malluhi, Q.M., Zakaria, N., Razzak, I., Iqbal, W., 2021. An efficient secure data compression technique based on chaos and adaptive Huffman coding. Peer-to-Peer Networking and Applications 14 (5), 2651–2664. https://doi.org/10.1007/s12083-020-00981-8.

Wenfa Zhan and Aiman El-Maleh,"A new scheme of test data compression based on equal-run-length coding (ERLC), "*Integration*, Vol 45, no 1, pp. 91-98, 2012). doi: 10.1016/j.vlsi.2011.05.001.

Witten, J., Bell, I., Calgary, C.T., 1990. Corpus. http://www.data-compression.info/Corpora/CalgaryCorpus/.

Wong, K.-W., Lin, Q., Chen, J., 2010. Simultaneous Arithmetic Coding and Encryption Using Chaotic Maps. IEEE Transactions on Circuits and Systems II: Express Briefs 57 (2), 146–150. https://doi.org/10.1109/TCSII.2010.2040315.

Z. -L. Zhu, Y. Tang, Q. Liu, W. Zhang and H. Yu, "A Chaos-based Joint Compression and Encryption Scheme Using Mutated Adaptive Huffman Tree,"*2012 Fifth International Workshop on Chaos-fractals Theories and Applications*, Dalian, China, pp. 212-216, 2012. doi: 10.1109/IWCFTA.2012.52.