EMITTER International Journal of Engineering Technology
Vol. 14, No. 1, June 2026, pp. 77~97
DOI : 10.24003/emitter.v14i1.1020

Area-Efficient Concryption for Secure Data Transmission

Thejaswini P[1], Gunasagari G S [1], Vivekananda G[2] , Sunita Shirahatti [1]

[1]JSS Academy of Technical Education, Bengaluru, 560060, India
[2]Indian Institute of Technology Guwahati, 781039, India
Corresponding Author: sunithalshirahatti@jssateb.ac.in

Received January 7, 2026; Revised March 10, 2026; Accepted June 4, 2026

Abstract

As computer networks become more connected, it is more important than ever to protect users' data and privacy. Especially because modern cyberattacks are so advanced, cryptography is a key way to keep user data private, real, and safe. However, since cryptographic methods do not reduce file sizes, concryption techniques are employed to optimize storage and enhance bandwidth for secure and efficient data transmission. The concryption method is implemented using two of the most flexible and efficient symmetric block ciphers: The concryption method uses the Blowfish and Twofish symmetric block ciphers, along with compression techniques such as LZW and MTF. We propose an area-efficient encryption algorithm that is better than the Twofish algorithm. The proposed architecture is a suitable fit for IoT devices, embedded systems, and hardware platforms that don't need much power (0.135 mW) and don't need much space (1,197 gates). Therefore, the proposed encryption architecture is a good balance between the cost of hardware, the effectiveness, and the security of lightweight cryptographic applications. The proposed encryption technique is applied to the concryption process, where compression before encryption results in a better compression ratio compared to encrypting the data first, thereby reducing bandwidth usage and enhancing transmission speed for secure data transfers.

Keywords: Cryptography, Encryption, Decryption, Concryption, Twofish, Blowfish.

1. INTRODUCTION

We live in an era where the exchange of digital data plays a crucial role in everyday communication. With advancements in computing technology, the amount of data that needs to be saved on hard drives and communicated over the internet has significantly increased. Because of these developments, information security has become one of the most important issues in the field of data communication today. For communication to be quick and safe, it is very important that data is both compressed and encrypted. To ensure quick and safe communication, compressing and encrypting data is crucial. For safe storage and transmission of information over untrusted networks, security is essential. [1].

Recent findings indicate that cybersecurity attacks expose user privacy by compromising secure algorithms and integrity, authenticity, and confidentiality of consumer data [2][3]. These attacks not only steal personal information, but they also watch what people do, which is a big concern. When it comes to security, there are many things to consider, such as making sure payments are safe, passwords are safe, and data is sent over safe channels. The increasing volume of data transfer and communication highlights the necessity of optimizing our internet bandwidth and storage capacity. To sustain both efficiency and security during data communication, it is essential to apply two critical processes: data encryption and compression [4]. "Data encryption" is a technique in which encryption and data compression are carried out one after the other [5]. Protecting data from unwanted access or while it is being transmitted over unsecure channels is the fundamental idea behind a cryptographic system [6].

The focus of this paper is on achieving secure and efficient communication by integrating compression and encryption techniques. The goal is to prevent unauthorized access to data while simultaneously reducing the file size to conserve memory during transmission. The primary objectives of this work include:

- Evaluating the performance of symmetric-key cryptographic algorithms, specifically Twofish and Blowfish.
- Assessing the performance of compression algorithms, namely LZW and MTF.
- Implementing and analyzing the concryption technique on text data.

In this paper, Section II discusses the concept of cryptography, an overview of cryptographic methods, and the Blowfish and Twofish encryption algorithms. Along with a detailed explanation of the Move-To-Front (MTF) and Lempel-Ziv (LZW) compression techniques. Section III highlights the contribution of this work. Section IV outlines the proposed method. Section V presents results and analysis. Finally, Section VI draws conclusions.

2. RELATED WORKS

Cryptography is a technique used for protecting information in computer systems and during communication. It ensures that only authorized users can access or transfer data in an encrypted manner. Typically, messages are encrypted in segments known as block ciphers. Figure 1 depicts the cryptographic procedure. The primary goals of cryptography include authentication, confidentiality, integrity, non-repudiation, service reliability, and availability [7]. cryptographic approaches are classified into three categories: symmetric key cryptography, asymmetric key cryptography, and hash functions [8].

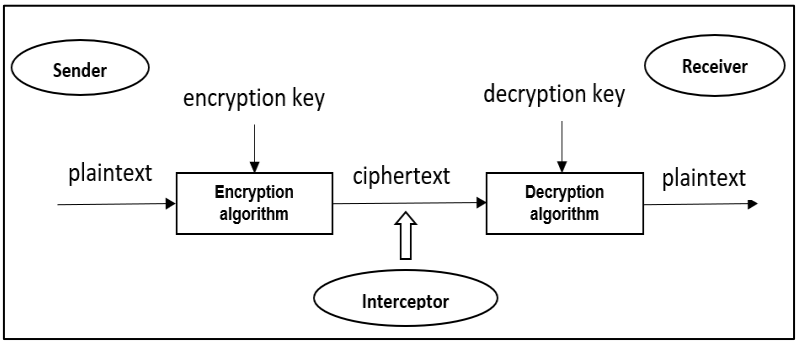

Figure 1. Cryptography process

2.1 Secret Key or Symmetric Cryptography

Secret Key Cryptography (SKC) uses a single secret key to encrypt and decrypt data. This key is securely shared between the sender and receiver. The following are some regularly used Symmetric Key Cryptography algorithms.

- Advanced Encryption Standard (AES): AES, invented by Belgian cryptographers Vincent Rijmen and Joan Daemen, use the Rijndael system with block and key lengths of 128, 192, or 256 bits. Depending on the key size, AES runs 10, 12, or 14 rounds[9].
- Data Encryption Standard (DES): DES, developed by IBM and adopted by the National Bureau of Standards in 1977, encrypts 64-bit data blocks with a 56-bit key over 16 cycles. Variants include: Triple-DES (3DES) encrypts and decrypts data using up to three 56-bit keys. Ron Rivest's DESX modification raises the key length to 120 bits by adding 64 bits before encryption [10].
- The International Data Encryption Algorithm (IDEA): IDEA operates on 64-bit data blocks and a 128-bit key. Developed in 1992 by Xuejia Lai and James Massey, it separates data into four 16-bit subblocks for encryption and decryption [11].
- Blowfish: It is a symmetric block cipher designed by Bruce Schneier in 1993. Its key length can range from 32 to 448 bits. It processes 64-bit blocks using a 16-round Feistel network with massive key-dependent S-boxes [12].
- Bruce Schneier's encryption, Twofish: It uses 128-bit data blocks with keys of 128, 192, or 256 bits. It encrypts data in 16 rounds and is designed for maximum security and efficiency, making it ideal for large microprocessors and dedicated hardware [13].

2.2 Public Key or Asymmetric Cryptography

Public Key Cryptography makes use of two mathematically related keys: a public key for encryption and a private key for decryption. Some popular PKC algorithms include:

- Rivest-Shamir-Adleman algorithm (RSA): RSA, invented by Ronald Rivest, Adi Shamir, and Leonard Adleman, encrypts data with big integers, usually 1,024 bits in size. It is commonly used in software to encrypt tiny data blocks and digital signatures[14].

- Elliptic Curve Cryptography (ECC): ECC gives comparable security to RSA while using much smaller keys. For example, ECC provides RSA-level security with a 164-bit key, making it perfect for devices with limited power and memory, such as smart cards and PDAs [15].
- Hash Functions

Hash functions convert a data of varying size in to smaller fixed size output referred as hash value or digest.

Symmetric key algorithms are designed to protect against data interception, ensuring the confidentiality of both stored data and data transmitted over networks or storage devices. For this quicker encryption and decryption processes are necessary that use shorter and more efficient algorithms. In symmetric encryption both parties use same key for data transfer. This method works especially well for encrypting long messages or big files without using up too much battery life, processing power, or memory [17]. Twofish and Blowfish are highly secure cryptographic algorithms cryptographic algorithms and are very safe and fast.

Cryptography researchers have been developing lightweight cryptographic algorithms to secure resource-constrained devices, such as IoT nodes, wireless sensor networks and embedded systems. In order to meet these new requirements, the National Institute of Standards and Technology started the Lightweight Cryptography Standardization Project. In 2023 they selected the Ascon family as the new lightweight cryptographic standard. Ascon includes secure hashing and encryption features that are meant to run on resource-constrained devices [18][19]. A more is given attention to hardware efficiency and security of lightweight cryptographic algorithms for modern embedded systems in recent years. These studies demonstrate that lightweight cryptographic designs must achieve a balance among security, speed, and hardware resource utilization, particularly in environments with limited memory, power, and processing capabilities [20][21]. The advanced cryptographic standards provide strong security assurances but, their implementations may need more memory, processing power, and hardware space than other standards. Lightweight cryptographic algorithms are made to work well in places with limited resources, such as IoT devices, wireless sensor networks, and embedded systems. They use simpler structures and efficient operations to make implementation easier. Therefore, lightweight or hybrid encryption methods are a good way to target power use, area efficiency, and processing speed as important design factors.

This study aims to provide an area-efficient encryption architecture by integrating the structural benefits of current block cipher algorithms safe data transfer and hardware-efficient implementations.

2.3 Blowfish

To encrypt and decrypt the data, Blowfish utilizes a single secret key. For both encryption and decryption, it divides the input data into blocks of 64-bit width called block ciphers [22]. The block diagram for Blowfish is depicted in Figure 2. The key expansion and data encryption mechanisms are indispensable

elements of Blowfish. The key expansion phase involves the conversion of a maximal key size of 56 bytes into a diverse array of subkeys, each of which is 32 bits in size. The data encryption process utilizes a 16-round Feistel network, which incorporates key-dependent permutations and data-dependent substitutions as shown in Algorithm 1. All operations consist of XOR operations and additions of 4-byte words [23] [24]. The functional block is depicted in Figures 2 and 3.

Algorithm 1 : Blowfish Encryption

```
Input: 64 bit Block
Output: CipherText
Start:
    1. XL, XR = Split (X, 64, 32)
    2. XL = XL xor P [1]
    3. for i = 1 to 16 do:
            F_result = F(XL, S)
            temp_XR = XR XOR F_result
            XR = XL
            XL = temp_XR
    4. XL = XL  xor   P[17]
        XR = XR  xor   P [18]
    5. CipherText = Combine (XL, XR)
    6.  return CipherText

Function F(XL, S):
  XA, XB, XC, XD = Split (XL, 32, 8)
  F_result = ((S[1][XA] + S[2][XB]) mod 2^32)  xor
S[3][XC]
  F_result = (F_result + S [4] [XD]) mod 2^32
  return F_result
```

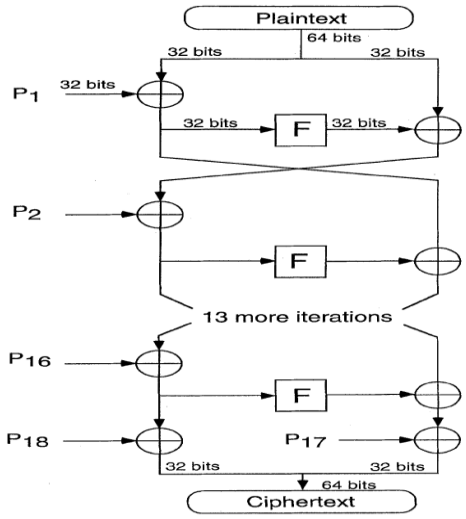

Figure 2. Block diagram of Blowfish

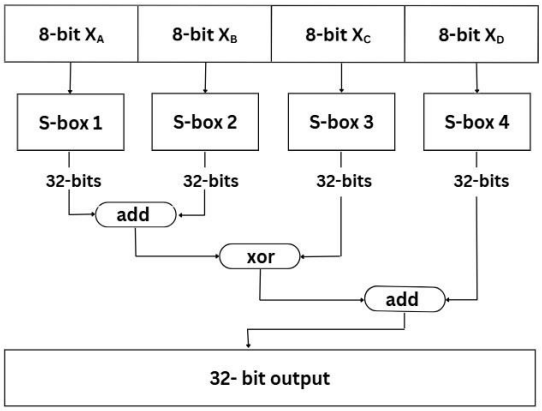

Figure 3. Functional block of Blowfish

2.4 Twofish

Twofish is a symmetric block cipher encryption algorithm. It uses blocks of 128 bits and accommodates key sizes of up to 256 bits. It is an advancement of the Blowfish block cipher [25]. Twofish employs a Feistel architecture and consists of 16 rounds, as depicted in Figure 4. The input and output data undergo XOR operations with eight subkeys, labelled K0 to K7, during the whitening process. The function F performs an 8-bit fixed left rotation, utilizes key-dependent S-boxes, employs Maximum Distance Separable (MDS) matrices, applies a Pseudo-Hadamard Transform (PHT), and incorporates two additions of subkeys modulo 2^{32}.

Four variations of key dependent S-boxes are combined with the MDS matrix and function g. The function g appears twice in the cipher structure to provide substantial redundancy. The working of different blocks of the Twofish algorithm is given at Algorithm 2.

S-Box: S-boxes carry out a nonlinear fixed substitution operation in most ciphers. The primary constituents are the fixed permutations "q_0" and "q_1.".

Q-Permutation:

Twofish's fundamental architecture includes the Q-Permutation. Functions q0 and q1 of fixed 8-bit values make up the majority of the S-boxes. The input message is divided into four 32-bit blocks. The variables R0, R1, R2, and R3 in input whitening pace are the outcomes of XORing the four sub-keys K0, K1, K2, and K3 with the four 32-bit blocks. There are 16 iterations that follow. Each iteration i of the Feistel function F uses three arguments. e. The round number r chooses the proper subkeys after two words, R0 and R1, are used as inputs. R0 is passed through the g function to yield T0. R1 is rotated eight bits to the left and then passed through the g function to yield T1. The g function consists of a linear MDS matrix after four byte-wide key-dependent S-boxes.

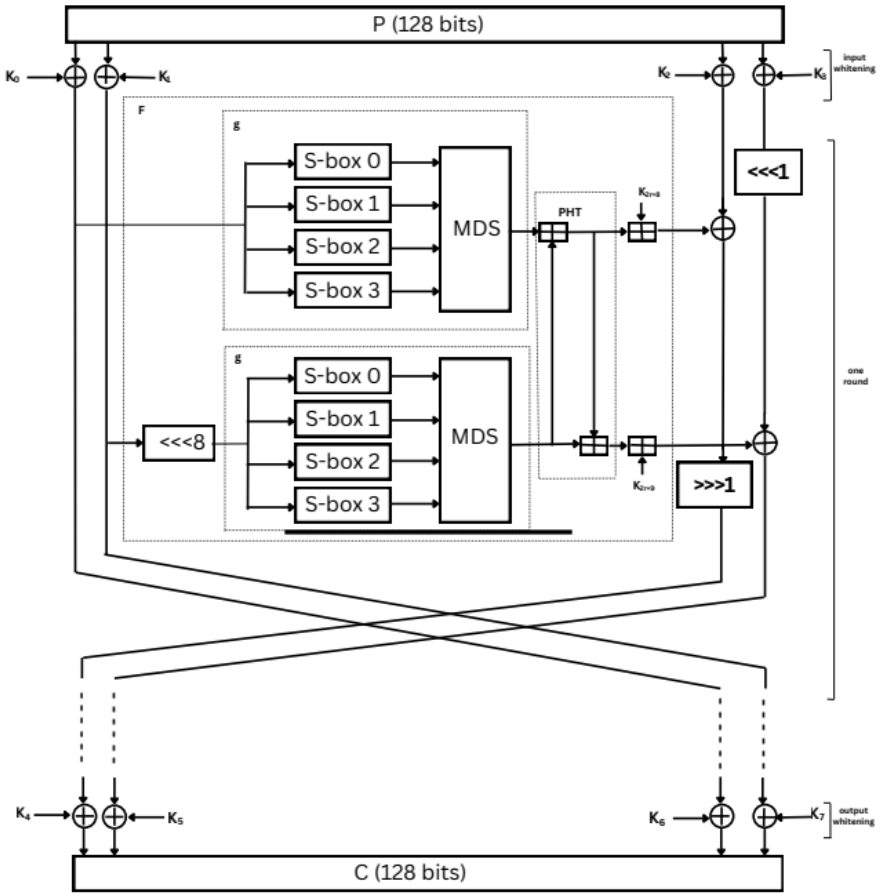

Figure 4. Block diagram of Twofish

The outputs are switched for the following round after these two results are XORed with R2 and R3 (1 bit left rotation of R2 and right rotation of R3). The ciphertext is obtained by reversing the swap and XORing it with the four sub-keys K4, K5, K6, and K7 at the end of the final round. Algorithm 2 describes the encryption process flow. The Decryption process involves reversing the order of the sub-keys.

2.5 COMPRESSION TECHNIQUES

The rapid increase in information storage and transmission capabilities of computer systems has prompted extensive research into data compression methods. Data compression algorithms make long sequences shorter by finding the best length for each symbol [26]. This improvement has made it possible to store less data and work faster, which is good for costs. It can be used in a number of ways, such as for backup and recovery and database security to boost efficiency. There are two main types of data compression: lossless and lossy. Lossless compression methods are used when a small amount of information loss is acceptable because they use approximate

representations of the content. Lossy compression is used in well-known methods like scalar and vector quantization, image and video compression.

Algorithm 2: Twofish Encryption

```
Input: plaintext: 128-bit block of plaintext to be encrypted
key: secret key (128, 192, or 256 bits)
Start:
    1. key_length = length(key)
       num_rounds = 16
       P_array, S_boxes = Key_Schedule(key)
    2.  X_L, X_R = Split (plaintext, 128, 64)
    3.  for round from 0 to num_rounds - 1 do:
          X_L = X_L  xor  P_array[2 * round]
          F_result = F(X_L, S_boxes)
          X_R = X_R  xor  F_result
        if round < num_rounds - 1 then:
          temp = X_L
          X_L = X_R
          X_R = temp
    4.  X_R = X_R  xor  P_array[1]
        X_L = X_L  xor  P_array[0]
    5.  ciphertext = Combine (X_L, X_R)
    6.  return ciphertext

Function Key_Schedule(key):
    1.  Initialize P_array with predefined values
        Initialize S_boxes with predefined values
    2. for i from 0  to length(P_array) do:
          P_array[i] = P_array[i] xor key [i mod key_length]
    3. return P_array, S_boxes

Function F(X, S_boxes):
    1. XA, XB, XC, XD = Split (X, 64, 8)
    2. Apply S-boxes
       S1 = S_boxes[0] [XA] ,  S2 = S_boxes[1] [XB]
       S3 = S_boxes[2][XC] ,  S4 = S_boxes[3] [XD]
    3. result = (S1 + S2)  xor (S3 + S4)
    4. return result
```

Lossless compression, on the other hand, lets you perfectly rebuild the original data from the compressed version. Recent works [27] have used several lossless compression methods, such as Huffman Coding, Run Length Encoding, Arithmetic Encoding, Dictionary-Based Encoding, Lempel-Zev-Welch (LZW), Burrows-Wheeler Transform (BWT), and Move-To-Front (MTF) Coding. Encoding techniques like Huffman and Arithmetic are more complex compared to MTF and LZW [28]. Both MTF and LZW are types of dictionary-based compression that use codes that are always the same size [29]. A

dictionary is used to encode the input data in this process. The MTF algorithm is an extra step in the compression process that rearranges the data to make it easier to compress. On the other hand, LZW is a simple but very effective algorithm that takes advantage of how often patterns repeat in the data. Notably, there is no requirement for prior analysis of the source, as the code table does not need to be sent to the decoder.

2.6 Move-To-Front (MTF)

One heuristic for self-organizing lists is the Move-To-Front (MTF) rule. Requests for each element in a list with N elements are handled one after the other. The corresponding element is shifted to the top of the list upon processing a new request, but the other elements' order stays the same [30]. The number of comparisons needed for a linear search is indicated by the newly requested element's position in the list. In Algorithm 3, the MTF encoding procedure is described. The source alphabets are first listed, with the first position in the list being assigned the number "0," and the remaining positions being assigned in ascending order. Every symbol is moved to the top after being encoded according to its location in the list. Long sequences of distinct symbols are efficiently compressed into 0s

Algorithm 3: Move To Front

```
Encoder:
Input: a string or list of symbols to be encoded
Start:
    Initialize :  symbol_list with all unique symbols from input in order
       Initialize output_list as an empty list
    for each symbol in input do:
          index = Find (symbol, symbol_list)
         Append index to output_list
         Remove symbol from symbol_list
         Insert symbol at the front of symbol_list
    return output_list

Algorithm MoveToFront_Decode:
Input:encoded: a list of indices corresponding to the MTF encoded
input
Start:
  1. Initialize: symbol_list with all unique symbols in original  order
       output_list as an empty list
  2. for each index in encoded do:
        symbol = symbol_list[index]
       Append symbol to output_list
       Remove symbol from symbol_list
       Insert symbol at the front of symbol_list
  3. return output_list
```

when repeated symbols are transmitted, leading to better compression results. Both the input and output streams have the same number of entries. Because it adapts to the frequency of symbols in particular regions of the input stream, the MTF method is locally flexible. This method works especially well in situations where the input stream contains groups of identical symbols.

2.7 Lempel-Ziv-Welch (LZW)

The LZW is a lossless dictionary-based compression teqnique. The dictionary is constructed as the input is processed. Figure 5 shows the block diagram of the LZW Technique.

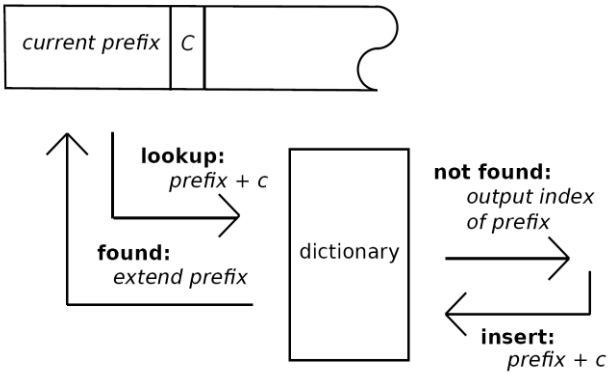

Figure 5 . Block diagram of LZW

The LZW algorithm functions on the principle that input symbols are gathered into a string. Initially, all individual characters are documented in the dictionary 'D'. In each iteration, the algorithm determines the longest prefix of the input string 'p' that is present in 'D' (i.e., $p \in D$ but $p\,c \notin D$, where 'c' denotes the following character). The concatenation of the strings 'p' and 'c' is subsequently appended to 'D', and the index of 'p' within 'D' is generated as output. The process is then repeated for the subsequent input, beginning with the character 'c'. The detailed LZW encoding is given in Algorithm 4.

The decompression procedure is initiated by utilizing the introductory entries from its dictionary. The input is analyzed, and the uncompressed symbols are extracted from the dictionary. The dictionary is constructed in a manner identical to that used in the compression process.

3. ORIGINALITY

A modified encryption algorithm combining the features of Blowfish and Twofish algorithms is proposed. Blowfish encryption has a simple Feistel structure and S-boxes, requiring a significantly smaller area and consuming low power but it lacks security aspects. It also achieves higher throughput, making it suitable for high-speed applications. However, Twofish involves complex key-dependent S-boxes and MDS transforms and is a better choice for applications where security is important. Overhead analysis of the proposed encoder is carried out in terms of area, power, delay, throughput, and energy consumption compared with Blowfish, Twofish, AES, and Ascon.

Algorithm 4: LZW encoding

```
Input: dictionary containing all possible single characters with
       their respective indices}
Start:
    1. Initialize the dictionary
       next_code = length of the dictionary + 1
       current_string = ""
       output_code_list = []
    2. for each character in input_string do:
           new_string = current_string + character
               if new_string exists in dictionary then:
                   current_string = new_string
               else:

output_code_list.append(dictionary[current_string])
           dictionary[new_string] = next_code
           next_code = next_code + 1
           current_string = character
    3.   if current_string != "":
           output_code_list.append(dictionary[current_string])
    4. return output_code_list
```

4. SYSTEM DESIGN

A modified encryption algorithm combining the features of Blowfish and Twofish algorithms is presented. It consists of several rounds of transformations that depend on the key, which include XOR operations, function processing, rotations, and swaps. The input plaintext block is partitioned into two 32-bit segments, referred as X_L and X_R, which undergo processing through the Feistel rounds. The key schedule makes 22 subkeys (K0–K21) from the secret key, similar to the Twofish algorithm. These subkeys are used during the whitening phases and round conversions. The 128-bit secret key is split into four 32-bit words, which are then combined with fixed constants using XOR and rotation operations. The round function F uses substitution and mixing to add nonlinearity. The 32-bit input word is split into four 8-bit parts, each of which goes through S-box processing, modular arithmetic, and rotation. The encryption algorithm changes the plaintext block by whitening the input, running it through 16 Feistel rounds, and whitening the output. As the cipher is based on a Feistel architecture, the same round function is used for decryption, but the subkeys are used in reverse order.

The plaintext block is first whitened with subkeys. After that, it goes through 16 Feistel rounds, during which replacement and mixing processes help spread the information and make it harder to understand. In the end, output whitening makes the ciphertext. This architecture makes the best use of hardware and cryptography, making the solution suitable for secure settings with few resources. Algorithm 5 shows how the proposed algorithm works.

**********Encryption**********

Algorithm 5A: Proposed Algorithm (Encryption)

Input: 64-bit Plaintext (P), Array of 22 subkeys (K [0...21])
Output: 64-bit Ciphertext (C)

// Phase 1: 16-Round Blowfish-style Feistel Network
1. X_L, X_R = Split_32(P) // Split into two 32-bit halves
2. For i = 0 to 15 do:
3. X_L = X_L ⊕ K[i] // XOR with round key
4. X_R = X_R ⊕ F(X_L) // Apply F-function and XOR
5. Swap (X_L, X_R) // Feistel swap
6. End For

// Phase 2: Intermediate Whitening
7. X_L = X_L ⊕ K [16]
8. X_R = X_R ⊕ K [17]

// Phase 3: Twofish-style 4-Branch Mixing
9. P0, P1 = Split_16(X_L) // Split X_L into two 16-bit blocks
10. P2, P3 = Split_16(X_R) // Split X_R into two 16-bit blocks

11. R0 = P0 ⊕ K [18] // Branch 0 whitening
12. R1 = P1 ⊕ K [19] // Branch 1 whitening

13. T0 = G_16(R0)
14. T1 = G_16(Rotate_Left_16(R1, 8)) // 8-bit circular shift on 16-bit R1
15. Mix = (T0 + T1) mod 2^16 // Pseudo-Hadamard Transform (PHT) mix

// Phase 4: Final Whitening & Output
16. Final_R0 = (P2 ⊕ Mix) ⊕ K [20]
17. Final_R1 = (P3 ⊕ Mix) ⊕ K [21]

18. C = Combine_16(Final_R0, Final_R1, R0, R1)
19. Return C

**********Decryption**********

Algorithm 5B: Proposed Algorithm (Decryption)

Input: 64-bit Ciphertext (C), Array of 22 subkeys (K [0...21])
Output: 64-bit Plaintext (P)

// Phase 1: Undo Final Whitening & 4-Branch Mixing
1. Final_R0, Final_R1, R0, R1 = Split_16(C) // Extract the 4 blocks

// Reconstruct the internal Mix variable
2. T0 = G_16(R0)
3. T1 = G_16(Rotate_Left_16(R1, 8))
4. Mix = (T0 + T1) mod 2^16

// Undo Branch 2 and 3 mixing and whitening
5. P2 = (Final_R0 ⊕ K [20]) ⊕ Mix
6. P3 = (Final_R1 ⊕ K [21]) ⊕ Mix

// Undo Branch 0 and 1 whitening
7. P0 = R0 ⊕ K [18]
8. P1 = R1 ⊕ K [19]

// Phase 2: Undo Intermediate Whitening
9. X_L = Combine_16(P0, P1)
10. X_R = Combine_16(P2, P3)
11. X_R = X_R ⊕ K [17]
12. X_L = X_L ⊕ K [16]

// Phase 3: Reverse 16-Round Feistel Network
13. For i = 15 down to 0 do:
14. Swap (X_L, X_R) // Reverse the Feistel swap
15. X_R = X_R ⊕ F(X_L) // Undo F-function XOR
16. X_L = X_L ⊕ K[i] // Undo round key XOR
17. End For

18. P = Combine_32(X_L, X_R) // Merge back to 64 bits
19. Return P

4.1 CONCRYPTION TECHNIQUE

Data can be changed into a different format by using both data compression and encryption techniques, which have different uses. The goal of data compression is to make files smaller so that they take up less memory or take less time to send. Encryption, on the other hand, protects data by changing it into a format that can't be read. The ciphertext that comes out is not easy to decode because it needs a certain piece of information called a key. In contrast to compressed data, which does not need such keys to be unpacked, it may need special hardware depending on the type of data. Combining encryption and compression methods is necessary for safe and effective

communication. This integration meets the needs for memory efficiency, security, and reliability when sending important information over an unsecured channel. The sequence in which encryption and compression are applied can influence data transmission and can be executed through two distinct approaches.

- Compression before encryption.
- Compression after encryption.

The initial approach involves compressing the input data prior to encryption. Initially, the data undergoes a compression process, and the resulting compressed output is then utilized as input for an encryption algorithm, leading to the creation of an encrypted file. Figure 6 shows the compression of input data before encryption.

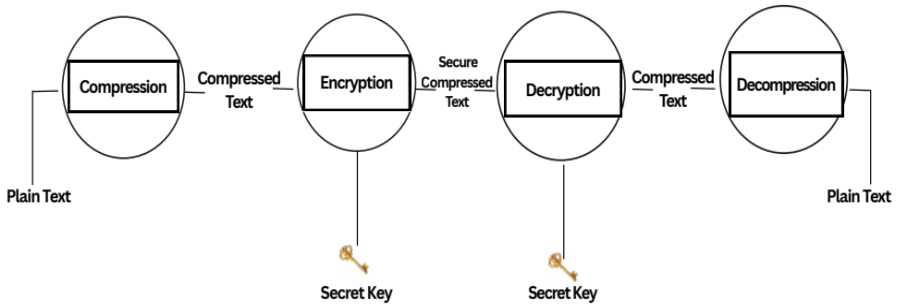

Figure 6. Block diagram of Compression of input data before encryption.

In our research, combinations of compression and encryption algorithms are incorporated to analyze the effects of compression and encryption. In the first combination, the data is first encrypted, and the resultant encrypted data is subsequently entered into the compression module to produce a compressed file. In the second combination, the data is first compressed and then the result is encrypted.

5. EXPERIMENT AND ANALYSIS

We conduct a comprehensive performance evaluation of compression and encryption techniques. The evaluation is focused on compression ratio and area utilization. Parametric analysis of the architectures in terms of area, power, throughput, and energy is conducted by synthesizing them in the Cadence Genus Tool using the TSMC90 ($90nm$) technology library. The input text file of size 50MB is considered for analysis. The compression and encryption methods are examined using different types of text data with varying characteristics.

5.1 Compression

The compression capabilities are analyzed with LZW and MTF algorithms against both varied text profiles and standard images. The text data analysis presented in Table 1 reveals that LZW consistently outperforms a basic MTF implementation on structured CSVs and natural English text,

achieving compression ratios of 17.81x and 15.95x, respectively, by effectively mapping recurring multi-character strings. The MTF pipeline exhibited extreme compression (33,333x) on zero-entropy data, demonstrating its strict reliance on long, identical byte runs and its inability to compress natural text. Table 2 presents the compression ratio results. When applied to standard image datasets, LZW algorithm achieved compression ratios > 1.0 across all samples, peaking at 1.830. The MTF transform, when paired with a basic run-length encoder, expanded the file sizes in every test, yielding ratios strictly below 1.0 (0.916 to 0.972). Because natural photographs lack long continuous sequences of identical pixel values, MTF fails to generate the zero-runs necessary for RLE. These results demonstrate that LZW serves as a capable standalone compressor.

Table 1. Compression ratio comparison for LZW and MTF for varied entropy text data.

Text Data Profile	Original Size (MB)	LZW Ratio	MTF Ratio
Zero Entropy	97.66	111.857	33333.333
Structured CSV	104.98	17.810	0.977
English Text	87.89	15.952	1.000

Table 2. Compression ratio comparison for LZW and MTF for standard images.

Standard Images	LZW Ratio	MTF Ratio
boat.png	1.189	0.972
house.png	1.327	0.939
cameraman.tif	1.830	0.916
livingroom.tif	1.209	0.965
mandril_gray.tif	1.227	0.945
fruits.png	1.464	0.946
peppers.png	1.317	0.967

Since LZW is better at compression than MTF, we consider LZW along with our proposed encryption algorithm to analyze the performance of concryption. Table 3 empirically validates the Compress-then-Encrypt (CtE) architecture over the inverse Encrypt-then-Compress (EtC) approach. The results demonstrate that EtC degrades system performance. Encrypting first maximizes data entropy and destroys the structural redundancies required for compression. CtE successfully compresses plaintext redundancies first, achieving ratios of 11.32x for text and 1.40x for images. By significantly reducing the data size prior to encryption, the CtE reduces processing latency (13.65 ms vs. 58.03 ms for text) and yields better bandwidth (3.15 MB/s vs. 0.74 MB/s).

Table 3. Performance comparison of Compress-then-Encrypt (CtE) and Encrypt-then-Compress (EtC) architectures using LZW and proposed encryption.

Dataset	Order	Orig (KB)	Ratio	Time (ms)	BW (MB/s)
English Text	CtE	43.95	11.32	13.65 ± 0.45	3.15
	EtC	43.95	2.08	58.03 ± 3.61	0.74
Cameraman Image	CtE	256.00	1.40	287.11 ± 10.64	0.97
	EtC	256.00	0.77	389.37 ± 15.16	0.64

5.2 Hardware Analysis

Table 4 presents the ASIC synthesis results of the proposed encryption algorithm compared to AES, Blowfish, Twofish, and Ascon ciphers. The proposed algorithm, along with Blowfish and Twofish, was implemented in Verilog and synthesized using the Cadence Genus tool in 90 nm technology. The proposed algorithm consumes a hardware area of 1,197 gates and 0.135 mW of power. Though the proposed algorithm has a slight increase in area and power consumption compared to Blowfish, it offers stronger cryptographic security. The proposed cipher produces a throughput of 2,396 Mbps and consumes 0.056 pJ/bit of energy, which outperforms AES. Compared to Ascon, the proposed algorithm achieves better resource utilization and area while maintaining similar energy consumption. These results confirm that the proposed algorithm successfully bridges the gap between robust security and resource efficiency.

Table 4. Hardware Implementation and Performance Comparison

Metric	Blowfish	Twofish	AES [31]	Ascon [32]	Proposed
Area (Gates)	1,039	1,312	2,297	7,080	1,197
Total Power (mW)	0.11	0.24	13.67	0.53	0.135
Delay (ns)	1.57	10.98	1.27	10.00	1.57
Frequency (MHz)	636.94	91.07	787.40	100.00	636.94
Cycles per Block	16	16	10	12	17
Block Size (bits)	64	128	128	128	64
Throughput (Mbps)	2,547	728.60	10,078	1,067	2,396
Throughput/Area (Mbps/Gate)	2.45	0.55	4.39	0.15	2
Energy per bit (pJ/bit)	0.043	0.329	1.360	0.050	0.056

5.3 Security Analysis

We considered 100K random plaintext and key pairs and performed avalanche analysis for the proposed cipher. The round-wise average, minimum, maximum, and median avalanche effect results are presented in the table. Table 5 shows that the cipher achieves the ideal 50% avalanche effect after round 3, showing strong diffusion properties. With respect to other security measures, since the proposed cipher uses the same SBOX and the function used in Blowfish and Twofish, the security resistance of the proposed cipher remains at par with those without any degradation.

Table 5. Round-wise Avalanche effect for proposed Cipher (ideal value for Avg: 50%)

Round	Avalanche Effect			
	Avg	**Min**	**Max**	**Median**
Input White	1.56	1.56	1.56	1.56
Round 1	14.17	1.56	40.62	15.62
Round 2	38.43	9.38	70.31	37.50
Round 3	50.05	26.56	71.88	50.00
Round 4	50.06	28.12	73.44	50.00
Round 5	50.09	21.88	75.00	50.00
Round 6	50.03	28.12	76.56	50.00
Round 7	50.00	26.56	71.88	50.00
Round 8	50.05	26.56	76.56	50.00
Round 9	50.08	26.56	71.88	50.00
Round 10	50.05	28.12	73.44	50.00
Round 11	50.02	25.00	71.88	50.00
Round 12	50.01	26.56	73.44	50.00
Round 13	49.97	21.88	73.44	50.00
Round 14	49.95	28.12	75.00	50.00
Round 15	50.02	29.69	71.88	50.00
Round 16	50.09	26.56	71.88	50.00
Final Mix	49.99	29.00	71.88	50.00

5.4 Limitations

One drawback of the current study is that the experimental evaluation employed a text dataset of approximately 50 MB. This dataset is sufficient for assessing the functional behaviour and initial performance of the proposed encryption and decryption methods. However, the efficiency of compression and encryption may vary, as multimedia data often has more redundancy and different entropy patterns. Therefore, an evaluation with larger datasets and a variety of data types is necessary to validate the scalability and general applicability of the proposed strategy in multimedia communication systems.

The effectiveness of the proposed method in real-time communication environments and resource-constrained IoT devices needs to be investigated. To improve practical secure communication applications having better area and power and more throughput, design of efficient hardware with advanced technology nodes should be carried out.

6. CONCLUSION

In this paper, we evaluate compression and encryption techniques using various types of text data with different characteristics. We propose a new encryption algorithm, which is a variant of the Twofish/Blowfish algorithms. From the results discussed, AES has the best throughput of the algorithms we looked at, but it uses a lot more power and needs a lot more hardware resources, making it less appropriate for contexts with very limited resources. On the other hand, the suggested architecture is designed to put hardware efficiency and low power use first. This means that it can handle data at a rate similar to that of lightweight ciphers like Blowfish, but it needs far less hardware than AES and Ascon. The proposed architecture is a good fit for IoT devices, embedded systems, and hardware platforms that don't need much power (0.135 mW) and don't need much space (1,197 gates). So, the proposed encryption architecture is a good balance between the cost of hardware, the effectiveness, and the security of lightweight cryptographic applications. The suggested encryption architecture shows that lightweight cryptographic solutions can strike a good balance between security, hardware efficiency, and energy use without needing complicated hardware. Results show that lightweight cryptographic solutions can efficiently balance security, hardware efficiency, and energy usage without using complicated hardware. Additionally, it is found that applying compression before encryption is more efficient, resulting in reduced bandwidth usage and improved transmission speed.

REFERENCES

[1] A. S. Ansari, **A review on the recent trends of image steganography for VANET applications**, *Computers, Materials & Continua*, vol. 78, no. 3, pp. 2865–2892, 2024. DOI: 10.32604/cmc.2024.045908.

[2] J. K. Liu and P. Samarati, eds., **Information security practice and experience: 13th International Conference, ISPEC 2017, Melbourne, VIC, Australia, December 13–15, 2017, proceedings**, *Lecture Notes in Computer Science*, vol. 10701, Springer, 2017. DOI: 10.1007/978-3-319-72359-4.

[3] M. F. Mushtaq, S. Jamel, A. H. Disina, Z. A. Pindar, N. S. A. Shakir, and M. M. Deris, **A survey on the cryptographic encryption algorithms**, *International Journal of Advanced Computer Science and Applications*, vol. 8, no. 11, pp. 333–344, 2017. DOI: 10.14569/IJACSA.2017.081141.

[4] B. Carpentieri, **Efficient compression and encryption for digital data transmission**, *Security and Communication Networks*, vol. 2018, p. 9591768, 2018. DOI: 10.1155/2018/9591768.
[5] J. Ye, R. Zhang, M. Zhong, and Z. Zhang, **Design of data encryption and compression methods**, *Procedia Computer Science*, vol. 243, pp. 1257–1264, 2024. DOI: 10.1016/j.procs.2024.09.148.
[6] D. K. Sharma, N. C. Singh, D. A. Noola, A. N. Doss, and J. Sivakumar, **A review on various cryptographic techniques & algorithms**, *Materials Today: Proceedings*, vol. 51, pp. 104–109, 2022. DOI: 10.1016/j.matpr.2021.04.583.
[7] B. A. Buhari *et al.*, **Performance and security analysis of symmetric data encryption algorithms: AES, 3DES and Blowfish**, *International Journal of Advanced Networking and Applications*, vol. 16, no. 4, pp. 6473–6486, 2025. DOI: 10.35444/IJANA.2025.16404.
[8] J. Y. I. Al-Zamily, S. B. Ariffin, and S. S. Abu-Naser, **A survey of cryptographic algorithms with deep learning**, *AIP Conference Proceedings*, vol. 2808, no. 1, p. 050002, 2023. DOI: 10.1063/5.0133509.
[9] M. E. Smid, **Development of the Advanced Encryption Standard**, *Journal of Research of the National Institute of Standards and Technology*, vol. 126, p. 126024, 2021. DOI: 10.6028/jres.126.024.
[10] J. Z. E. Basco, A. F. Romualdo, J. A. Saluta, C. S. Santiago Jr., and R. M. Marfil, **Comparison of Data Encryption Standard (DES) and Advanced Encryption Standard (AES) in security issues of cloud computing: A literature review**, in *Innovations in Information and Decision Sciences*, pp. 189–200, Springer, 2025. DOI: 10.1007/978-981-96-0147-9_16.
[11] S. Mohammed, S. Nanthini, N. B. Krishna, I. V. Srinivas, M. Rajagopal, and M. A. Kumar, **A new lightweight data security system for data security in the cloud computing**, *Measurement: Sensors*, vol. 29, p. 100856, 2023. DOI: 10.1016/j.measen.2023.100856.
[12] S. Ilasariya, P. Patel, V. Patel, and S. Gharat, **Image steganography using Blowfish algorithm and transmission via Apache Kafka**, *2022 4th International Conference on Smart Systems and Inventive Technology (ICSSIT)*, pp. 1320–1325, 2022. DOI: 10.1109/ICSSIT53264.2022.9716292.
[13] A. A. Nurdin and D. Djuniadi, **Securing audio chat with Cryptool-based Twofish algorithm**, *Journal of Soft Computing Exploration*, vol. 3, no. 1, pp. 37–43, 2022. DOI: 10.52465/joscex.v3i1.65.
[14] M. Melina *et al.*, **Digital signature authentication using Rivest-Shamir-Adleman cryptographic algorithm**, *AIP Conference Proceedings*, vol. 2867, no. 1, p. 020011, 2024. DOI: 10.1063/5.0225078.
[15] A. E. Adeniyi, R. G. Jimoh, and J. B. Awotunde, **A systematic review on elliptic curve cryptography algorithm for Internet of Things: Categorization, application areas, and security**, *Computers & Electrical Engineering*, vol. 118, p. 109330, 2024. DOI: 10.1016/j.compeleceng.2024.109330.

[16] A. Mittelbach and M. Fischlin, **The theory of hash functions and random oracles: An approach to modern cryptography**, *Information Security and Cryptography*, Springer, 2021. DOI: 10.1007/978-3-030-63287-8.

[17] D. K. Sharma, N. C. Singh, D. A. Noola, A. N. Doss, and J. Sivakumar, **A review on various cryptographic techniques & algorithms**, *Materials Today: Proceedings*, vol. 51, pp. 104–109, 2022. DOI: 10.1016/j.matpr.2021.04.583.

[18] J. Kaur, A. C. Canto, M. M. Kermani, and R. Azarderakhsh, **A comprehensive survey on the implementations, attacks, and countermeasures of the current NIST lightweight cryptography standard**, *arXiv preprint arXiv:2304.06222*, 2023. DOI: 10.48550/arXiv.2304.06222.

[19] M. S. Turan, K. McKay, D. Chang, J. Kang, and J. Kelsey, **Ascon-based lightweight cryptography standards for constrained devices: Authenticated encryption, hash, and extendable output functions**, *NIST Special Publication 800-232, Initial Public Draft*, 2024. DOI: 10.6028/NIST.SP.800-232.ipd.

[20] A. Patel and A. K. Cherukuri, **Analysis of light-weight cryptography algorithms for UAV-networks**, *arXiv preprint arXiv:2504.04063*, 2025. DOI: 10.48550/arXiv.2504.04063.

[21] A. Vahi, **Key length-oriented classification of lightweight cryptographic algorithms for IoT security**, *arXiv preprint arXiv:2512.21368*, 2025. DOI: 10.48550/arXiv.2512.21368.

[22] S. Sharma, K. N. Patel, and A. S. Jha, **Cryptography using Blowfish algorithm**, *2021 3rd International Conference on Advances in Computing, Communication Control and Networking (ICAC3N)*, pp. 1375–1377, 2021. DOI: 10.1109/ICAC3N53548.2021.9725661.

[23] A. E. Adeniyi, S. Misra, E. Daniel, and A. Bokolo Jr., **Computational complexity of modified Blowfish cryptographic algorithm on video data**, *Algorithms*, vol. 15, no. 10, p. 373, 2022. DOI: 10.3390/a15100373.

[24] H. Alabdulrazzaq and M. N. Alenezi, **Performance evaluation of cryptographic algorithms: DES, 3DES, Blowfish, Twofish, and Threefish**, *International Journal of Communication Networks and Information Security*, vol. 14, no. 1, 2022. DOI: 10.17762/ijcnis.v14i1.5262.

[25] A. H. Saieed and A. A. Hattab, **Modifications and improvements to the Twofish encryption algorithm: A review**, *AIP Conference Proceedings*, vol. 2834, no. 1, p. 050007, 2023. DOI: 10.1063/5.0161502.

[26] K. L. Ketshabetswe, A. M. Zungeru, B. Mtengi, C. K. Lebekwe, and S. R. S. Prabaharan, **Data compression algorithms for wireless sensor networks: A review and comparison**, *IEEE Access*, vol. 9, pp. 136872–136891, 2021. DOI: 10.1109/ACCESS.2021.3116311.

[27] U. Jayasankar, V. Thirumal, and D. Ponnurangam, **A survey on data compression techniques: From the perspective of data quality,**

coding schemes, data type and applications, *Journal of King Saud University – Computer and Information Sciences*, vol. 33, no. 2, pp. 119–140, 2021. DOI: 10.1016/j.jksuci.2018.05.006.

[28] G. Vivekananda and P. Thejaswini, **Extended Base-Delta compression technique for on-chip data transfer**, *2021 IEEE 18th India Council International Conference (INDICON)*, pp. 1–7, 2021. DOI: 10.1109/INDICON52576.2021.9691570.

[29] C. Rodrigues, E. M. Jishnu, C. R. Nair, and M. Soumya Krishnan, **A study on data compression algorithms for its efficiency analysis**, in *Ubiquitous Intelligent Systems*, Smart Innovation, Systems and Technologies, vol. 243, pp. 475–488, Springer, 2022. DOI: 10.1007/978-981-16-3675-2_36.

[30] Baisakh, H. Mohapatra, and A. Guru, **Behavioral insights into compression: A study of Move-to-Front-or-Middle deterministic online algorithm through sequence classification and characterization**, *International Journal of Information Technology*, vol. 17, no. 1, pp. 189–203, 2025. DOI: 10.1007/s41870-024-02218-w.

[31] R. Swann and J. Stine, **Evaluation of a modular approach to AES hardware architecture and optimization**, *Journal of Signal Processing Systems*, vol. 95, pp. 797–813, 2023. DOI: 10.1007/s11265-022-01832-w.

[32] A. Kandi *et al.*, **Hardware implementation of ASCON**, *NIST Lightweight Cryptography Workshop 2023*, Virtual, 2023.