Indian Journal of Cryptography and Network Security (IJCNS)
ISSN: 2582-9238 (Online), Volume-4 Issue-2, November 2024

A Comprehensive Methodology for Data Compression and Decompression Utilizing Huffman Coding, LZW Compression, and Run-Length Encoding, Integrated with Data Encryption Standard (DES) and Advanced Encryption Standard (AES) for Enhanced Security

Sangeen Khan

Abstract: ***With advancements in communication technologies, transitioning from 5G to 6G systems has led to exponential data growth, requiring secure and efficient data transmission solutions. This study integrates data compression techniques—Huffman Coding, Lempel-Ziv-Welch (LZW), and Run-Length Encoding (RLE)—with symmetric encryption algorithms, AES (Advanced Encryption Standard) and DES (Data Encryption Standard). The primary goal is to enhance computational performance while ensuring data security. Using a 32-byte dataset and implementing the algorithms in Go language via Visual Studio IDE, results demonstrate the significant reduction in encryption time when combining compression and encryption. Among the AES combinations, AES with Huffman Coding showed the highest efficiency, reducing encryption time by approximately 15% compared to standalone AES. Similarly, DES paired with LZW compression achieved a 20% improvement in computational time over standalone DES. The findings emphasize that selecting the optimal combination depends on data type and user requirements, facilitating secure and efficient communication in high-bandwidth, low-latency 6G systems. This research underscores the potential of cryptography, combined with compression, to enhance data transmission efficiency without compromising security. The integration approach highlights cryptographic strength in safeguarding big data, addressing challenges in modern technologies like the Internet of Everything (IoE). These results establish a foundation for future secure communication frameworks, promoting reliable and scalable cryptographic solutions tailored for 6G and beyond..***

Keywords: ***6G, AES, Go Language, Computational Performance, Big data.***

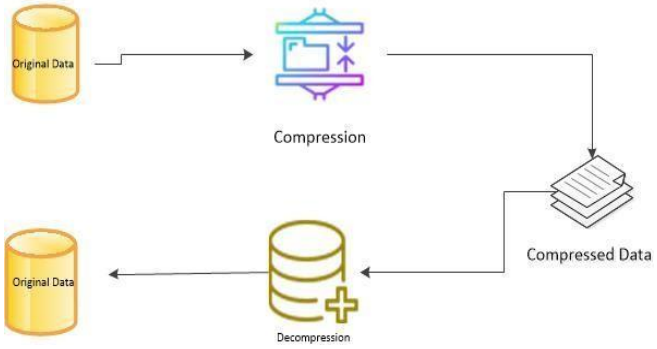
I. INTRODUCTION

The number of devices (sensors) connected is increasing very rapidly daily, and it is now the time of the Internet of Everything (IoE) instead of IoT.

Manuscript received on 20 February 2024 | Revised Manuscript received on 24 October 2024 | Manuscript Accepted on 15 November 2024 | Manuscript published on 30 November 2024.

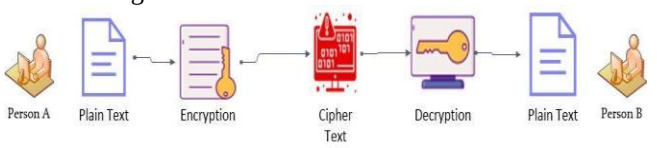
***** Correspondence Author(s)
Sangeen Khan*, Department of Communication Engineering University of Science and Technology Beijing China. Email ID: Sangeenkhan2662@gmail.com, ORCID ID: 0000-0003-4193-9340

For such approaches, high throughput, less latency, and vast coverage are required. 6G is the optimal candidate for achieving all these characteristics to have effective communication. The increase in the number of connected devicesmeans a huge volume of data over the internet, which will result in various security issues. It needs time to bring efficient and secure communication paradigms for the smooth and reliable working of Artificial Intelligence (AI)-based smart systems. The practice of expressing certain material while using fewer data to do so is generally referred to as data compression. The method of encoding the provided information in fewer bits is known as data compression. In the present era of communication, it is playing a very important function [1]. The process of data compression can be seen in Figure 1.



[Fig.1: Data Compression][1]

Encryption involves transforming plain text (or any other type of data) into cipher text. It isextremely crucial for safe communication. Symmetric or Asymmetric encryption (separate keys for encryption and decryption) is both possible. The original data will only be accessible to the intended recipient [1]. The overall encryption process is shown in Figure 2.



[Fig.2: Encryption][1]

A. Huffman Coding

Huffman coding provides a more advanced and effective lossless compression method that converts symbols in a data source to binary codes, resulting in the most common letters generating the shortest binary codes and the least common having the largest [2]. The main steps involved inthe process of Huffman coding are given in algorithm 1.

Algorithm_1 Huffman_Coding

```
package main

import (
	"container/heap"
	"fmt"
	"io/ioutil"
	"log"
	"time"
)

type HuffmanTree interface {
	Freq() int
}

type HuffmanLeaf struct {
	freq  int
	value rune
}

type HuffmanNode struct {
	freq	int
	left, right HuffmanTree
}

func (self HuffmanLeaf) Freq() int {
	return self.freq
}

func (self HuffmanNode) Freq() int {
	return self.freq
}

type treeHeap []HuffmanTree

func (th treeHeap) Len() int { return len(th) }
func (th treeHeap) Less(i, j int) bool {
	return th[i].Freq() < th[j].Freq()
}
func (th *treeHeap) Push(ele interface{}) {
	*th = append(*th, ele.(HuffmanTree))
}
func (th *treeHeap) Pop() (popped interface{}) {
	popped = (*th)[len(*th)-1]
	*th = (*th)[:len(*th)-1]
	return
}
func (th treeHeap) Swap(i, j int) { th[i], th[j] = th[j], th[i] }

func buildTree(symFreqs map[rune]int) HuffmanTree {
	var trees treeHeap
	for c, f := range symFreqs {
		trees = append(trees, HuffmanLeaf{f, c})
	}
	heap.Init(&trees)
	for trees.Len() > 1 {

		a := heap.Pop(&trees).(HuffmanTree)
		b := heap.Pop(&trees).(HuffmanTree)

		heap.Push(&trees, HuffmanNode{a.Freq()
+ b.Freq(), a, b})
	}
	return heap.Pop(&trees).(HuffmanTree)
}

func printCodes(tree HuffmanTree, prefix []byte) {

	switch i := tree.(type) {
	case HuffmanLeaf:
		fmt.Printf("%c\t%d\t%s\n", i.value, i.freq,
string(prefix))
	case HuffmanNode:
		prefix = append(prefix, '0')
		printCodes(i.left, prefix)
		prefix = prefix[:len(prefix)-1]
		prefix = append(prefix, '1')
		printCodes(i.right, prefix)
		prefix = prefix[:len(prefix)-1]
	}
}

func main() {
	fmt.Printf("\n\nReading a file in Go lang\n")
	fileName := "test.txt"

	data, err := ioutil.ReadFile("test.txt")
	if err != nil {
		log.Panicf("failed reading data from file:
%s", err)
	}
	fmt.Printf("\nFile Name: %s", fileName)
	fmt.Printf("\nSize: %d bytes", len(data))
	fmt.Printf("\nData: %s", data)
	var test string = string(data)
	symFreqs := make(map[rune]int)

	for _, c := range test {
		symFreqs[c]++
	}

	exampleTree := buildTree(symFreqs)

	fmt.Println("SYMBOL\tWEIGHT\tHUFFMAN
CODE")
	start := time.Now()
	printCodes(exampleTree, []byte{})
	elapsed := time.Since(start)
	fmt.Println(elapsed)

}
```

B. LZW Compression

Abraham Lempel, Jacob Ziv, and Terry Welch developed the global lossless data compression technique known as Lempel-Ziv-Welch (LZW). One of the Adaptive Dictionary approaches is LZW compression. The dictionary is constructed concurrently with the encoding of the data. Therefore, on-the-fly encoding is possible. It's not necessary to convey the dictionary. At the receiving end, a dictionary may be created on the spot. If the dictionary becomes overflowing, wemust re-initialize the dictionary and slightly expand each of the code words [2]. The main procedure of LZW compression is given in algorithm 2.

Algorithm_2 LZW_Compression

```
package main

import (
	"fmt"
	"io/ioutil"
	"log"
	"strconv"
	"strings"
	"time"
)

func compressLZW(testStr string) []int {
	code := 256
	dictionary := make(map[string]int)
	for i := 0; i < 256; i++ {
```

```
                dictionary[string(i)] = i
        }

        currChar := ""
        result := make([]int, 0)
        for _, c := range []byte(testStr) {

phrase := currChar + string(c)
                if _, isTrue := dictionary[phrase]; isTrue {
                        currChar = phrase
                } else {
                        result = append(result,
dictionary[currChar])
                        dictionary[phrase] = code
                        code++
                        currChar = string(c)
                }

        if currChar != "" {
                result = append(result, dictionary[currChar])
        }
        return result
}

func decompressLZW(compressed []int) string {
        code := 256
        dictionary := make(map[int]string)
        for i := 0; i < 256; i++ {
                dictionary[i] = string(i)
        }

        currChar := string(compressed[0])
        result := currChar
        for _, element := range compressed[1:] {
                var word string
                if x, ok := dictionary[element]; ok {
                        word = x
                } else if element == code {
                        word = currChar + currChar[:1]
                } else {
                        panic(fmt.Sprintf("Bad compressed
element: %d", element))
                }

                result += word

                dictionary[code] = currChar + word[:1]
                code++

                currChar = word
        }
        return result
}

func main() {

        fmt.Printf("\n\nReading a file in Go lang\n")
        fileName := "test.txt"

        data, err := ioutil.ReadFile("test.txt")
        if err != nil {
                log.Panicf("failed reading data from file: %s", err)
        }
        fmt.Printf("\nFile Name: %s", fileName)
        fmt.Printf("\nSize: %d bytes", len(data))
        fmt.Printf("\nData: %s", data)
        var test string = string(data)
        start := time.Now()
        compressed := compressLZW(test)
        fmt.Println("\nAfter Compression :", compressed)
        elapsed := time.Since(start)
        fmt.Println(elapsed)
        cod := IntToString1(compressed)
        fmt.Println("required string is", cod)

        uncompression := decompressLZW(compressed)
        fmt.Println("\nAfter Uncompression :", uncompression)
}
func IntToString1(a []int) string {
        b := ""
        for _, v := range a {
                if len(b) > 0 {
                        b += ","
                }
                b += strconv.Itoa(v)
        }

        return strings.Replace(b, ",", "", -1)
}
```

C. Run Length Encoding

Run length encoding (RLE) reduces the size of a sequence of characters by more effectively resembling a subsequence made up of runs of the same character. A tuple that includes the start and outcome of a run is substituted for it. The beginning is the position of the substring's first item, and the value is the letter that appears there [3]. The process of RLE is explained in algorithm 3.

Algorithm_3 Runlength_Encoding

```
import timeit
def printRLE(st):

        n = len(st)
        i = 0
        while i < n- 1:
                count = 1
                while (i < n - 1 and
                        st[i] == st[i + 1]):
                        count += 1
                        i += 1
                i += 1

                print(st[i - 1] +
                        str(count),
                        end = "")
if __name__ == "__main__":
        st = "IamgoingtoChinaIloveBeijingChina"
        start_time = timeit.default_timer()
        printRLE(st)
et = timeit.default_timer()
print("The time of execution of above program is :",
    (et-start_time) * 10**3, "ms")
```

D. AES Encryption

AES is a symmetric block encryption algorithm with a defined block size that's employed to safeguard private data. AES uses 10, 12, and 14 encryption repetitions with key sizes of 128, 192, and 256 bits supported. A round key obtained from the encryption key is mixed with the data during each round [4]. The architecture of AES encryption is shown in algorithm 4.

Algorithm_4 AES_Encryption

```
package main

import (
        "crypto/aes"
        "encoding/hex"
        "fmt"
        "io/ioutil"
        "log"
        "time"
)

func main() {
```

```
	key := "thisis32bitlongpassphraseimusing" // key for
AES
	// Reading the input file
	fmt.Printf("\n\nReading a file in Go lang\n")
	fileName := "test4.txt"

	data, err := ioutil.ReadFile("test.txt")
	if err != nil {
		log.Panicf("failed reading data from file:
%s", err)
	}
	fmt.Printf("\nFile Name: %s", fileName)
	fmt.Printf("\nSize: %d bytes", len(data))
	fmt.Printf("\nData: %s", data)
	var test string = string(data)
	// plaintext
	//pt := "This is a secret"
	//pt := test
	c := EncryptAES([]byte(key), test)

	// plaintext
	fmt.Println(test)

	// ciphertext
	fmt.Println(c)

	// decrypt
	DecryptAES([]byte(key), c)
}
func timeTrack(start time.Time, name string) {
	time.Sleep(time.Second)
	elapsed := time.Since(start)
	fmt.Printf("%s took %s \n", name, elapsed)
}
func EncryptAES(key []byte, plaintext string) string {
	defer timeTrack(time.Now(), "encryption")
	//start := time.Now()
	c, err := aes.NewCipher(key)
	CheckError(err)
	out := make([]byte, len(plaintext))
	c.Encrypt(out, []byte(plaintext))
	//elapsed := time.Since(start)
	//elapsed := time.Since(start)
	//fmt.Printf("Sum function took %s", elapsed)
	return hex.EncodeToString(out)
}
func DecryptAES(key []byte, ct string) {
	ciphertext, _ := hex.DecodeString(ct)
	c, err := aes.NewCipher(key)
	CheckError(err)
	pt := make([]byte, len(ciphertext))
	c.Decrypt(pt, ciphertext)
	s := string(pt[:])
	fmt.Println("DECRYPTED:", s)
}
func CheckError(err error) {
	if err != nil {
		panic(err)
	}
}
```

E. DES Encryption

A common private key is used by the cryptosystem DES to encrypt and decode data. DES method applies a predetermined piece of sequence in plaintext bits and encodes it by performing several steps into cipher text of identical size and every block is 64 bits. There are 16 similar operating cycles or steps. There is also a first and last permutation, denoted by the letters IP and FP, respectively [5]. Algorithm 5 explains the DES encryption system.

Algorithm_5 DES_Encryption

```
package main
import (
	"crypto/cipher"
	"crypto/des"
	"fmt"
	"io/ioutil"
	"log"
	"os"
	"time"
)
func main() {
	triplekey := "12345678" + "12345678" + "12345678"
	// Reading the input file
	fmt.Printf("\n\nReading a file in Go lang\n")
	fileName := "test4.txt"

	data, err := ioutil.ReadFile("test.txt")
	if err != nil {
		log.Panicf("failed reading data from file: %s", err)
	}
	fmt.Printf("\nFile Name: %s", fileName)
	fmt.Printf("\nSize: %d bytes", len(data))
	fmt.Printf("\nData: %s", data)
	var test string = string(data)
	plaintext := []byte(test)
	block, err := des.NewTripleDESCipher([]byte(triplekey))

	if err != nil {
		fmt.Printf("%s \n", err.Error())
		os.Exit(1)
	}
	fmt.Printf("%d bytes NewTripleDESCipher key with block
size of %d bytes\n", len(triplekey), block.BlockSize)
	ciphertext := []byte("abcdef1234567890")
	iv := ciphertext[:des.BlockSize] // const BlockSize = 8
	// encrypt
	mode := cipher.NewCBCEncrypter(block, iv)
	defer timeTrack(time.Now(), "encryption")
	encrypted := make([]byte, len(plaintext))

	mode.CryptBlocks(encrypted, plaintext)
	fmt.Printf("%s encrypt to %x \n", plaintext, encrypted)
	//decrypt
	decrypter := cipher.NewCBCDecrypter(block, iv)
	decrypted := make([]byte, len(plaintext))
	decrypter.CryptBlocks(decrypted, encrypted)
	fmt.Printf("%x decrypt to %s\n", encrypted, decrypted)
}
// Time calculation
func timeTrack(start time.Time, name string) {
	time.Sleep(time.Second)
	elapsed := time.Since(start)
	fmt.Printf("%s took %s", name, elapsed)
}
```

The main aim of the proposed study is to discuss the efficiency in the computational performance using the reliable combination of compression encryption. The study also aims to:

- To discuss the existing literature related to the area of research.
- To obtain statistical data about various compression encryption techniques.
- To realize how to select an effective combination.

The article is structured as follows: The project idea is summarized in Section 1. Section 2 providesa summary of the methods employed to examine the case.

Section three of the report goes into further depth on the specifics of how it was used. The main subject of Section 4 is the investigation's findings. Section 5 examines the full body of research.

II. RELATED WORK

Carpentieri [5] has investigated the use of both compression and encryption on digital records. To be effective and secure, communication should be built on a scheme defined as two activities that are diverse and occasionally at odds with one another.

Compression and cryptography are these two procedures. The adversary of compression is unpredictability, while on the other hand, encryption has to infuse randomization into the electronic data to ensure protection.

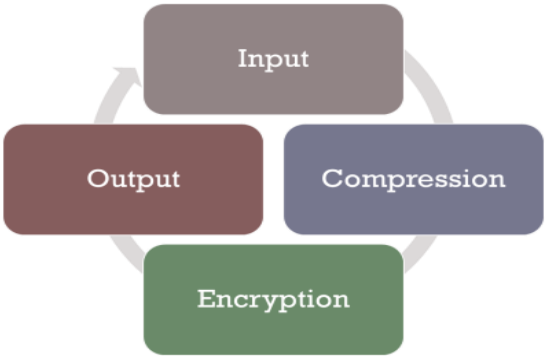
Poor security, ineffective picture propagation, and inefficient image storage are becoming major issues. Tong et al. [6] have presented a brand-new picture lossless compression coupled encryption technique utilizing chaotic maps with all source data unaltered. According to empirical results, the reduced data size is around 50% of the actual file size, achieving a respectable lossless compression ratio. Additionally, the encryption system satisfies several safety checks. For large datasets, the privacy of electronic patient records (EPR) is a major problem. The EPR information for the hospital setting was protected using a compression-then-encryption-based dual watermarking technology, which results in many noteworthy characteristics. The potential of the suggested strategy for telemedicine has been demonstrated through trials on a sizable quantity of patient records. Furthermore, the suggested technique is superior in terms of resilience and reliability when compared to the current approaches [7]. Hameed et al. [8] have suggested a method that allowssmooth and encrypted transfer of the Ecg waveform from the detector to the screen utilizing buffer blocks, peak detection, compression, and encryption mechanisms. It was discovered that the suggested system's discrete wavelet transform, Huffman coding, and Cipher Block Chaining- Advanced Encryption Standard algorithm could provide rebuilt waveforms of a top standard thanthose produced by unencrypted compression.

One of the best methods for protecting information when saving and delivering it over a network connection is encryption. Ashila et al. [9] have suggested an approach utilizing AES-Huffman in conjunction to create lossless compressed encrypted documents. when the compression step is carried out after encryption. Entropy following encryption and compression and the avalanche effect (AE) were used to determine the degree of file privacy. According to the test findings, it has been shown that AES encryption causes files to grow by about 25% of their initial dimensions. However, the encrypted file code shrunk by around 30% after Huffman compression. The analysis showed the use of arithmetic coding with the AES (Advanced Encryption Standard) technique to secure and condense content. The basic experiment includes first encoding the material arithmetically, then encrypting it via the AES method, and then transmitting the data. The information is encrypted and interpreted at the other end to create the user data. The ability of the paper to simultaneously encode, decode, and compress data is a benefit. The source file size is 128 bits or 256 bits in AES, therefore the goal is to use digital arithmetic coding to compress the data before encryption [10].

Nowadays, since everything is accomplished via information technology, there is a much greater demand for cryptography [11]. By integrating Huffman coding to shorten the content, Kumari et al. have tried to enhance the privacy of internet data [12]. The practiced approach is an attempt to compact, protect, and conceal the data [13]. It outlines the process by employing different encryption algorithms one at a time, and the goal is to achieve the level of protection possible out of the solutions that are already in place [14][15]. The suggested strategy is applied in MATLAB2016a, and the results obtained in this research demonstrate that this approach is superior to the previous methods [16]. Joshi and Sharma have used the LZW technique to do data immersion [17]. Then, using the AES method, resilience is achieved. Lastly, computerized data is embedded in an encrypted picture using a spatial approach [18]. Research is conducted using actual dataset images. Results for quality factors demonstrate that the suggested approach preserved SNR and PSNR values with excellent data resilience [19]. Encrypting the digital data may be used to overcome the ownership issue. This method starts by clustering human readable texts and encrypting them using AES and Elliptic Curve Encryption (ECC). Next, it utilizes compression to generate cipher blocks, and eventually, it attaches the MAC address and the AES key encrypted by ECC to generate all of the encrypted messages. The findings of the method's explanation and application demonstrate that itmay decrease encryption time, decryption time, and overall operating computational burden without sacrificing safety.

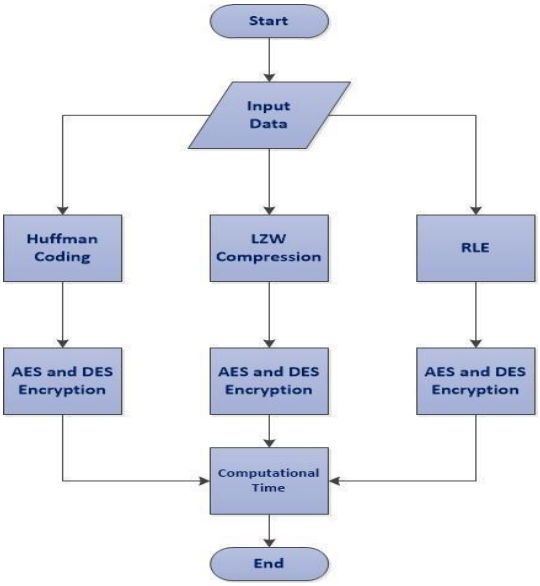
III. METHODOLOGY

Compression and encryption of data are two different procedures but somehow they are interrelated. Both can be applied to achieve randomness in the input data. In today's modern communication systems, the computational performance and security of data are the most essential requirements. In this article, the compression-encryption algorithms are employed in combination to analyze the performance in terms of time. The procedure followed in the study is that first the input data is compressed using various algorithms and then the AES and DES encryptiontechniques are applied to it as shown in figure 3.



[Fig.3: Methodology]

The Main Structure of the Performed Study is Given in Figure 4.



[Fig.4: Architecture of the Method]

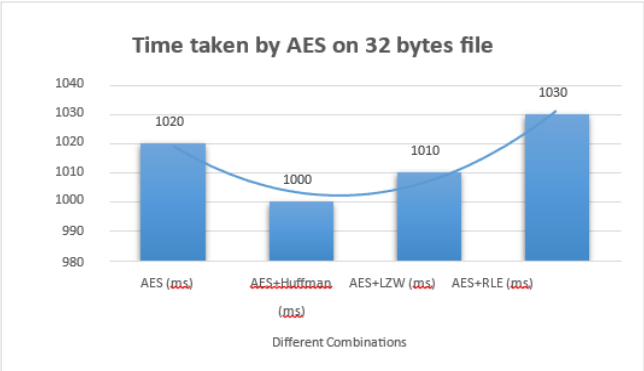
After the compression of input data using each compression technique, the encryption algorithms were applied to the resultant compressed data separately. All the codes of the used algorithms were implemented in Visual Studio using the Go language.

IV. RESULTS AND DISCUSSION

In the proposed article, the performance of different compression techniques with AES and DES encryption was analyzed. The resultant data revealed that the integration of compression encryption greatly depends on the type of data and requirements of a user. The study shows that an effective combination of compression and encryption can achieve the goal of efficient and secure processing and transmission of data in today's modern technological approaches like the Internet of Everything (IoE). With characteristics like high bandwidth and low latency of 6G, these procedures should be considered in the first place. The overall results are discussed below.

A. AES and Compression

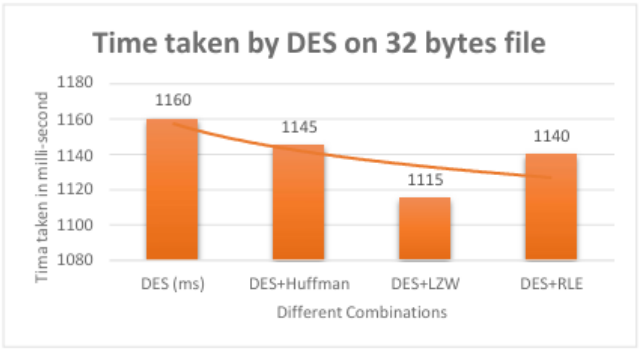
The data extracted from the combined implementation of various compression algorithms with AES is shown in Figure 5. It can be realized that the encryption time can be greatly reduced with these combinations. Overall, the time taken for encryption can be represented in chronological order as AES+RLE > AES > AES+LZW > AES+Huffman. It can be concluded that the AES- Huffman combination is very efficient in terms of computational time.



[Fig.5: AES and Compression]

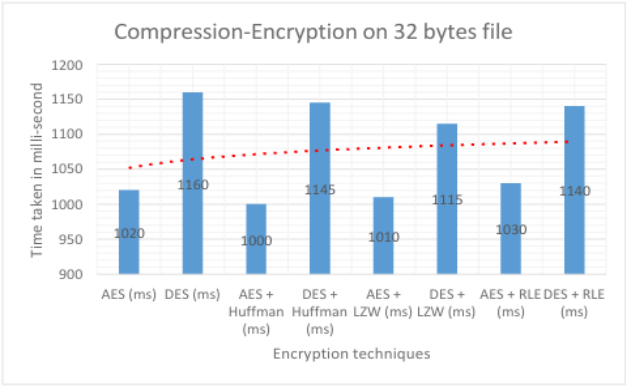
B. DES and Compression

The effective integration of DES with different compression algorithms achieved some interesting results as shown in figure 6. The time complexity was minimized efficiently. The obtained data can be represented in chronological order as DES > DES+RLE > DES+Huffman > DES+LZW. So, the most efficient one is the DES+LZW combination.



[Fig.6: DES and Compression]

The whole data obtained from the study is shown in Figure 7.



[Fig.7: AES, DES, and Encryption]

V. CONCLUSION

Because condensed data is much more dependable and manageable, data compression is a crucial aspect of information safety. Data that has been compressed well is reliable, safe, and simple to communicate. Cryptography is the basis for any secure and reliable communication technique that allows the end users to communicate securely and confidentially. The techniques of compression were integrated with Huffman Coding, LZW, and RLE encryption, and their performance was studied. The study shows that the encryption time can be greatly minimized with the employment of compression and encryption in integration.

DECLARATION STATEMENT

After aggregating input from all authors, I must verify the accuracy of the following information as the article's author.

- **Conflicts of Interest/ Competing Interests:** Based on my understanding, this article has no conflicts of interest.

Retrieval Number:100.1/ijcns.A142704010524
DOI: 10.54105/ijcns.A1427.04021124
Journal Website: www.ijcns.latticescipub.com

Published By:
Lattice Science Publication (LSP)

- **Funding Support:** This article has not been sponsored or funded by any organization or agency. The independence of this research is a crucial factor in affirming its impartiality, as it has been conducted without any external sway.
- **Ethical Approval and Consent to Participate:** The data provided in this article is exempt from the requirement for ethical approval or participant consent.
- **Data Access Statement and Material Availability:** The adequate resources of this article are publicly accessible.
- **Authors Contributions:** The authorship of this article is contributed solely.

REFERENCES

1. J. D. A. Correa, A. S. R. Pinto, and C. Montez, "Lossy Data Compression for IoT Sensors: A Review," Internet of Things, vol. 19, p. 100516, 2022. https://doi.org/10.1016/j.iot.2022.100516
2. R. Bhanot and R. Hans, "A review and comparative analysis of various encryption algorithms," International Journal of Security Its Applications, vol. 9, no. 4, pp. 289-306, 2015. https://www.earticle.net/Article/A245530
3. A. Moffat, "Huffman coding," ACM Computing Surveys, vol. 52, no. 4, pp. 1-35, 2019. https://doi.org/10.1145/3342555
4. H. Dheemanth, "LZW data compression," American Journal of Engineering Research, vol. 3, no. 2, pp. 22-26, 2014. http://www.ajer.org/
5. B. Strasser, A. Botea, and D. Harabor, "Compressing optimal paths with run length encoding," Journal of Artificial Intelligence Research, vol. 54, pp. 593-629, 2015. https://doi.org/10.1613/jair.4931
6. K.-L. Tsai, Y.-L. Huang, F.-Y. Leu, I. You, Y.-L. Huang, and C.-H. Tsai, "AES-128 based secure low power communication for LoRaWAN IoT environments," Ieee Access, vol. 6, pp. 45325-45334, 2018. DOI: https://doi.org/10.1109/ACCESS.2018.2852563
7. K. Logunleko, O. Adeniji, and A. Logunleko, "A comparative study of symmetric cryptography mechanism on DES AES and EB64 for information security," Int. J. Sci. Res. in Computer Science Engineering, vol. 8, no. 1, 2020. https://www.isroset.org/journal/IJSRCSE/full_paper_view.php?paper_id=1690
8. B. Carpentieri, "Efficient compression and encryption for digital data transmission," Security Communication Networks, vol. 2018, 2018. https://doi.org/10.1155/2018/9591768
9. X.-J. Tong, P. Chen, and M. Zhang, "A joint image lossless compression and encryption method based on chaotic map," Multimedia Tools Applications, vol. 76, no. 12, pp. 13995-14020, 2017. https://doi.org/10.1007/s11042-016-3775-6
10. A. Anand, A. K. Singh, Z. Lv, and G. Bhatnagar, "Compression-then-encryption-based secure watermarking technique for smart healthcare system," IEEE MultiMedia, vol. 27, no. 4, pp. 133- 143, 2020. DOI: https://doi.org/10.1109/MMUL.2020.2993269
11. M. E. Hameed, M. M. Ibrahim, N. Abd Manap, and A. A. Mohammed, "A lossless compression and encryption mechanism for remote monitoring of ECG data using Huffman coding and CBC-AES," Future generation computer systems, vol. 111, pp. 829-840, 2020. https://doi.org/10.1016/j.future.2019.10.010
12. M. R. Ashila, N. Atikah, 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), 2019, pp. 1-5: IEEE. DOI: https://doi.org/10.1109/ICIC47613.2019.8985899
13. P. S. Mukesh, M. S. Pandya, and S. Pathak, "Enhancing AES algorithm with arithmetic coding," in 2013 International Conference on Green Computing, Communication and Conservation of Energy (ICGCE), 2013, pp. 83-86: IEEE. DOI: https://doi.org/10.1109/ICGCE.2013.6823404
14. M. Kumari, V. Pawar, P. J. I. J. o. N. S. Kumar, and I. A. Vol, "A novel image encryption scheme with Huffman encoding and steganography technique," International Journal of Network Security Its Applications, vol. 11, 2019 2019. https://ssrn.com/abstract=3847524
15. A. K. Joshi and S. Sharma, "Reversible data hiding by utilizing AES encryption and LZW compression," in Proceedings of International Conference on Recent Advancement on Computer and Communication, 2018, pp. 73-81: Springer. https://doi.org/10.1007/978-981-10-8198-9_8
16. T. Yue, C. Wang, and Z.-x. Zhu, "Hybrid encryption algorithm based on wireless sensor networks," in 2019 IEEE international conference on mechatronics and automation (ICMA), 2019, pp. 690- 694: IEEE. DOI: https://doi.org/10.1109/ICMA.2019.8816451
17. Sisodia, Mr. A., Mrs. Swati, & Hashmi, Mrs. H. (2020). Incorporation of Non-Fictional Applications in Wireless Sensor Networks. In International Journal of Innovative Technology and Exploring Engineering (Vol. 9, Issue 11, pp. 42–49). https://doi.org/10.35940/ijitee.k7673.0991120
18. Patil, Mrs. Suvarna. S., & Vidyavathi, Dr. B. M. (2022). Application o f Advanced Machine Learning and Artificial Neural Network Methods in Wireless Sensor Networks Based Applications. In International Journal of Engineering and Advanced Technology (Vol. 11, Issue 3, pp. 103–109). https://doi.org/10.35940/ijeat.c3394.0211322
19. Sharma, P. (2023). Zigbee based Wireless Sensor Network for Smart Energy Meter. In International Journal of Recent Technology and Engineering (IJRTE) (Vol. 12, Issue 3, pp. 20–27). https://doi.org/10.35940/ijrte.c7861.0912323
20. Chitransh, A., & Kalyan, B. S. (2021). ARM Microcontroller Based Wireless Industrial Automation System. In Indian Journal of Microprocessors and Microcontroller (Vol. 1, Issue 2, pp. 8–11). https://doi.org/10.54105/ijmm.b1705.091221
21. Pramod, K., Mrs. Durga, M., Apurba, S., & Shashank, S. (2023). An Efficient LEACH Clustering Protocol to Enhance the QoS of WSN. In Indian Journal of Artificial Intelligence and Neural Networking (Vol. 3, Issue 3, pp. 1–8). https://doi.org/10.54105/ijainn.a3822.043323

AUTHOR PROFILE

Sangeen Khan, I have completed a BS in Computer Science from the University of Swabi, Pakistan. Currently, I am doing an MS in Information and Communication Engineering at the University of Science and Technology Beijing, China. My research interests are 1) Privacy-preserving Computing. 2) Homomorphic Encryption. 3) Differential Privacy. 4) Cryptography.