International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 06
Article Received:25 April 2023 Revised:12 May 2023 Accepted:30 June 2023

Java-Based Hardware-Oriented Secure Messaging System Using Symmetric Encryption Techniques

[1]Alka Sawlikar, [2]Bireshwar Ganguly, [3]Devashri Kodgire

[1]Dept. of Electronics Engg, R.C.E.R.T, Chandrapur-India

alkaprasad.sawlikar@gmail.com

[2]Dept. of Data Science, R.C.E.R.T,

Chandrapur-India, bireshwar.ganguly@gmail.com

[3]Dept. of Data Science, R.C.E.R.T,

Chandrapur-India

devashriraich@gmail.com

ABSTRACT: This paper presents a hardware-oriented implementation of a secure message service leveraging symmetric cryptographic techniques, with development and simulation performed using Java. Symmetric encryption algorithms, particularly the Advanced Encryption Standard (AES), were employed for securing message transmission, optimizing both performance and resource usage. The primary goal is to enhance the energy efficiency of secure data transmission by selecting optimal cryptographic and compression algorithm pairs suitable for hardware implementation. Experimental results reveal that combining AES encryption with lightweight compression algorithms significantly reduces computational overhead and improves transmission efficiency.

Keyword*s*- AES, Symmetric Encryption, Compression, Secure Messaging, Java, Hardware Implementation, Energy Optimization

1 INTRODUCTION

The increasing reliance on mobile and embedded systems for secure communications necessitates efficient encryption techniques that maintain low energy consumption. Symmetric key algorithms are preferred for their computational efficiency and suitability for real-time hardware deployment. This paper investigates the implementation of a secure message transmission system using AES, integrated within a Java-based environment, and evaluates its performance for potential hardware translation. By incorporating compression techniques alongside encryption, we aim to reduce the energy cost per message while maintaining robust security.Java is basically developed by Sun Microsystems and now merges into ORACLE Corporation and it is platform independent whose codes are famous as WORA that is writing once run anywhere. Now a day's three billion devices are running java as it is adopted as platform independent programming language. It is used to create real time web pages, in standalone application, enterprise application in mobile, etc [1].Because of its precious facts like object oriented, platform independent, simple, secure, portable, robust, architectural neural, multi threaded, interpreted, dynamic, high performance, distributed etc it is highly used for developing software. Here we are using Eclipse editor [2] [3].For real implementation we have created application file in which UID(user interface design) coding is saved and this will create a window as follows.This window basically divided into LHS and RHS .Left hand side shows the different blocks which are utilized as select a file, read that selected file, convert that file into ASCII ,then compress it and finally encrypt the compressed file[4][5][6]In the right hand side select the encrypted file, read that encrypted file, then decrypt it, decompressed this and finally convert that file into ASCII to get the original one. The data which is to be operated should be in notepad which can have any character, numbers or symbols of any size [7][8].

2. Background and Related Work
Symmetric cryptography, particularly AES, has become the cornerstone for secure data exchange due to its balance of security and performance. Prior work has focused on its implementation in software and hardware, especially in wireless communications utilizing OFDM. Research also indicates that message encryption and compression can significantly affect energy consumption, particularly in power-constrained systems like smartphones and IoT devices.

System Architecture and Proposed Method
The proposed system architecture comprises the following core components:

Message Input Module: Captures user message as input.

Compression Engine: Implements lightweight compression (e.g., Run-Length Encoding - RLE) to reduce message size.

Encryption Module: Uses AES for secure encoding of the compressed message.

Transmission Interface: Simulates message sending over a network.

Receiver Module: Receives, decrypts, and decompresses the message to retrieve the original content.

All modules were developed using Java, chosen for its platform independence and suitability for simulation of embedded systems. The system's modular design also facilitates translation to hardware through HDL or FPGA-based implementations.

3. Algorithms Used

A. AES (Advanced Encryption Standard)
AES is a symmetric block cipher that processes 128-bit blocks with key sizes of 128, 192, or 256 bits. Its round-based structure and substitution-permutation network make it both secure and efficient for hardware realization.

B. RLE (Run-Length Encoding)
RLE is a simple yet effective compression algorithm that minimizes the size of repetitive data, thereby reducing the amount of information to be encrypted and transmitted. It proves particularly efficient in energy-limited environments.

4 PROPOSED SYMMETRIC CRYPTOGRAPHIC ALGORITHM

4.1 Steps of Encipher Algorithm

Input: Read from a text file containing any data.

Operation: Encrypt every symbol using a stream cipher method.

Key: Same key used for both encryption and decryption.

Method:

- Take modulus of key character's ASCII value with key length.
- XOR this remainder with each character of the message.

4.2 Steps of Decryption algorithm

Only single secret key have been exercised in the encryption. It is a symmetric algorithm, so same key is used for encryption and decryption. Decryption method of the algorithm is the reverse of their encryption method.

i) Take input as encrypted message.

ii) Key is same

iii) Perform XOR operation of message with key but not directly. First calculate length of that key used which is constant throughout and take modulus of key length and each key character. Note the remainder.

Then XOR remainder with every character of message. Thus message will get decrypted.

5 Implementation of Proposed Encryption Algorithm In Java

The above are the essential steps for reading file, compressing and encrypting. Similarly we had written for reading encrypted file, Original decrypting, decompressing and conversion into ACSCII and finally getting the original file. In encoding and decoding process we had taken single key and single XOR operation. The reason behind it is if used multiple keys and more operations then time and moreover size of encryption was increasing that means characters are increasing in encryption process. Actually read file is a method to go to declaration where the logic is written that is read the file and convert every message character to corresponding ASCII number. When it enters into declaration for read file it will check all characters in every line and also checks whether that file is present in that path or not. If not then it will give error as "please select the text file"

Then in while loop it will read and after checking add in file length. Thus file reading process is going on and once it will finish size is observed and then return the ASCII values. The ASCII data have to send to compression algorithm along with file path and ASCII data and this method will return compressed data. Then compression algorithm will catch ASCII along with data file and logically compressed it and add into compressed file. This compressed data is saved which is again return type and send for encryption. We have created encryption ,decryption and decompression in another file that's why take its object means only method can call and is already available which then applied for encryption. Then in encryption process first set key, give null, means void means nothing no compressed message neither key is present so no performance will occur and confirm that both key and message should be available.

Once it is confirm then split it into character then see length, it will appear in integer and then XOR it according to algorithm. Save as base 64 encode and it is that file where actual encoding takes place and it takes bytes and perform encryption on it. This encrypted data is saved in new message and that is returned back for further operation. Base 64 encoder X-ored data is saved, this is internal method of java. Then write it on final encrypted data. Now for decryption read the encrypted file and reverse process have to apply to get decrypted data and then write decrypted data and then send for decompression we will get decompressed ASCII data then convert ASCII integer to original data and now it will not return as it is final file. Decryption is done on direct encrypted file only with same key which is used in encryption then send to base 64 decoder then XOR so that we will get exactly decoded data which we had used for encrypted. In decompression first decrypted data is converted as below:

File Reading

- Read a file line by line.
- Validate the file path.
- Convert each character to ASCII.
- Store ASCII values.

Compression (KSA or similar)

- Apply a custom compression algorithm to ASCII values.
- Return compressed data.

Encryption

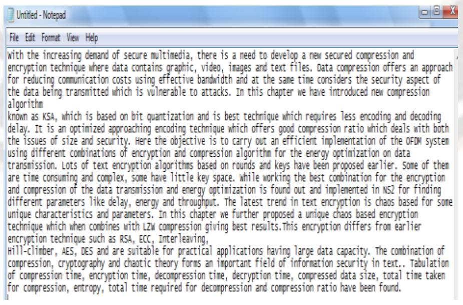
- Verify that the compressed data and encryption key exist.
- Generate the encryption key (single key used).
- XOR every character using your custom method.
- Base64 encode the result.
- Save encrypted file.

Decryption

- Read encrypted file.
- Base64 decode.
- Use the same XOR key to decrypt.
- Return compressed ASCII data.

Decompression

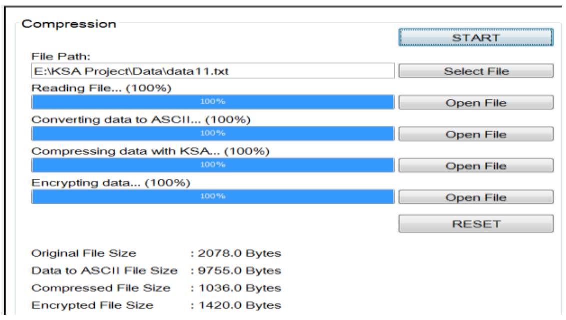
- Split string into character array.
- Convert characters to ASCII integers.
- Apply decompression algorithm (inverse of KSA).
- Convert decompressed ASCII back to characters.
- Save to output file.



```
with the increasing demand of secure multimedia, there is a need to develop a new secured compression and
encryption technique where data contains graphic, video, images and text files. Data compression offers an approach
for reducing communication costs using effective bandwidth and at the same time considers the security aspect of
the data being transmitted which is vulnerable to attacks. In this chapter we have introduced new compression
algorithm
known as KSA, which is based on bit quantization and is best technique which requires less encoding and decoding
delay. It is an optimized approaching encoding technique which offers good compression ratio which deals with both
the issues of size and security. Here the objective is to carry out an efficient implementation of the OFDM system
using different combinations of encryption and compression algorithm for the energy optimization on data
transmission. Lots of text encryption algorithms based on rounds and keys have been proposed earlier. Some of them
are time consuming and complex, some have little key space. While working the best combination for the encryption
and compression of the data transmission and energy optimization is found out and implemented in NS2 for finding
different parameters like delay, energy and throughput. The latest trend in text encryption is chaos based for some
unique characteristics and parameters. In this chapter we further proposed a unique chaos based encryption
technique which when combines with LZW compression giving best results.This encryption differs from earlier
encryption technique such as RSA, ECC, Interleaving,
Hill-climber, AES, DES and are suitable for practical applications having large data capacity. The combination of
compression, cryptography and chaotic theory forms an important field of information security in text.. Tabulation
of compression time, encryption time, decompression time, decryption time, compressed data size, total time taken
for compression, entropy, total time required for decompression and compression ratio have been found.
```

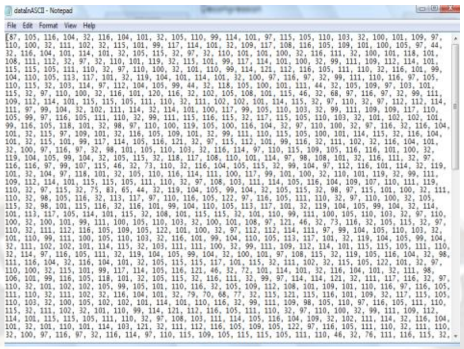
Fig. 1. Text File fetched from Notepad

This is the text file written in notepad .With the help of programming we can see its size, ASCII values, compressed file, encrypted data and decrypted data as follows:



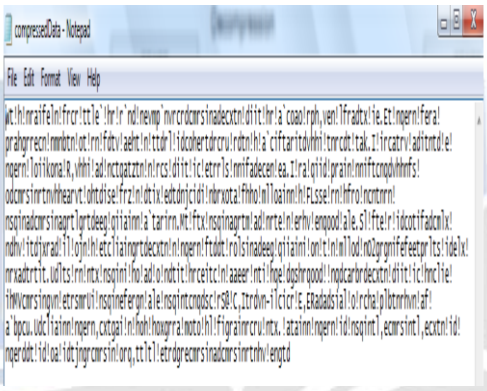
```
Compression                                                    START
File Path:
E:\KSA Project\Data\data11.txt                                 Select File
Reading File... (100%)
100%                                                           Open File
Converting data to ASCII... (100%)
100%                                                           Open File
Compressing data with KSA... (100%)
100%                                                           Open File
Encrypting data... (100%)
100%                                                           Open File
                                                               RESET
Original File Size         : 2078.0 Bytes
Data to ASCII File Size    : 9755.0 Bytes
Compressed File Size       : 1036.0 Bytes
Encrypted File Size        : 1420.0 Bytes
```

Fig..2. Fetched file with size, ASCII, Compressed and Encrypted values



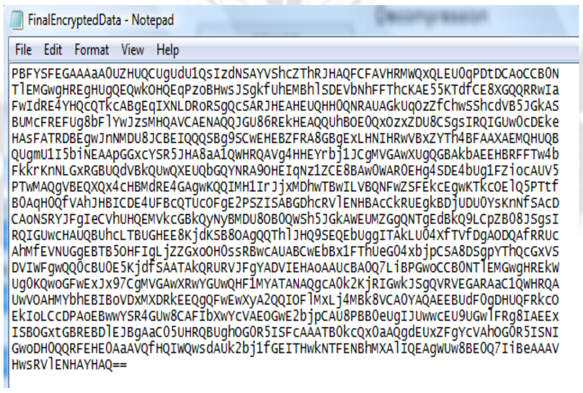
```
[87, 105, 116, 104, 32, 116, 104, 101, 32, 105, 110, 99, 114, 101, 97, 115, 105, 110, 103, 32, 100, 101, 109, 97,
110, 100, 32, 111, 102, 32, 115, 101, 99, 117, 114, 101, 32, 109, 117, 108, 116, 105, 109, 101, 100, 105, 97, 44,
32, 116, 104, 101, 114, 101, 32, 105, 115, 32, 97, 32, 110, 101, 101, 100, 32, 116, 111, 32, 100, 101, 118, 101,
108, 111, 112, 32, 97, 32, 110, 101, 119, 32, 115, 101, 99, 117, 114, 101, 100, 32, 99, 111, 109, 112, 114, 101,
115, 115, 105, 111, 110, 32, 97, 110, 100, 32, 101, 110, 99, 114, 121, 112, 116, 105, 111, 110, 32, 116, 101, 99,
104, 110, 105, 113, 117, 101, 32, 119, 104, 101, 114, 101, 32, 100, 97, 116, 97, 32, 99, 111, 110, 116, 97, 105,
110, 115, 32, 103, 114, 97, 112, 104, 105, 99, 44, 32, 118, 105, 100, 101, 111, 44, 32, 105, 109, 97, 103, 101,
115, 32, 97, 110, 100, 32, 116, 101, 120, 116, 32, 102, 105, 108, 101, 115, 46, 32, 68, 97, 116, 97, 32, 99, 111,
109, 112, 114, 101, 115, 115, 105, 111, 110, 32, 111, 102, 102, 101, 114, 115, 32, 97, 110, 32, 97, 112, 112, 114,
111, 97, 99, 104, 32, 102, 111, 114, 32, 114, 101, 100, 117, 99, 105, 110, 103, 32, 99, 111, 109, 109, 117, 110,
105, 99, 97, 116, 105, 111, 110, 32, 99, 111, 115, 116, 115, 32, 117, 115, 105, 110, 103, 32, 101, 102, 102, 101,
99, 116, 105, 118, 101, 32, 98, 97, 110, 100, 119, 105, 100, 116, 104, 32, 97, 110, 100, 32, 97, 116, 32, 116, 104,
101, 32, 115, 97, 109, 101, 32, 116, 105, 109, 101, 32, 99, 111, 110, 115, 105, 100, 101, 114, 115, 32, 116, 104,
101, 32, 115, 101, 99, 117, 114, 105, 116, 121, 32, 97, 115, 112, 101, 99, 116, 32, 111, 102, 32, 116, 104, 101,
32, 100, 97, 116, 97, 32, 98, 101, 105, 110, 103, 32, 116, 114, 97, 110, 115, 109, 105, 116, 116, 101, 100, 32,
119, 104, 105, 99, 104, 32, 105, 115, 32, 118, 117, 108, 110, 101, 114, 97, 98, 108, 101, 32, 116, 111, 32, 97,
116, 116, 97, 99, 107, 115, 46, 32, 73, 110, 32, 116, 104, 105, 115, 32, 99, 104, 97, 112, 116, 101, 114, 32, 119,
101, 32, 104, 97, 118, 101, 32, 105, 110, 116, 114, 111, 100, 117, 99, 101, 100, 32, 110, 101, 119, 32, 99, 111,
109, 112, 114, 101, 115, 115, 105, 111, 110, 32, 97, 108, 103, 111, 114, 105, 116, 104, 109, 107, 110, 111, 119,
110, 32, 97, 115, 32, 75, 83, 65, 44, 32, 119, 104, 105, 99, 104, 32, 105, 115, 32, 98, 97, 115, 101, 100, 32, 111,
110, 32, 98, 105, 116, 32, 113, 117, 97, 110, 116, 105, 122, 97, 116, 105, 111, 110, 32, 97, 110, 100, 32, 105,
115, 32, 98, 101, 115, 116, 32, 116, 101, 99, 104, 110, 105, 113, 117, 101, 32, 119, 104, 105, 99, 104, 32, 114,
101, 113, 117, 105, 114, 101, 115, 32, 108, 101, 115, 115, 32, 101, 110, 99, 111, 100, 105, 110, 103, 32, 97, 110,
100, 32, 100, 101, 99, 111, 100, 105, 110, 103, 32, 100, 101, 108, 97, 121, 46, 32, 73, 116, 32, 105, 115, 32, 97,
110, 32, 111, 112, 116, 105, 109, 105, 122, 101, 100, 32, 97, 112, 112, 114, 111, 97, 99, 104, 105, 110, 103, 32,
101, 110, 99, 111, 100, 105, 110, 103, 32, 116, 101, 99, 104, 110, 105, 113, 117, 101, 32, 119, 104, 105, 99, 104,
32, 111, 102, 102, 101, 114, 115, 32, 103, 111, 111, 100, 32, 99, 111, 109, 112, 114, 101, 115, 115, 105, 111, 110,
32, 114, 97, 116, 105, 111, 32, 119, 104, 105, 99, 104, 32, 100, 101, 97, 108, 115, 32, 119, 105, 116, 104, 32, 98,
111, 116, 104, 32, 116, 104, 101, 32, 105, 115, 115, 117, 101, 115, 32, 111, 102, 32, 115, 105, 122, 101, 32, 97,
110, 100, 32, 115, 101, 99, 117, 114, 105, 116, 121, 46, 32, 72, 101, 114, 101, 32, 116, 104, 101, 32, 111, 98,
106, 101, 99, 116, 105, 118, 101, 32, 105, 115, 32, 116, 111, 32, 99, 97, 114, 114, 121, 32, 111, 117, 116, 32, 97,
110, 32, 101, 102, 102, 105, 99, 105, 101, 110, 116, 32, 105, 109, 112, 108, 101, 109, 101, 110, 116, 97, 116, 105,
111, 110, 32, 111, 102, 32, 116, 104, 101, 32, 79, 70, 68, 77, 32, 115, 121, 115, 116, 101, 109, 32, 117, 115, 105,
110, 103, 32, 100, 105, 102, 102, 101, 114, 101, 110, 116, 32, 99, 111, 109, 98, 105, 110, 97, 116, 105, 111, 110,
115, 32, 111, 102, 32, 101, 110, 99, 114, 121, 112, 116, 105, 111, 110, 32, 97, 110, 100, 32, 99, 111, 109, 112,
114, 101, 115, 115, 105, 111, 110, 32, 97, 108, 103, 111, 114, 105, 116, 104, 109, 32, 102, 111, 114, 32, 116, 104,
101, 32, 101, 110, 101, 114, 103, 121, 32, 111, 112, 116, 105, 109, 105, 122, 97, 116, 105, 111, 110, 32, 111, 110,
32, 100, 97, 116, 97, 32, 116, 114, 97, 110, 115, 109, 105, 115, 115, 105, 111, 110, 46, 32, 76, 111, 116, 115, 32,
```

Fig. 3. ASCII values of original file



```
wt!h!nraifeln!frcr!ttle`!hr!r`nd!newmp`nvrcrdcmrsinadecxtn!diit!hr!a`coao!rph,ven!lfradtx!ie.Et!nqern!fera!
prahgrrecn!nmnbtn!ot!rn!fdtv!aeht!n!ttdrl!idcohertdrcru!rdtn!h!a`ciftaritdvhhi!tnrcdt!tak.I!ircatrv!aditntd!e!
nqern!loiikona!R,vhhi!ad!nctqatztn!n!rcs!diit!ic!etrrls!nnifadecen!ea.I!ra!qiid!prain!nniftcnqdvhhnfs!
odcmrsinrtnvhhearvt!ohtdise!frz!n!dtix!edtdnjcidi!nbrxota!fhho!mlloainn!h!FLsse!rn!hfro!ncntnrn!
nsqinadcmrsinagrtlgrtdeeg!qiiainn!a`tarirn.Mt!ftx!nsqinagrtm!ad!nrte!n!erhv!enqood!ale.Sl!fte!r!idcotifadcmlx!
ndhv!itdjxrad!il!ojn!h!etcliaingrtdecxtn!n!nqern!ftddt!rolsinadeeg!qiiaini!on!t!n!mllod!n02grgnifefeetprlts!idelx!
nrxadtrtit.Udlts!rn!ntx!nsqini!ho!ad!o!ndtit!hrceitc!n!aaeer!nti!hqe!dgshrqood!!nqdcarbrdecxtn!diit!ic!hnclie!
ihMVcmrsingvn!etrsmrUi!nsqinefergn!ale!nsqintcnqdsc!rS@!C,Itrdvn-ilcicr!E,ERadadsial!o!rcha!plbtnrhvn!af!
a`bpcu.Udcliainn!nqern,cxtgai!n!hoh!hoxgrra!moto!hl!figrainrcru!ntx.!atainn!nqern!id!nsqintl,ecmrsintl,ecxtn!id!
nqerddt!id!oa!idtjngrcmrsin!orq,ttltl!etrdgrecmrsinadcmrsinrtnhv!engtd
```

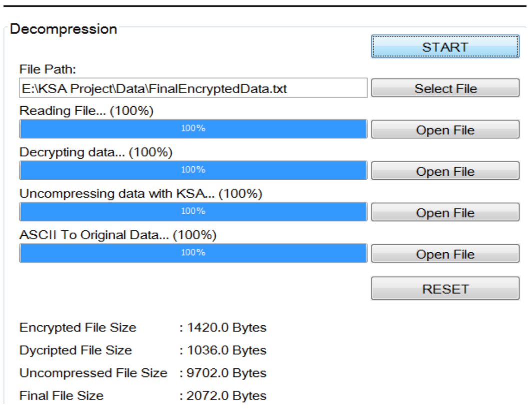
Fig. 4. Compressed data of original file



```
PBFYSFEGAAAaA0UZHUQCUgUdU1QsIzdNSAYVShcZThRJHAQFCFAVHRMWQxQLEU0gPDtDCAoCCB0N
TlEMGwgHREgHUgQEQwkOHQEqPzoBHwsJSgkfUhEMBhlSDEVbNhFFThcKAE55KTdfCE8XGQQRRwIa
FwIdRE4YHQcQTkcABgEqIXNLDRoRSgQcSARJHEAHEUQHH0QNRAUAGkUqOzZfChwSShcdVB5JGkAS
BUMcFREFUg8bFlYwJzsMHQAVCAENAQQJGU86REkHEAQQUhBOE0QxOzxZDU8CSgsIRQIGUw0cDEke
HAsFATRDBEgwJnNMDU8JCBEIQQQSBg9SCwEHEBZFRA8GBgExLHNIHRwVBxZYTh4BFAAXAEMQHUQB
QUgmU1I5biNEAApGGxcYSR5JHA8aA1QWHRQAVg4HHEYrbj1JCgMVGAwXUgQGBAkbAEEHBRFFTw4b
FkkrKnNLGxRGBUQdVBkQUwQXEUQbGQYNRA9OHEIqNz1ZCE8BAw0WAR0EHg4SDE4bUg1FZiocAUV5
PTwMAQgVBEQXQx4cHBMdRE4GAgwKQQIMH1IrJjxMDhwTBwILVBQNFwZSFEkcEgwKTkcOElQ5PTtf
B0AqH0QfVAhJHBICDE4UFBcQTUcOFgE2PSZISABGDhcRVlENHBAcCkRUEgkBDjUDU0YsKnNfSAcD
CAoNSRYJFgIeCVhUHQEMVkcGBkQyNyBMDU8OB0QWSh5JGkAWEUMZGgQNTgEdBkQ9LCpZB08JSgsI
RQIGUwcHAUQBUhcLTBUGHEE8KjdKSB8OAgQQThlJHQ9SEQEbUggITAkLU04XfTVfDgAODQAfRRUc
AhMfEVNUGgEBTB5OHFIgLjZZGxoOH0ssRBwCAUABCwEbBx1FThUeG04xbjpCSA8DSgpYThQcGxVS
DVIWFgwQQ0cBU0E5KjdfSAATAkQRURVJFgYADVIEHAoAAUcBA0Q7LiBPGwoCCB0NTlEMGwgHREkW
Ug0KQwoGFwExJx97CgMVGAwXRwYGUwQHF1MYATANAQgcA0k2KjRIGwkJSgQVRVEGARAaC1QWHRQA
UwVOAHMYbhEBIBoVDxMXDRkEEQgQFwEwXyA2QQIOFlMxLj4MBk8VCA0YAQAEEBUdF0gDHUQFRkcO
EkIoLCcDPAoEBwwYSR4GUw8CAFIbXwYcVAEOGwE2bjpCAU8PBB0eUgIJUwwcEU9UGwlFRg8IAEEx
ISBOGxtGBREBDlEJBgAaC05UHRQBUghOG0R5ISFcAAATB0kcQx0aAQgdEUxZFgYcVAhOG0R5ISNI
GwoDH0QQRFEHE0AaAVQfHQIWQwsdAUk2bj1fGEITHwkNTFENBhMXAlIQEAgWUw8BEOQ7IiBeAAAV
HwsRVlENHAYHAQ==
```

Fig. 5. Encrypted file after compression using proposed encryption technique

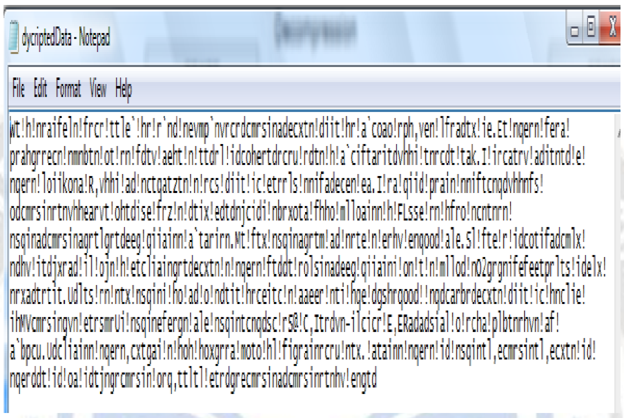
Similarly size of every decrypted file, decompressed data and ASCII to original file in reverse manner is observed as follows. Figure 6,figure 7,figure 8 and figure 9 shows the decrypted file, decompressed data and ASCII to original.



File Path:
E:\KSA Project\Data\FinalEncryptedData.txt
Reading File... (100%)
Decrypting data... (100%)
Uncompressing data with KSA... (100%)
ASCII To Original Data... (100%)

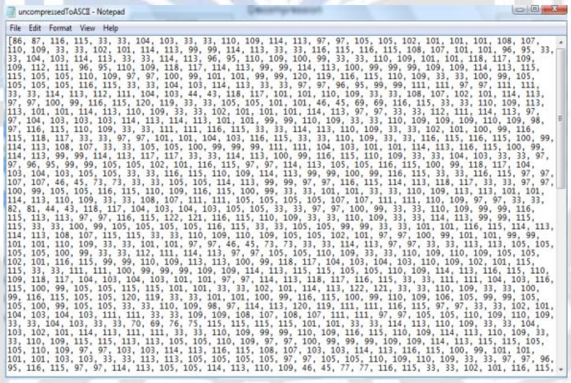
Encrypted File Size	: 1420.0 Bytes
Dycripted File Size	: 1036.0 Bytes
Uncompressed File Size	: 9702.0 Bytes
Final File Size	: 2072.0 Bytes

Fig. 6. Values of decrypted file, decompressed data and ASCII to original



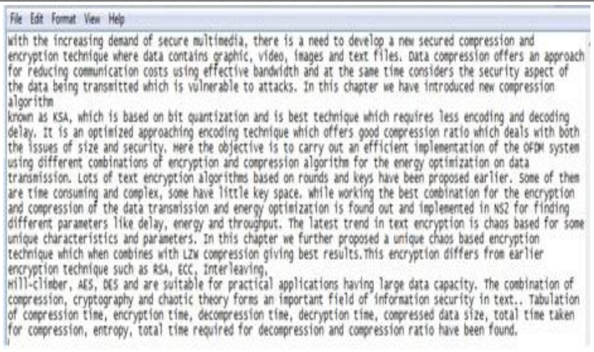
```
wt!h!nraifeln!frcr!ttle`!hr!r`nd!nevmp`nvrcrdcmrsinadecxtn!diit!hr!a`coao!rph,ven!lfradtx!ie.Et!ngern!fera!
prahgrrecn!mmbtn!ot!rn!fdtv!aeht!n!ttdrl!idcohertdrcru!rdtn!h!a`ciftaritdvhhi!tnrcdt!tak.I!ircatrv!aditntd!e!
ngern!loiikona!R,vhhi!ad!nctgatztn!n!rcs!diit!ic!etrrls!nnifadecen!ea.I!ra!qiid!prain!nniftcnqdvhhnfs!
odcmrsinrtnvhhearvt!ohtdise!frz!n!dtix!edtdnjcidi!nbrxota!fhho!mlloainn!h!FLsse!rn!hfro!ncntnrn!
nsqinadcmrsinagrtlgrtdeeg!qiiainn!a`tarirn.Mt!ftx!nsqinagrtm!ad!nrte!n!erhv!enqood!ale.Sl!fte!r!idcotifadcmlx!
ndhv!itdjxrad!il!ojn!h!etcliaingrtdecxtn!n!ngern!ftddt!rolsinadeeg!qiiaini!on!t!n!mllod!n02grgnifefeetprlts!idelx!
nrxadtrtit.Udlts!rn!ntx!nsqini!ho!ad!o!ndtit!hrceitc!n!aaeer!nti!hge!dgshrqood!!nqdcarbrdecxtn!diit!ic!hnclie!
ihWVcmrsingvn!etrsmrUi!nsqinefergn!ale!nsqintcnqdsc!rS@!C,Itrdvn-ilcicr!E,ERadadsial!o!rcha!plbtnrhvn!af!
a`bpcu.Udcliainn!ngern,cxtgai!n!hoh!hoxgrra!moto!hl!figrainrcru!ntx.!atainn!ngern!id!nsqintl,ecmrsintl,ecxtn!id!
ngerddt!id!oa!idtjngrcmrsin!orq,ttltl!etrdgrecmrsinadcmrsinrtnhv!engtd
```

Fig. 7. Decrypted data file with new symmetric cryptographic algorithm



```
[86, 87, 116, 115, 33, 33, 104, 103, 33, 33, 110, 109, 114, 113, 97, 97, 105, 105, 102, 101, 101, 101, 108, 107,
110, 109, 33, 33, 102, 101, 114, 113, 99, 99, 114, 113, 33, 33, 116, 115, 116, 115, 108, 107, 101, 101, 96, 95, 33,
33, 104, 103, 114, 113, 33, 33, 114, 113, 96, 95, 110, 109, 100, 99, 33, 33, 110, 109, 101, 101, 118, 117, 109,
109, 112, 111, 96, 95, 110, 109, 118, 117, 114, 113, 99, 99, 114, 113, 100, 99, 99, 99, 109, 109, 114, 113, 115,
115, 105, 105, 110, 109, 97, 97, 100, 99, 101, 101, 99, 99, 120, 119, 116, 115, 110, 109, 33, 33, 100, 99, 105,
105, 105, 105, 116, 115, 33, 33, 104, 103, 114, 113, 33, 33, 97, 97, 96, 95, 99, 99, 111, 111, 97, 97, 111, 111,
33, 33, 114, 113, 112, 111, 104, 103, 44, 43, 118, 117, 101, 101, 110, 109, 33, 33, 108, 107, 102, 101, 114, 113,
97, 97, 100, 99, 116, 115, 120, 119, 33, 33, 105, 105, 101, 101, 46, 45, 69, 69, 116, 115, 33, 33, 110, 109, 113,
113, 101, 101, 114, 113, 110, 109, 33, 33, 102, 101, 101, 101, 114, 113, 97, 97, 33, 33, 112, 111, 114, 113, 97,
97, 104, 103, 103, 103, 114, 113, 114, 113, 101, 101, 99, 99, 110, 109, 33, 33, 110, 109, 109, 109, 110, 109, 98,
97, 116, 115, 110, 109, 33, 33, 111, 111, 116, 115, 33, 33, 114, 113, 110, 109, 33, 33, 102, 101, 100, 99, 116,
115, 118, 117, 33, 33, 97, 97, 101, 101, 104, 103, 116, 115, 33, 33, 110, 109, 33, 33, 116, 115, 116, 115, 100, 99,
114, 113, 108, 107, 33, 33, 105, 105, 100, 99, 99, 99, 111, 111, 104, 103, 101, 101, 114, 113, 116, 115, 100, 99,
114, 113, 99, 99, 114, 113, 117, 117, 33, 33, 114, 113, 100, 99, 116, 115, 110, 109, 33, 33, 104, 103, 33, 33, 97,
97, 96, 95, 99, 99, 105, 105, 102, 101, 116, 115, 97, 97, 114, 113, 105, 105, 116, 115, 100, 99, 118, 117, 104,
103, 104, 103, 105, 105, 33, 33, 116, 115, 110, 109, 114, 113, 99, 99, 100, 99, 116, 115, 33, 33, 116, 115, 97, 97,
107, 107, 46, 45, 73, 73, 33, 33, 105, 105, 114, 113, 99, 99, 97, 97, 116, 115, 114, 113, 118, 117, 33, 33, 97, 97,
100, 99, 105, 105, 116, 115, 110, 109, 116, 115, 100, 99, 33, 33, 101, 101, 33, 33, 110, 109, 113, 113, 101, 101,
114, 113, 110, 109, 33, 33, 108, 107, 111, 111, 105, 105, 105, 105, 107, 107, 111, 111, 110, 109, 97, 97, 33, 33,
82, 81, 44, 43, 118, 117, 104, 103, 104, 103, 105, 105, 33, 33, 97, 97, 100, 99, 33, 33, 110, 109, 99, 99, 116,
115, 113, 113, 97, 97, 116, 115, 122, 121, 116, 115, 110, 109, 33, 33, 110, 109, 33, 33, 114, 113, 99, 99, 115,
115, 33, 33, 100, 99, 105, 105, 105, 105, 116, 115, 33, 33, 105, 105, 99, 99, 33, 33, 101, 101, 116, 115, 114, 113,
114, 113, 108, 107, 115, 115, 33, 33, 110, 109, 110, 109, 105, 105, 102, 101, 97, 97, 100, 99, 101, 101, 99, 99,
101, 101, 110, 109, 33, 33, 101, 101, 97, 97, 46, 45, 73, 73, 33, 33, 114, 113, 97, 97, 33, 33, 113, 113, 105, 105,
105, 105, 100, 99, 33, 33, 112, 111, 114, 113, 97, 97, 105, 105, 110, 109, 33, 33, 110, 109, 110, 109, 105, 105,
102, 101, 116, 115, 99, 99, 110, 109, 113, 113, 100, 99, 118, 117, 104, 103, 104, 103, 110, 109, 102, 101, 115,
115, 33, 33, 111, 111, 100, 99, 99, 99, 109, 109, 114, 113, 115, 115, 105, 105, 110, 109, 114, 113, 116, 115, 110,
109, 118, 117, 104, 103, 104, 103, 101, 101, 97, 97, 114, 113, 118, 117, 116, 115, 33, 33, 111, 111, 104, 103, 116,
115, 100, 99, 105, 105, 115, 115, 101, 101, 33, 33, 102, 101, 114, 113, 122, 121, 33, 33, 110, 109, 33, 33, 100,
99, 116, 115, 105, 105, 120, 119, 33, 33, 101, 101, 100, 99, 116, 115, 100, 99, 110, 109, 106, 105, 99, 99, 105,
105, 100, 99, 105, 105, 33, 33, 110, 109, 98, 97, 114, 113, 120, 119, 111, 111, 116, 115, 97, 97, 33, 33, 102, 101,
104, 103, 104, 103, 111, 111, 33, 33, 109, 109, 108, 107, 108, 107, 111, 111, 97, 97, 105, 105, 110, 109, 110, 109,
33, 33, 104, 103, 33, 33, 70, 69, 76, 75, 115, 115, 115, 115, 101, 101, 33, 33, 114, 113, 110, 109, 33, 33, 104,
103, 102, 101, 114, 113, 111, 111, 33, 33, 110, 109, 99, 99, 110, 109, 116, 115, 110, 109, 114, 113, 110, 109, 33,
33, 110, 109, 115, 115, 113, 113, 105, 105, 110, 109, 97, 97, 100, 99, 99, 99, 109, 109, 114, 113, 115, 115, 105,
105, 110, 109, 97, 97, 103, 103, 114, 113, 116, 115, 108, 107, 103, 103, 114, 113, 116, 115, 100, 99, 101, 101,
101, 101, 103, 103, 33, 33, 113, 113, 105, 105, 105, 105, 97, 97, 105, 105, 110, 109, 110, 109, 33, 33, 97, 97, 96,
95, 116, 115, 97, 97, 114, 113, 105, 105, 114, 113, 110, 109, 46, 45, 77, 77, 116, 115, 33, 33, 102, 101, 116, 115,
```

Fig. 8. ASCII values of decompressed data



```
with the increasing demand of secure multimedia, there is a need to develop a new secured compression and
encryption technique where data contains graphic, video, images and text files. Data compression offers an approach
for reducing communication costs using effective bandwidth and at the same time considers the security aspect of
the data being transmitted which is vulnerable to attacks. In this chapter we have introduced new compression
algorithm
known as KSA, which is based on bit quantization and is best technique which requires less encoding and decoding
delay. It is an optimized approaching encoding technique which offers good compression ratio which deals with both
the issues of size and security. Here the objective is to carry out an efficient implementation of the OFDM system
using different combinations of encryption and compression algorithm for the energy optimization on data
transmission. Lots of text encryption algorithms based on rounds and keys have been proposed earlier. Some of them
are time consuming and complex, some have little key space. while working the best combination for the encryption
and compression of the data transmission and energy optimization is found out and implemented in NS2 for finding
different parameters like delay, energy and throughput. The latest trend in text encryption is chaos based for some
unique characteristics and parameters. In this chapter we further proposed a unique chaos based encryption
technique which when combines with LZW compression giving best results.This encryption differs from earlier
encryption technique such as RSA, ECC, Interleaving,
Hill-climber, AES, DES and are suitable for practical applications having large data capacity. The combination of
compression, cryptography and chaotic theory forms an important field of information security in text.. Tabulation
of compression time, encryption time, decompression time, decryption time, compressed data size, total time taken
for compression, entropy, total time required for decompression and compression ratio have been found.
```

Fig. 9. ASCII to Original converted file

6 CONSTRUCTION AND WORKING

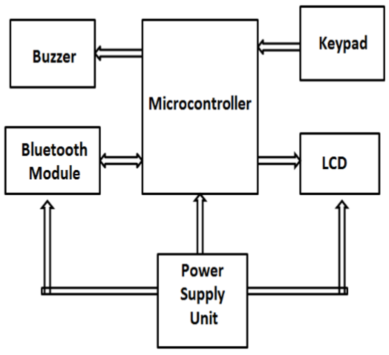


Fig 10. Block Diagram of complete hardware

The screen used is LCD (Liquid Crystal Display) is an HD44780 controller based electronic display module and are preferred over seven segments and other multi-segment LEDs. Working with LCD modules feels the real power of MCU and creations are touching to the sky. In this hardware 20X4 alphanumeric LCD is used which displays 20 characters in each row out of four and to interface with AVR ATmega328 microcontroller 8-bit and 4-bit interfacing method is used where all the eight data pins are used in 8-bit interfacing method, and in 4-bit interfacing method only the upper 4 data pins of the alphanumeric LCD are used to send 8-bit command from the microcontroller. Where human interaction or human input is needed keypads which are parts of HMI or Human Machine Interface plays an important role in embedded system. Whether any key is pressed or not or which key is pressed or for displaying message , interfacing of microcontroller with program is written. Program algorithm is written in such a way that, the microcontroller scans each row or column of the 4X4 keypad one by one continuously and when any key of a 4X4 keypad is pressed, the read_keypad function returns the value of the pressed key and the microcontroller displays it in the 16X2 alphanumeric LCD. If none of the keys is pressed the read_keypad function returns 0XFF and the LCD displays the value of the last pressed key.

Here Arduinos are connected to laptop and a HC-05 connected to Arduino and input data into the application . The message or information is sent to the HC-05 and consequently to the Arduino which assigns a number value to each character and when the arduino, receives these characters, it performs a function on the number, and prints the 'output' to the Serial Monitor.

The other Arduino used as the encrypter key provider to enter the password into the serial monitor of the other but one character at a time and if this is accepted the Arduino sends the encoded key to the Encrypting Arduino - which is actually high signal and unless the Encrypting Arduino, receives a HIGH signal, it does not encrypt the data.

8 Conclusion

Preventive security measures must be integrated into computer systems that are potential targets for malicious or defamatory attacks. This is particularly essential for systems handling financial transactions, classified data, or any other sensitive information where confidentiality and integrity are of utmost importance.

Cryptography has emerged as one of the most critical technologies in the IT security domain. The modern challenge is not only to implement robust cryptographic systems but also to ensure that IT infrastructures can efficiently manage, detect, and respond to data breaches and hacking attempts.

The encryption technique presented in this paper has been implemented and simulated using Java. Through practical demonstrations, the system shows how data is read, compressed, converted into ASCII, encrypted, and subsequently decrypted and decompressed. It also provides a clear view of the data size at various stages—original, compressed, encrypted, and decrypted—showcasing the effectiveness and efficiency of the approach.

This proposed cryptographic module combines a symmetric encryption strategy with data compression, forming a secure and lightweight solution that strengthens cryptographic analysis resistance. The complexity added through this integration makes it significantly more difficult for intruders or attackers to reverse-engineer or compromise the system.

Furthermore, by blending software techniques with theoretical cryptographic principles, the solution provides a formidable barrier against intrusion attempts. The secure transmission of confidential data over insecure media is thus successfully achieved, demonstrating the practical viability of the proposed method.

ACKNOWLEDGMENT

It is with great pleasure and sincere gratitude that I acknowledge the invaluable guidance and support of Prof. Dr. Manisha Sharma, Professor BIT Bhilai and Prof. Dr. S. G. Akojwar, Senior Member IEEE and Principal Government College of EngineeringRatnagiri .Their insightful suggestions, expert supervision, and constant encouragement played a pivotal role in the successful completion of this work. It has been a great honor and privilege to work under their mentorship.

REFERENCES

[1]. Mahindra Agrawal, "Cryptography: A Survey,"IETE Technical ReviewVolume 16,Issue 3-4,pp.287_96,March.2015.

[2]. M. Wherate, Dr. S. Sherekar, Dr. V. M. Thakre, "Two Layer Security Using Visual Cryptographyand Steganography," IETE 46th Mid Term Symposium on Impact of Technology on Skill Development" pp.92_5, April, 2015.

[3]. Huan Zhang, Xiao-ping Fan, Shao-qiang Liu, Zhi Zhong(2011) Design and Realization of Improved LZW Algorithm for Wireless Sensor Networks, International Conference on Information Science and Technology, IEEE ,pp.671-675.

[4]. YuanJing(2011) The Combinational Application of LZSS and LZW Algorithms for Compression Based On Huffman, International Conference on Electronics and Optoelectronics,IEEE, pp.397-399.

[5]. Kuo-Kun Tseng, JunMin Jiang, Jeng-Shyang Pan, LingLing Tang, Chih-Yu Hsu, Chih-Cheng Chen(2012) Enhanced Huffman Coding with Encryption for Wireless Data Broadcasting System, International Symposium on Computer, Consumer and Control, IEEE, pp. 622-625.

[6]. S. Karunakaran, D. Indumathy, Dr.R.Rani Hemamalini(2013) Implementation of Software Compression Techniques, International Conference on Smart Structures & Systems,IEEE,pp.193-196.

[7]. Depavath Harinath, M V Ramana Murthy , B Chitra(2015) Cryptographic Methods and Performance Analysis of Data Encryption Algorithms in Network Security,International Journal of Advanced Research in Computer Science and Software Engineering Volume 5, Issue 7, pp.680-688.

[8]. R. Chandramouli (2006) Battery power-aware encryption - ACM Transactions on Information and System Security (TISSEC), Volume 9 , Issue2.

[9]. A Performance Comparison of Data Encryption Algorithms, IEEE Information and Communication Technologies, ICICT 2005 First International Conference proceedings report, 2006, pp. 84- 89.

[10]. G. Ramesh Dr. R. Umarani (2012) Performance Analysis of Most Common Symmetrical Encryption Algorithms, International Journal of Power Control Signal and Computation(IJPCSC) ISSN: 0976-268X Volume 3.

[11] N. Khanna, J. Nath, J. James, A. Chakrabarti, S. Chakraborty A. Nath (2011) New symmetric key Cryptographic algorithm using combined bit manipulation and MSA encryption algorithm. NJJSAA symmetric key algorithm. International Conference on Communication Systems and Network Technologies, pp 125-130.

[12] George Makris , Ioannis Antoniou(2012) Cryptography with Chaos. Proceedings, 5th Chaotic

Modeling and Simulation Internationa Conference, Athens Greece, pp 309-497.

[13] E. Celikel and M. E. Dalkilic(2004) Experiments on A Secure Compression Al algorithm. IEEE Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'04) 0-7695-2108-8/04 .

[14]William Stallings(2005) Cryptography and Network Security: Principles and Practices. International 5th Edition, Prentice Hall.

[15]Y. K. Jain and P. B. Gosavi(2008) Email Security using Encryption and Compression. IEEE International Conference of Computational Intelligence for Modeling Control Automation, pp 136-139.