TEXT-BASED ENCRYPTION (TBEnc) TECHNIQUE TO SECURE DATA IN CLOUD STORAGE SERVER

B. Rex Cyril & Dr. S. Britto Ramesh Kumar

Research Scholar *Research Supervisor*

Assistant Professor *Assistant Professor*

St. Joseph's College *St. Joseph's College, Trichy.*

rexcyrilsjc@gmail.com

Abstract: ***Cryptographic concepts are generally used for protecting the data in travel or rest. This chapter has proposed a confidentiality technique namely, TBEnc for secured cloud storage. The proposed TBEnc is a symmetric encryption algorithm. This algorithm is delivered as a service to the user. If the users' want to encrypt their text-based data; they can use this algorithm to encrypt their data before they are uploaded to the cloud***.

Keywords: Cloud Computing, Security, Encryption, Text-based

1. Introduction

Security concerns have got more critical issues nowadays in the cloud. The data outsourced to the cloud is to be secured by using cryptography techniques. Cryptography techniques are generally used for security data using different parameters like confidentiality, integrity and authentication [Kel, 13]. Authentication is the first level security; if authentication is broken, then hackers easily get the data in travel or storage. It is needed to ensure that the pirates see even those data, they could not understand the original message while is in travel or in the server. Data security is achieved by using cryptography techniques; these techniques are preferred to ensure the safety of data stored in the cloud by using confidentiality parameter [Wil, 05].

Encryption techniques are the only possibility to ensure the security of data in rest. As it is discussed in the previous chapters, Encryption is categorized into two types, symmetric and asymmetric. As per the literature, symmetric encryption is suitable for the cloud environment. Traditional techniques are available in the cryptosystem, but each has some issues when it is implemented in the cloud environment. Cloud environment should be equipped with a well-established security system, and this chapter proposes an enhanced confidentiality technique using an encryption technique. The proposed algorithm is a symmetric encryption algorithm, and it considers the text-based data in the plaintext format. It is implemented as a cloud service in the proposed framework.

. From the literature review, it is observed that a novel security architecture is needed to ensure the privacy of the data stored in the cloud. This research work proposes a confidentiality technique (CT); namely, TBENC to safeguard the confidentiality of data in cloud storage. The proposed CT is based on the symmetric encryption algorithm.

2. Objectives of this chapter

This section describes the objective of the chapter.

- To propose an efficient symmetric encryption technique called TBEnc to secure text-based data sent to the cloud concerning security and performance.

3. Methodology

Cloud storage with outsourcing provides many benefits to users. Outsourced data are protected by using encryption technique. Earlier the data

were uploaded from users. In the proposed research, three confidentiality techniques are proposed with a framework called SECON. The proposed SECON framework contains three CTs to safeguard the data in the storage server. TBEnc is one of the CTs provided by EOaaS in SECON framework. Figure 3.1 shows the methodological diagram of TBEnc.

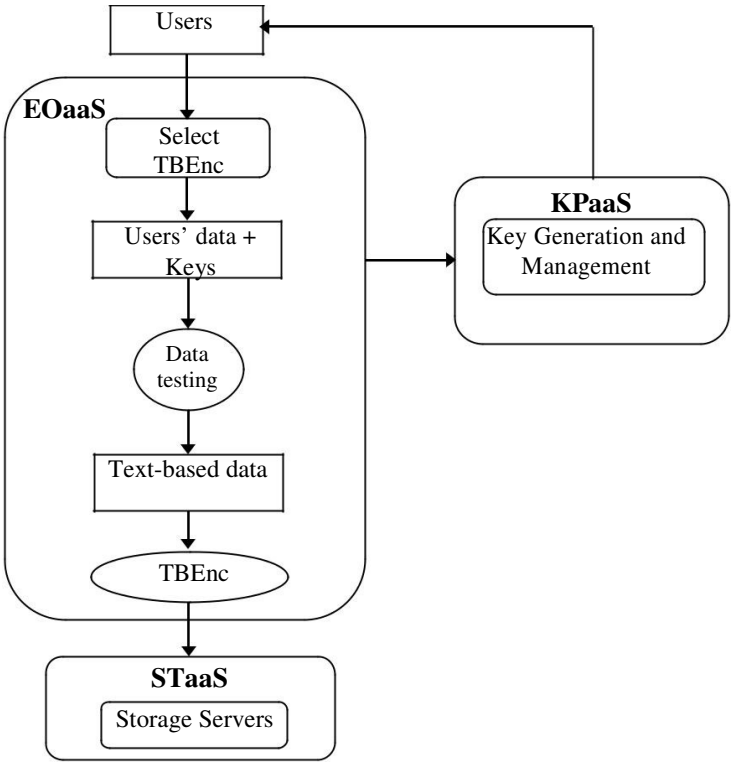

Figure 3.1. Process flow Methodology of TBEnc

TBEnc is an encryption algorithm which is executed when the users select this service for data security. TBEnc is applied to encrypt the data of text-based data type. Users data may be mixed with text-based data and value-based data, but TBEnc categorizes the data by testing the data to define the text-based data. Values-based data in the plaintext are not in the scope of this encryption TBEnc. Encryption is only possible with a key. This proposed

TBEnc also uses keys for encryption. The keys used for TBEnc are generated from KPaaS in SECON framework. Users receive the keys from the cloud and apply with TBEnc to encode the data. Data are uploaded to the cloud, once encryption is completed. The keys used for TBEnc are not communicated to the CSP.

4. Proposed TBEnc Algorithm

The proposed TBEnc is a symmetric encryption algorithm. This algorithm is delivered as a service to the user. If the users' want to encrypt their text-based data; they can use this algorithm to encrypt their data before they are uploaded to the cloud. Following section describes the step by step procedure involved in the proposed algorithm.

4.1. The procedure of TBEnc encryption algorithm:

Step 1. *Input considered as text values (Non-Numerical)*

Step 2. *Find the size of the input*

Step 3. *Input is converted into ASCII decimal, and it is further converted into binary values 0s and 1s*

Step 4. *Count no. of bits in the binary bits*

Step 5. *Write the bits in an 8x8 matrix form from left to right in row by row manner*

Step 6. *Form matrix for all bits and if no bits are remaining to fill the matrix use '0' to fill the remaining portion of the matrix. The matrices are formed based on the number of bits divided by eight.*

Step 7. Read the matrix values from top to bottom in a column by column manner

Step 8. Divide the binary bits into two blocks [block1 and block2] by fetching 2bits alternatively

Step 9. Generate the two Keys for K_1 and K_2 from Key Generation Cloud Service (KGCS)

Step 10. Calculate XOR of K_1, and every 8 bits in block1and also Calculate XOR of K_2 and every 8 bits in block2

Step 11. Keys K_1 and K_2 are incremented by one for each next 8 bit values in block1 and block2

Step 12. Calculate 1's complement for each block

Step 13. Merge the block1 and block2 in the same position

Step 14. Divide the whole binary bits into 8bits block

Step 15. Convert the binary into decimal and convert the decimal into ASCII character code to produce the Ciphertext.

4.2. Pseudo code for TBEnc Encryption

Algorithm:- *tbenc(PLNTXT)*

Inputs:- *Users' sensitive data*

Declarations

PLNTXT ← *Plaintext*

N ← *Size of PLNTXT*

ASC ← *ASCII decimal codes of PLNTXT*

BNRY ← *ASCII Binary code for ASC*

BUFF ← *Buffer Variable for binaries*

CNT ← *Count the bits in the BUFF*

NMAT ← *Numbers of Matrix*

MAT ← *Matrix of 8x8 size*

BINBITS ← *Binaries read from MAT*

Block_1 ← *alternatively fetched two binary values from BINBITS*

Block_2 ← *alternatively fetched two binary values from BINBITS*

NB1 ← *Number of 8bit blocks in Block_1*

NB2 ← *Number of 8bit blocks in Block_2*

BK1BIN ← *Divided 8 bits from Block_1*

BK2BIN ← *Divided 8 bits from Block_2*

XOR_BK1 ← *Result of XOR with BK1BIN and* K_1

XOR_BK2 ← *Result of XOR with BK2BIN and* K_2

NB1_OC ← *One's Complement of XOR_BK1*

NB2_OC ← *One's Complement of XOR_BK2*

BUF_MERGE ← *the Merged result of NB1_OC and NB2_OC*

LBUF ← *Number of 8 bits blocks in BUF_MERGE*

K_1 & K_2 ← *Keys for Encryption and Decryption*

FBIN ← *8 bits binary block of BUF_MERGE*

DECI ← *Decimal value of 8bit binary*

CT_BUF ← *Buffer Variable to append the decimal values*

```
CT ← Cipher text
1. start
2. N ← sizeof(PLNTXT) //find the size of the Plain Text
3. for i ← 1 to N
        ASC[i] ← asciidecimal(PLNTXT[i]) //convert into ASCII
        Decimal BNRY[i] ← binary(ASC[i]) //convert into binary values
        0s and 1s BUFF ← append(BNRY[i]) // Buffering binaries
4. next i
5. CUNT ← count(BUFF)
6. NMAT ← CUNT/64
7. k ← 1
   // Write the bits in a 8x8 matrix form from left to right and row by row
8. for m ← 1 to NMAT
        for i ← 1 to 64
            l ← 1
            for j ← i to i+7
                if (!isempty(BUFF))
                    MATm[k][l] ← BUFF[j]
                else
                    MATm[k][l] ← 0
                l ← l+1
            next j
```

```
                k=k+1
                i=i+7
            next i
9.  next m
    // Read the matrix values from left to right and column by column
10. for m ← 1 to NMAT
            for i ← 1 to 64
                l ← 1
                for j ← i to i+7
                    BINBITS ← append(MATm[l][k])
                    l ← l+1
                next j
                k=k+1
                i=i+7
            next i
11. next m
12. j ← 3
13. for i ← 1 to sizeof(BINBITS)
            Block_1 ← retrievetext(BINBITS(i),BINBITS(i+1))
            Block_2 ← retrievetext(BINBITS(j), BINBITS(j+1))
            i ← i+4
            j ← j+4
```

next i

14. *NB1* ← *sizeof(Block_1)/8 // to find the no. of 8bits in block*
15. *NB2* ← *sizeof(Block_2)/8*
16. *i* ← *1*
17. *j* ← *1*
18. *while(i<NB1)*

BK1BIN[i] ← *split(block_1(j), 8) //split the binaries into 8bits block*

BK2BIN[i] ← *split(block_2(j), 8)*

i++

j ← *j+8*

19. *loop*
20. *Generate two keys for* K_1 *and* K_2
21. *for i* ← *1to NB1*

XOR_BK1 ← *BK1BIN[i]* $\oplus$ K_1

K_1 ← K_1+1

22. *next i*
23. *for i* ← *1to NB2*

XOR_BK2 ← *BK1BIN[i]* $\oplus$ K_2

K_2 ← K_2+1

24. *next i*

25. *NB1_OC ← once_complement(XOR_BK1) //method to calculate one's complement*
26. *NB2_OC ← once_complement (XOR_BK2)*
27. *BUF_MERGE ← merge(NB1_OC,NB2_OC)*
28. *LBUF ← sizeof(BUF_MERGE)/8 // to find the no. of 8bits blocks*
29. *i ← 1, j ← 1*
30. *while(i<LBUF)*

    FBIN[i] ← split((j), 8) //split the binaries into 8bits block

    i++

    j ← j+8
31. *loop*
32. *for i ← 1 to LBUF*

    DEC[i] ← asciidecimal(FBIN[i]) // convert the 8bits into decimal CT_BUF ← append(ascii(DECI[i]))
33. *next i*
34. *CT ← CT_BUF*
35. *Stop*

4.3. Sample experiment for TBEnc

The proposed TBEnc is evaluated with a sample data. Step by step procedure of the algorithm is described in detail below.

Sample text considered for experiment is **"I will meet you tomorrow."** This text is given as the input to the proposed TBEnc and generates ciphertext as per the procedure described in the pseudo code.

4.3.1. Encryption Procedure

Sample Text: ***I will meet you tomorrow***

Step 1. Plain Text as Input

$PLNTXT \leftarrow$ ***I will meet you tomorrow***

Step 2. Find the size of input

$N \leftarrow sizeof(PLNTXT) = 24$

Step 3. Input is converted into ASCII decimal

I $\leftarrow 73$	***w*** $\leftarrow 119$	***i*** $\leftarrow 105$	***l*** $\leftarrow$ ***108***	***l*** $\leftarrow$ ***108***
m $\leftarrow 109$	***e*** $\leftarrow 101$	***e*** $\leftarrow 101$	***t*** $\leftarrow 116$	***y*** $\leftarrow 121$
o $\leftarrow 111$	***u*** $\leftarrow 117$	***t*** $\leftarrow 116$	***o*** $\leftarrow 111$	***m*** $\leftarrow 117$
o $\leftarrow 111$	***r*** $\leftarrow 114$	***r*** $\leftarrow 114$	***o*** $\leftarrow 111$	***w*** $\leftarrow 119$

Step 4. It is further converted into binary values 0s and 1s

$73 \leftarrow 01001001$	$32 \leftarrow 00100000$	$119 \leftarrow 01110111$
$105 \leftarrow 01101001$	$108 \leftarrow 01101100$	$109 \leftarrow 01101101$
$101 \leftarrow 01100101$	$116 \leftarrow 01110100$	$121 \leftarrow 01111001$
$111 \leftarrow 01101111$	$117 \leftarrow 01110101$	$114 \leftarrow 01110010$

Step 5. Count total numbers of bits in the binary

$CUNT \leftarrow 192$ *bits*

Step 6. Write the bits in a 8x8 matrix form from left to right in row by row manner. Number of matrices is formed based on the number of bits divided by 64.

Number of Matrix = 192/64 = 3 matrix

MAT1[][]	***MAT2[][]***	***MAT3[][]***
01001001	*01100101*	*01110100*
00100000	*01100101*	*01101111*
01110111	*01110100*	*01101101*
01101001	*00100000*	*01101111*
01101100	*01111001*	*01110010*
01101100	*01101111*	*01110010*
00100000	*01110101*	*01101111*
01101101	*00100000*	*01110111*

Step 7. Read the matrix values from top to bottom in a column by column manner

BINBITS ← *000000001011110101111111100100000100111010010110 100100000101100010000000011101110111111111001010100000110 011100110000001001100111000000000111111111111111111000110 101110010111100110101111101110011*

Step 8. Divide the binary bits into two blocks [block1 and block2] by fetching 2bits alternatively

BK1BIN ← *0000110111111000010110011000110100001010111110100000101000000010000011111111000111101111011111111*

BK2BIN ← *0000101101110000101100110000100000001111111100100011110100011110000111111111011010011000111010100*

Step 9. Separate two blocks into 8-bit binaries

BK1BIN ← *00001101 11111000 01011001 10001101 00001010 11111010 00001010 00000010 00001111 11110001 11101111 01111111*

BK2BIN ← *00001011 01110000 10110011 00001000 00001111 11110010 00111101 00011111 00001111 11111011 01001100 01110100*

Step 10. Generate the two Keys for K_1 and K_2 from Key Generation Cloud Service (KGCS)

Sample Key K_1 ← *201 K_2* ← *97*

Step 11. Calculate XOR of K_1, and every 8 bits in block1and also Calculate XOR of K_2 and every 8 bits in block2, Keys K_1, and K_2 are incremented by one for each next 8 bit values in block1 and block2.

XOR_BK1 ← *11000100 00110010 10010010 01000001 11000111 00110100 11000101 11010010 11011110 00100011 00111100 10101011*

XOR_BK2 ← *01101010 00000010 11010000 01101100 01101010 10010100 01011010 01110111*

01100110 10010001 00100111 00011000

Step 12. Calculate 1's complement for each block

NB1_OC ← *00111011 11001101 01101101 10111110*
00111000 11001011 00111010 00101101

00100001 11011100 11000011 01010100

NB2_OC ← *10010101 11111101 00101111 10010011*
10010101 01101011 10100101 10001000

10011001 01101110 11011000 11100111

Step 13. Merge the block1 and block2 in same position

BUFF_MERGE ← *10000111011001111111110011110101000110101 1*
11110110100111001111101000011101100100011110001010111110

00101101100110100000101011000110000110100001010111100111

11100011110100100000111101100101011100

Step 14. Divide the whole binary bits into 8bits block

BUFF_MERGE ← *10000111 01100111 11111100 11110101*
00011010 11111101 10100111 00111110

10000111 01100100 01111000 10101111

10001011 01100110 10000010 10110001

10000110 10000101 01111001 11111000

11110100 10000011 11011001 01011100 Step 15.

Convert the binary into decimal

DEC ← *135 103 252 245 26 253 167 62 135*
100 120 175 139 102 130 177 134 133
121 248 244 131 217 92

Step 16. Convert the decimal into ASCII character code to produce the Cipher text.

CT ← ç g ⁿ ⌡ → ² º = ç d
x » ï f é ▒ å à y °
⌠ â ┘ \

Encrypted Text stored in cloud storage is,

Cipher Text = ç g ⁿ ⌡ → ² º = ç d x » ï f é ▒ å à y ° ⌠ â ┘ \

4.3.2. Decryption Procedure

The proposed technique is a symmetric encryption algorithm, which uses the same key for encrypting and decrypting the data in the cloud. This section describes the decryption procedure of proposed TBEnc. Decryption is reverse process encryption using the same key. Users should safeguard the key and the same key is used here for decrypting the data from the cloud

Consider the sample ciphertext for Decryption

Cipher Text = ç g ⁿ ⌡ → ² º = ç d x » ï f é ▒ å à y ° ⌠ â ┘ \

Step 1. Convert the ciphertext into ASCII decimal code

DEC ← *135 103 252 245 26 253 167 62 135*
100 120 175 139 102 130 177 134 133
121 248 244 131 217 92

Step 2. Convert ASCII decimal code into corresponding 8 bits Binaries

BUFF_MERGE ← *10000111 01100111 11111100 11110101 00011010 11111101 10100111 00111110*

10000111 01100100 01111000 10101111

10001011 01100110 10000010 10110001

10000110 10000101 01111001 11111000

11110100 10000011 11011001 01011100 Step 3.

Make the 8bits binaries into single block

BINBITS ← *10000111011001111111110011110101000110101111111011*

01001110011111010000111011001000111100010101111100010110110011010000010101100011000011010000101011110011111100011110100100000111101100101011100

Step 4. Divide the binary bits into two blocks [block1 and block2] by fetching alternatively taken 2bits

BK1BIN ← *00111011 11001101 01101101 10111110 00111000 11001011 00111010 00101101*

00100001 11011100 11000011 01010100

BK2BIN ← *10010101 11111101 00101111 10010011 10010101 01101011 10100101 10001000*

10011001 01101110 11011000 11100111 Step 5.

Calculate 1's complement for each block

NB1_OC ← *11000100 00110010 10010010 01000001*

11000111 00110100 11000101 11010010

11011110 00100011 00111100 10101011

NB2_OC ← *01101010 00000010 11010000 01101100*
01101010 10010100 01011010 01110111

01100110 10010001 00100111 00011000

Step 6. Use the same key K_1 and K_2

Sample Key K_1 ← *201 K_2* ← *97*

Step 7. Calculate XOR of K_1 and every 8 bits in block1and also Calculate XOR of K_2 and every 8 bits in block2, Keys K_1, and K_2 are incremented by one for each next 8 bit values in block1 and block2

BK1BIN ← *00001101 11111000 01011001 10001101*

00001010 11111010 00001010 00000010 00001111 11110001 11101111 01111111

BK2BIN ← *00001011 01110000 10110011 00001000 00001111*

11110010 00111101 00011111 00001111 11111011 01001100 01110100.

Step 8. Merge each 8 bits block into single chunk in both blocks

BK1BIN ← *0000110111111000010110011000110100001010111110100000101000000010000011111111000111101111011111111*

BK2BIN ← *0000101101110000101100110000100000001111111100100011110100011110000111111111101101001100011101000*

Step 9. Merge the block1 and block2 in the same position

BINBITS ← *000000001011110101111111001000001001110100101101001000001011000100000000111011101111111100101010000011001111000001001100111000000000011111111111111111100011010111001011110011010111110111001101*

Step 10. Count total numbers of bits in the

binary CUNT ← *192 bits*

Step 11. Write the bits in an 8x8 matrix form from top to bottom column by column manner. The matrices are formed based on the number of bits divided by 64.

Number of Matrix = 192/64 = 3 matrix

MAT1[][]	***MAT2[][]***	***MAT3[][]***
01001001	*01100101*	*01110100*
00100000	*01100101*	*01101111*
01110111	*01110100*	*01101101*
01101001	*00100000*	*01101111*
01101100	*01111001*	*01110010*
01101100	*01101111*	*01110010*
00100000	*01110101*	*01101111*
01101101	*00100000*	*01110111*

Step 12. Read the matrix from left to right and row by row from each matrix

01001001 00100000 01110111 01101001 01101100

01101100 00100000 01101101 01100101 01100101

01110100 00100000 01111001 01101111 01110101

00100000 01110100 01101111 01101101 01101111

01110010 01110010 01101111 01110111 Step

13. Find the ASCII Decimal value for the Binaries

73 ← *01001001 32* ← *00100000 119* ← *01110111*

105 ← *01100111 108* ← *01101100 109* ← *01101101*

101 ← *01100101 116* ← *01110100 121* ← *01111001*

111 ← *01101111 117* ← *01110101 114* ← *01110010*

Step 14. Convert the decimal into ASCII character code

PLNTXT ← ***I will meet you tomorrow***

4.5. Experiment Result analysis

The proposed technique is experimented with sample data. Encryption procedure is described step by step and it produces the cipher text.

Sample plaintext: *I will meet you tomorrow*

Ciphertext produced by the proposed algorithm is as shown below,

Cipher Text = ç *g* ⁿ ⌡ → ² º = ç d x » ï f é ▒ å à y ° ⌠ â ┘ \

The ciphertext is analyzed in the way of checking whether the proposed techniques produce the same ciphertext for same plain text or not.

It can be seen that these characters, w, t, o, l, e, and r appear more than once in the plain text in different places. The ciphertext produced by the proposed algorithm has different ciphertext for the same plaintext that appeared more than once.

Character ‘w’ has appeared in two places of plaintext in 3rd and 24th place but for the same alphabet ‘w’ in ciphertext, it is ⁿ \. Table 3.1 shows all other characters that have appeared more than once in the plaintext.

Cipher Text = ç *g* ⁿ ⌋ → ² ° = ç d x » ï f é ▒ å à y ° ⌈ â ┘ \

Table 3.1 Result analysis for proposed algorithm TBEnc

The character appeared more than once in the plain text	Character position the plaintext	Corresponding ciphertext for the same character and in the same position
W	3	ⁿ
	24	\
T	11	x
	17	å
O	14	f
	18	à
	20	°
	23	┘
L	5	→
	6	²
E	9	ç
	10	d
R	21	⌈

	22	â
Space	2	*g*
	7	º
	12	»
	16	▒

Table 3.1 shows that for the same plaintext which has appeared more than once; the proposed algorithm TBEnc produces different ciphertext. In the same way, the character '*ç*' is produced as cipher text for two different characters 'I' and 'e' in plaintext. It will confuse the hackers when attacking the data using any attacks like Dictionary or Brute force attack. Cryptanalyst could not get any idea or pattern to get the original data stored in the cloud. Hence, the proposed TBEnc strengthen the security of data stored in the cloud storage. The legitimate users could only access the cloud data stored in the server.

4.6. Simulation Result

The proposed TBEnc and key generation are developed as a cloud-based application using C#.Net. Simulation is conducted in a real-time environment. The cloud server is rented for simulating the whole research work. A cloud server is borrowed from Windows Azure Cloud Infrastructure and Platform as a Service. Procedure for hosting a Cloud server from Window Azure platform is described in Appendix II. Performance of the proposed TBEnc is calculated based on the time taken for encryption and decryption.

The security level of the proposed TBEnc is analyzed by using a security analysis tool called ABC Universal Hackman Tool. Security level is measured based on this tool for proposed and existing techniques. A cloud server is installed with this tool for analyzing the security level of proposed and existing symmetric encryption algorithms such as DES, 3DES, and Blowfish. Hackman tool attacks the encrypted text in the cloud server. It uses different attacks like dictionary and brute force attack to retrieve the original version. At the end of retrieval, Hackman compares the plain text with extracted text to find the percentage of initial text retrieval. Based on the percentage of comparison, the security level of the proposed algorithm is measured. In the same way, the security level of existing cryptographic techniques is calculated and compared with the proposed TBEnc.

Performance and security level of the proposed TBEnc is compared with existing techniques. Performance of the proposed TBEnc is calculated by the time taken for encryption and decryption. The proposed method is measured with different sizes of data. Time taken for each volume of data for encryption, decryption and security level is estimated and evaluated with existing techniques.

Table 3.2 and Figure 3.2 represent execution time taken for encryption by the proposed and existing encryption techniques. From the result, it is proved that the proposed TBEnc has taken minimum time duration for encryption than existing encryption techniques.

Table 3.2 Performance Comparison Based on Encryption Time

Size	Algorithms			
	DES	3DES	Blowfish	TBEnc
	(Milliseconds)			
1 MB	492	608	387	230
2 MB	967	1068	592	458
3 MB	1292	1412	881	656
4 MB	1691	1837	1063	879
5 MB	2098	2226	1197	1092
10 MB	4272	4394	2411	2243
15 MB	6321	6587	3632	3378

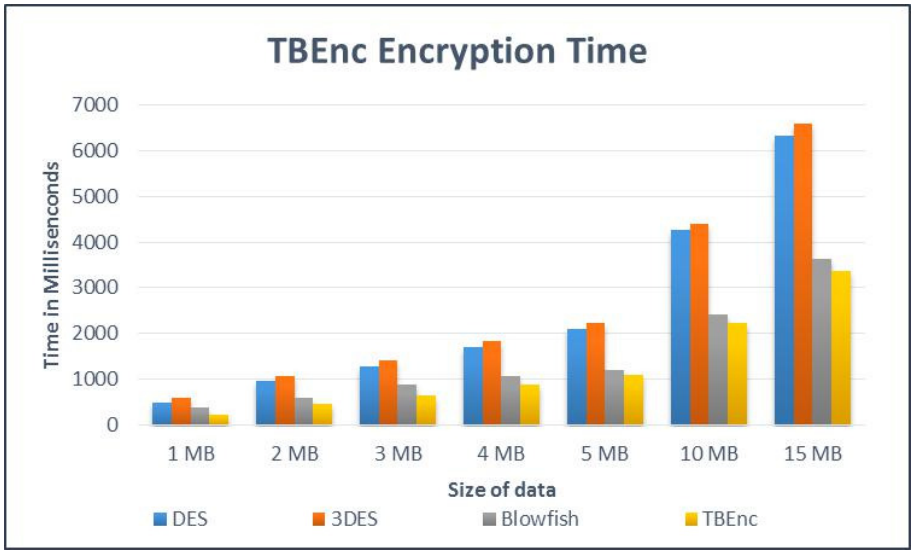

Figure 3.2 Performance Comparison Based on Encryption Time

Table 3.3 and Figure 3.3 represent execution time taken for decryption by the proposed and existing encryption techniques. From the result, it is

proved that the proposed TBEnc has taken minimum time duration for decryption than existing decryption techniques.

Table 3.3 Performance Comparison Based on Decryption Time

Size	Algorithms			
	DES	3DES	Blowfish	TBEnc
	(Milliseconds)			
1 MB	487	597	302	225
2 MB	948	1052	582	428
3 MB	1279	1393	827	617
4 MB	1679	1822	986	833
5 MB	2074	2209	1160	1059
10 MB	4106	4381	2221	2206
15 MB	6288	6568	3502	3331

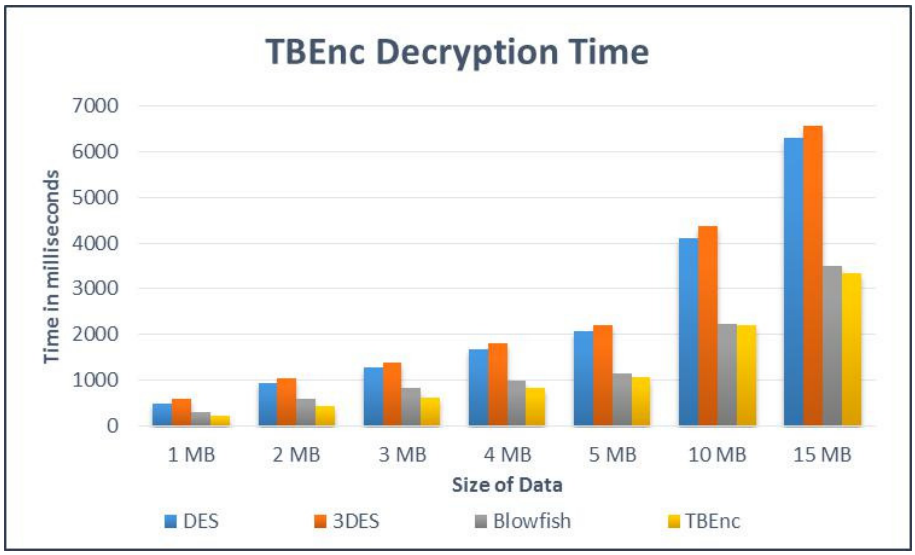

Figure 3.3 Performance Comparison Based on Decryption Time

Table 3.4 Comparison of Security Levels of Existing and Proposed Encryption Algorithms

S. No	Algorithm(s)	Security Level (%)
1.	DES	77
2.	3DES	85
3.	Blowfish	82
4.	TBEnc	91

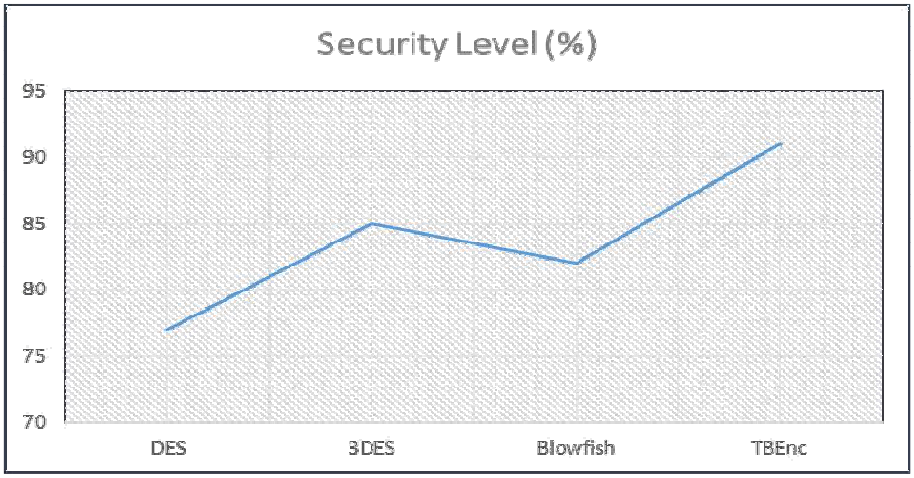

Figure 3.4 Comparison of Security Levels of Existing and Proposed Encryption Algorithms

4.8. Research Findings and Interpretations

The proposed TBEnc is a symmetric encryption algorithm. It uses similar keys for execution of encryption and decryption. Keys for TBENC are received from KPaaS. TBEnc takes minimum time duration concerning the performance matched with an existing technique. TBEnc allows users to

encrypt only text-based data. Users select TBEnc in EOaaS to encrypt text-based data. No one can disclose encrypted data stored in the server. Ciphertext generated from TBEnc makes confusion to cryptanalyst to retrieve plain text.

4.9. Conclusion

Cloud data outsourcing helps to improve the business of the SMEs at a low cost. Security of data most viewable in the cloud are to be addressed. Cryptographic concepts are generally used for protecting the data in travel or rest. This chapter has proposed a confidentiality technique namely, TBEnc for secured cloud storage. TBEnc is mainly focused on the data stored in cloud storage. It is one of the confidentiality techniques in EOaaS. TBEnc is provided through EOaaS in SECON secured confidentiality framework. The TBEnc is a symmetric encryption algorithm.

This proposed Confidentiality Techniques helps to protect the user data. TBEnc enables users to ensure the security of in cloud server and also it is proved that internal attackers cannot access the data. Even though they are privileged administrators; they cannot understand the original message encrypted in the data. A real-time environment is set up for the simulation to find and compare the proposed and existing technique concerned with performance and security level. TBEnc protects non-numeric data stored on a cloud server. The next chapter introduces another confidentiality technique called VBObfus based on obfuscation technique to secure the value-based data in the cloud.

REFERENCES

1. Sascha Fahl and Marian Harbach, “Confidentiality as a Service – Usable Security for the Cloud”, Proceedings of IEEE International Conferenceon Trust, Security and Privacy inComputing and Communications, 2012, pp. 153-162.

2. Satyendra Singh Rawat and Niresh Sharma, “A Survey of Various Techniques to Secure Cloud Storage”, International Journal of ComputerScience and Network Security, Volume 12 Issue 3, 2012, pp. 116-121.

3. Pardeep Sharma, Sandeep K. Sood, and Sumeet Kaur, “Security Issues in Cloud Computing”, High Performance Architecture and GridComputing Communications in Computer and Information Science, Volume 169, 2011, pp 36-45.

4. Shirole Bajirao Subhash and Dr Sanjay Thakur, “Data Confidentiality in Cloud Computing with Blowfish Algorithm”, International journal of Emerging Trends in Science and Technology, Volume 1, Issue 1, 2014, pp. 01-06.

5. Shucheng Yu, Wenjing Lou, and Kui Ren, “Data Security in Cloud Computing”, Handbook on Securing Cyber-Physical Critical Infrastructure, Chapter 15, Elsevier, Morgan Kaufmann Publisher, 2012, pp. 389-410.

6. Shweta Kaushik, Charu Gandhi. "Cloud data security with hybrid symmetric encryption", 2016 International Conference on Computational Techniques in Information and Communication Technologies (ICCTICT), 2016.

7. Siani Pearson, Yun Shen and Miranda Mowbray, “A Privacy Manager for Cloud Computing”, Proceedings of International Conference on Cloud Computing, Springer-Verlag Berlin, Heidelberg, LNCS Volume 5931, 2009, pp. 90-106.

8. Subashini S and Kavitha V., “A Survey on Security Issues in Service Delivery Models of Cloud Computing”, Elsevier Journal of Network and Computer Applications, Volume 34, Issue 1, 2011, pp. 1-11.

9. Subhasri P. and Padmapriya A., “Multilevel Encryption for Ensuring Public Cloud”, International Journal of Advanced Research in Computer Science and Software Engineering, Volume 3, Issue 7, 2013, pp. 527-532.

10. Sudha M and Monica M, “Enhanced Security Framework to Ensure Data Security in Cloud Computing Using Cryptography”, Advances inComputer Science and its Applications, Volume 1, Issue 1, 2012, pp. 32-37.

11. Atiq, U.R. and M. Hussain, “Efficient cloud data confidentiality for DaaS”, International Journal of Advanced Science and Technology, volume 35, 2011, pp.1-10.

12. Mather T., Kumaraswamy S. andShahed, L., “Cloud security and privacy”, Chapter 4, O’Reilly Media, Inc, 2009, pp.61-71.

13. William, S., “Cryptography and network security: principles & practices”, Fifth edition, Prentice Hall, 2005, pp. 6-56.

14. Sascha, F., Marian, H., Thomas, M and Matthew, S., “Confidentiality as a service – usable security for the cloud”, IEEE International Conference on Trust, Security and Privacy in Computing and Communications, 2012, pp. 153-162.

AUTHORS PROFILE

Prof. B. Rex Cyril is working as Assistant Professor and pursuing doctor of philosophy in Department of Computer Science, St. Joseph’s College,(Autonomous),Tiruchirappalli, Tamil Nadu, India. He received his M.Phil degree from Prist University. He received his MSc degree from St. Joseph’s College, Tiruchirappalli. His area of interest is Cloud Security Services. He has published research papers in the National/ International Conferences and Journals

Dr. S.Britto Ramesh Kumar is working as Assistant Professor in the Department of Computer Science, St. Joseph’s College (Autonomous), Tiruchirappalli, Tamil Nadu, India He has published many research articles in the National/International conferences and journals. His research interests include Cloud Computing, Data Mining, Web Web Mining, and Mobile Network