Journal of King Saud University – Computer and Information Sciences 34 (2022) 1417–1425

Contents lists available at ScienceDirect

Journal of King Saud University – Computer and Information Sciences

journal homepage: www.sciencedirect.com

BDNA-A DNA inspired symmetric key cryptographic technique to secure cloud computing

Manreet Sohal [*], Sandeep Sharma

Department of Computer Engineering and Technology, Guru Nanak Dev University, Amritsar 143005, India

A R T I C L E I N F O

Article history:
Received 26 April 2018
Revised 19 September 2018
Accepted 28 September 2018
Available online 29 September 2018

Keywords:
Cloud computing
Security
Cryptography
BDNA

A B S T R A C T

Cloud computing facilitates the storage and management of huge volumes of data. It offers flexibility for retrieving the data anytime and anywhere. In recent years, storing data onto the cloud achieved fame among corporations as well as private users. Although, the cloud is drawing a lot of attention still there are data security, privacy, reliability and interoperability concerns that need to be taken care. To deal with these issues, cloud data encryption comes to the rescue. Encrypting the data before uploading it onto the cloud prevents unauthorized users from accessing the data. A lot of encryption algorithms have been developed to secure data stored on the cloud. In this paper, a novel cryptographic technique has been presented that uses client-side data encryption for encrypting the data before uploading it onto the cloud. It is a multifold symmetric-key cryptography technique which is based upon DNA cryptography. Besides presenting the detailed design of our approach, we have compared it with the existing symmetric-key algorithms (DNA, AES, DES and Blowfish). The experimental results illustrate that our proposed algorithm outperforms these traditional algorithms in terms of ciphertext size, encryption time and throughput. Hence, the newly proposed technique is more efficient and offers better performance.

1. Introduction

Cloud Computing is quite proficient in providing infinite virtualized resources as services to the customers across the entire internet without providing any platform and implementation details. It is only because of cloud computing, that today the storage and parallel computing resources are available to the users at comparatively low costs. Since the cloud computing has become ubiquitous, a large amount of sensitive data is being concentrated on the cloud. It allows various users to share the data by means of specific access rights which vary from user to user. But, since data owners and cloud storage are not located in the same trusted domain, it has become mandatory to encrypt the data before uploading it onto the cloud (Liu et al., 2018; Yang et al., 2018; Paladi et al., 2017; Michalas, 2016). Encryption algorithms play a major role in providing data security. The major task of encryption algorithms is to guarantee privacy and security (Kaaniche and Laurent, 2017; Nie et al., 2010). The encryption algorithms are classified as symmetric and asymmetric key algorithms. In symmetric encryption, the same key is used for encryption as well as decryption while in asymmetric encryption public key is used for encryption and a private key is used for decryption. The approach presented in this paper is a symmetric key algorithm which uses client-side cloud data encryption (shown in Fig. 1). The cloud data encryption resolves many of the control issues that the enterprises come across while using the cloud (Rouse, 2014). Even if the cloud service providers are forced to reveal the data, the data which has been encrypted cannot be read by the unauthorized users, provided that the enterprises retain control of their encryption keys. Since today, a large number of cryptographic algorithms have been presented but all of these are based upon the conventional cryptographic algorithms (Kamara et al., 2012; Dowsley et al., 2017). All these algorithms have a strong mathematical and theoretical foundation. But, till date, the traditional approaches like DES, AES, Blowfish, RSA, DNA cryptography etc are still in operation for the real-time systems. But, these traditional approaches are consuming enormous amounts of computing resources. So, it is required to develop an insight that the new cryptographic techniques generate a bridge between the existing and the new technology.

* Corresponding author.
E-mail address: manreet.cetrsh@gndu.ac.in (M. Sohal).

Peer review under responsibility of King Saud University.

Production and hosting by Elsevier

https://doi.org/10.1016/j.jksuci.2018.09.024
1319-1578/© 2018 The Author(s). Published by Elsevier B.V. on behalf of King Saud University.

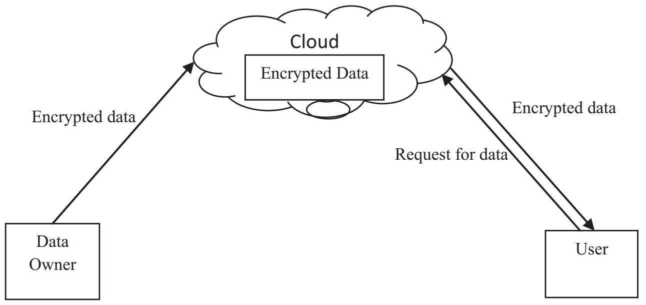


Fig. 1. Client-side Cloud Data Encryption.

The work presented in this paper is a DNA inspired multifold cryptographic technique (Sohal and Sharma, 2017) which enhances the security of the cloud data. This approach will strengthen the existing security systems by opening up a new possibility of a hybrid cryptographic system (Sohal and Sharma, 2017). DNA Cryptography has many computational drawbacks, as high technology laboratories are required for the actual implementation of this technology. The automation of DNA processes cannot be done. For the DNA synthesis, human reprocessing is required at every phase which has become a stumbling block in its extensive development. Therefore, to overcome the problems inherent in DNA cryptography, pseudo DNA cryptography techniques have gained popularity which work like DNA synthesis, but does not involve the actual synthesis of DNA bases in the laboratories. The approach presented in this paper is such pseudo DNA technique which is inspired by the way DNA cryptography works but it does not use DNA cryptography. Our approach is a symmetric key algorithm which works purely on binary data and hence, it is named as BDNA or the binary version of DNA cryptography.

The fundamental requirement for every encryption algorithm is that it should provide high security. But, these algorithms consume a considerable amount of computing resources such as CPU time, memory, and battery power. Consequently, it becomes very crucial to evaluate the performance of these algorithms. Therefore, in this paper, besides presenting the design of our algorithm, we have also evaluated the performance of our proposed algorithm by comparing it with the existing symmetric key algorithms.

The rest of the paper is organized as follows: Section 2 presents the related work, in Section 3 various symmetric encryption techniques have been discussed briefly, Section 4 provides the overview of the proposed system model, Section 5 discusses the proposed cryptographic algorithm in details, Section 6 contains comparison and results and finally, Section 7 concludes this paper.

2. Related work

Sharma and Garg (2016) have discussed the oldest symmetric key algorithm, DES and have highlighted its major flow, history, its implementation and its major drawbacks. Meyers and Desoky (2008) have examined a windows tool that uses blowfish algorithm for the encryption of files. The results obtained from the tool clearly exhibit that blowfish is faster as compared to subkey and S-box generations. The authors have further examined the cryptosystem security with the help of various test files. Nie et al. (2010) have compared and analyzed the encryption speed and power consumption of blowfish algorithm and have concluded that blowfish is faster than the DES in terms of encryption speed. But, the power consumption of both the schemes is almost similar. The authors have hence concluded that the blowfish is more appropriate for use in the security of wireless networks as compared to DES.

Kumar and Rana (2016) have presented an enhanced version of AES by increasing the number of rounds to 16. The authors have claimed that the resulting system provides high speed as well as greater security as compared to other algorithms. Rajput et al. (2016) have discussed the use of AES algorithm from the perspectives of cloud security. The authors have concluded that the memory requirements of AES are very less therefore, it can be used in environments having less space like 8-bit microprocessors. Moreover, it can be implemented in both hardware and software without any degradation in its performance. Bhardwaj et al. (2016) have discussed the various symmetric and asymmetric algorithms but, the major focus has been laid on symmetric algorithms. The authors have compared various symmetric algorithms and have suggested the suitability of different algorithms according to the type of cloud application. The authors evaluated symmetric algorithms for different encryption and encoding techniques and came up with the conclusions that AES is a good contender for key encryption.

3. Symmetric key algorithms used in our experiments

3.1. DES (Data Encryption Standard)

It is a symmetric key algorithm which uses a block cipher of 64 bits. Out of these 64 bits, 56 bits are used for actual encryption process while rest of the 8 bits is used for either padding or for checking the parity (Furht, 2006; Sharma and Garg, 2016; Pansotra and Singh, 2015). The encryption process involves initial permutation, final permutation, and 16 Fiestel cipher rounds. After applying the initial permutation, the 64-bit data is first divided into two parts of 32 bits each. Each of the 16 rounds involves performing Fiestel functions which involve substitution, permutation functions, and key mixing. The output of this function is then XORed with the other half of the data. These steps are repeated 16 times and the final permutation is applied. The output obtained from the final permutation is the final ciphertext. The decryption process is exactly the reverse of the encryption process. The operations of DES encryption are described as follows (Encyclopedia of Multimedia; Biryukov and Cannière, 2011).

a. First of all, a transposition is applied on 64 bit plaintext.
b. Then 16 fiestel rounds are applied. All these rounds perform same operations but on different functions of the key. These operations involve splitting of plaintext into two 32 bit inputs. Then, a two 32 bits output is obtained.
c. The first 32 bits are copied as such to form first half of the output.
d. A function of the last 32 bits and the key of that round are executed. This function involves performing substitutions and permutations.
e. Then Xor operation of this result is performed with the first 32 bits. This operation yields the second half of the output.
f. In second last step the first half of the output is interchanged with the second half of the output
g. The last step consists of the inverse transposition of the first step.

3.2. AES (Advanced Encryption Standard)

AES or Rijndael was announced by NIST in 2001 (Furht, 2006; Kumar and Rana, 2016). It is a symmetric key algorithm which uses a key size of 128,192 and 256 bits for 10, 12 and 14 cycles respectively (Rajput et al., 2016). Each round of AES involves permutation and combination. AES operates on 4 * 4 matrix also known as state matrix. AES splits the data into 4 blocks or array of bytes which form 4 * 4 matrices and are subjected to rounds (Pansotra and Singh, 2015; Thiyagarajan and Kamalakannan, 2014). AES involves a key expansion, an initial round and a final round. All the round of AES except the last round consists of same operations to be performed on the state matrix. These operations are as follows:

a. **Substitute Bytes:** These consist of operations that are based upon substitution box which has been designed specifically. The main motive of this operation is to prevent various types of attacks like mathematical attacks, differential and linear cryptanalysis etc (Wang and Kissel, 2015; Rachh et al., 2012).
b. **Shift Rows:** This is a basic linear operation which is performed on state matrices. The operation is performed with an objective of producing diffusion (Wang and Kissel, 2015; Rachh et al., 2012).
c. **Mix Columns:** This is also a basic operation which is similar to shift row operation. It involves matrix multiplication (Wang and Kissel, 2015; Rachh et al., 2012).
d. **Add round Key:** This is an elementary operation that performs the exclusive-or operation between the key of that round and the state matrix. The result is used in the next round. The objective of this operation is to create confusion (Wang and Kissel, 2015; Rachh et al., 2012).

The final round consists of all the operations of initial rounds excluding Mix Columns.

3.3. BlowFish

Blowfish was developed by Bruce Schneier in 1993 (Meyers and Desoky, 2008). It is a symmetric key algorithm similar to DES but has a varying key size of 32 to 448 bits. It is 64 bit block cipher. Blowfish algorithm works in 2 parts: Key expansion and data Encryption (Meyers and Desoky, 2008). Key expansion step involves breaking up the key into several sub keys and it takes place prior to data encryption process. The data encryption process involves a fiestel function which is repeated 16 times i.e algorithm works in 16 rounds. In each round a key dependent permutation and a key-and-data dependent substitution is applied (Schneier, 1996; Khatri and Kshirsagar, 2014; Alabaichi et al., 2013; Cornwell and Columbus, 2012). The fiestel function splits the 32-bit input into four bytes and these four bytes are used as indices in S-array. The results obtained from the lookups are then added and after that, their Xor operation is performed which yields the output.

4. Proposed system model

The proposed model (shown in Fig. 2) provides security to the data uploaded by the data owners onto the cloud. In this model the data owner uploads file using the interface of the proposed security framework. Besides uploading the file, it also provides the list of all the users who have been granted access to that file. The security framework on receiving the file encrypts it using the newly proposed symmetric key encryption technique (BDNA). The framework asks the data owner for the secret key and uses this key for encrypting the data. After encryption, the file is uploaded onto the cloud for storage. Further, the framework encrypts this secret key with the public key of all the authorized users provided by the data owners. The secret keys are encrypted using identity based cryptography i.e using public key (identity) of the authorized users. The key management takes place at the security framework module. Whenever a user needs to access the file, he/she enters the authentication credentials at framework's interface. The security framework retrieves the corresponding file from the cloud and the user is then prompted to enter his private key using which the secret key is decrypted. Using this secret key, the file is decrypted by DNA decryption technique and is sent to the user. The whole system model has been divided into three levels which have been discussed below.

4.1. End user level

This level consists of data owner and the users. The data owner uploads the files to be encrypted and provides the list of all the authorized users to the security framework. The users prove their authentication at the security framework interface and get access to the files for which they have been authorized.

4.2. Security level

The security level consists of security framework that has been designed as a web application. The overall workflow of the framework is described as follows:

a. **File Uploading:** The data owner browses the file from his/her local machine and uploads it using the security framework interface and stores this file in its own local database.

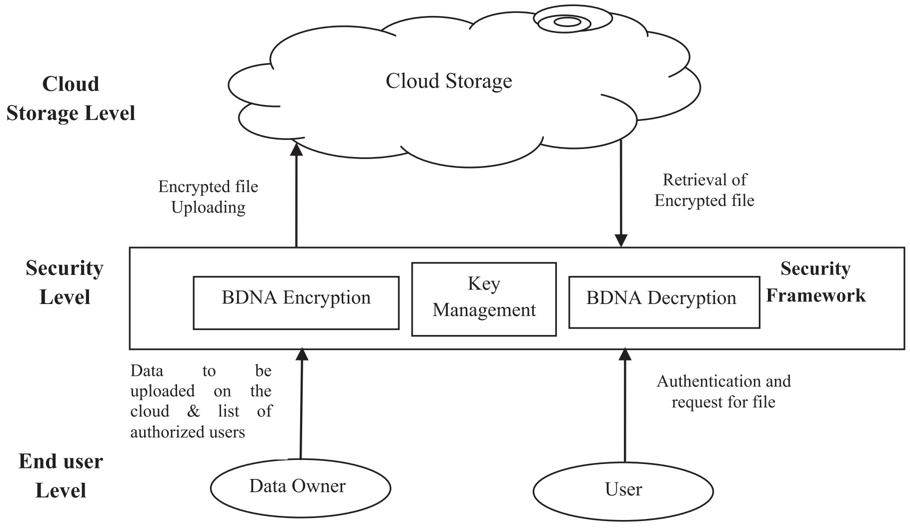


Fig. 2. The Overall System Model.

b. **File Encryption:** Once the file gets uploaded, it is encrypted using the proposed BDNA encryption algorithm (shown in Fig. 3.).
c. **Encrypted File Uploading:** It is the process of sending the encrypted file onto the cloud.
d. **File Downloading:** It is the process of retrieving file from the cloud storage. The process starts with submitting the name of the file to the cloud.
e. **File Decryption:** It is the process of decrypting the file using BDNA decryption algorithm. The file obtained from the cloud is decrypted here. This step first involves the decryption of secret key by identity based decryption scheme using private key of the authorized users and then this secret key is further used to decrypt the actual file.
f. **Key Management:** In this model the key management is done by the security framework. The security framework securely stores the secret keys using its local database as done in (Marwan et al., 2016, 2017, 2018). The framework manages these keys according to the access rights of the different users.

4.3. Cloud storage level

This is the topmost level of our proposed security model. It consists of the storage cloud where the data of the users are stored in encrypted format. The security framework uploads the encrypted files to this cloud through the cloud interface. When a user requests a file from security framework, it sends the request to the cloud and then, the encrypted file is downloaded from the cloud which is decrypted by the security framework and sent to the user who made the request.

This is the overall system model for securing data stored on the cloud and this model can also be extended to the multi-cloud environment. But the major focus of this paper is to discuss about a component of this model which is a novel symmetric key cryptographic technique which has been used to encrypt/decrypt the data before uploading it onto the cloud. Therefore, the rest of this paper discusses about BDNA cryptographic algorithm and its enhancements over the existing symmetric key algorithms.

5. Proposed BDNA symmetric key algorithm

BDNA is a symmetric key algorithm i.e. the same key is used for encryption as well as for decryption. The basic idea of this technique is based upon the DNA cryptographic technique presented by Hossain et al. (2017). Sohal and Sharma (2017) have already discussed about the shortcomings of this DNA technique (Hossain et al., 2017) and have presented the improvements over this technique. According to the proposed scheme, if a data owner wants to upload his/her data onto the cloud, first of all, he has to encrypt the data. This means that the client-side encryption has been adopted in this approach. For this purpose, the data owner requires the two encoding tables (Table 1 and Table 2), a 14-bitencryption key, and a random number N. This encryption process is highly dynamic in

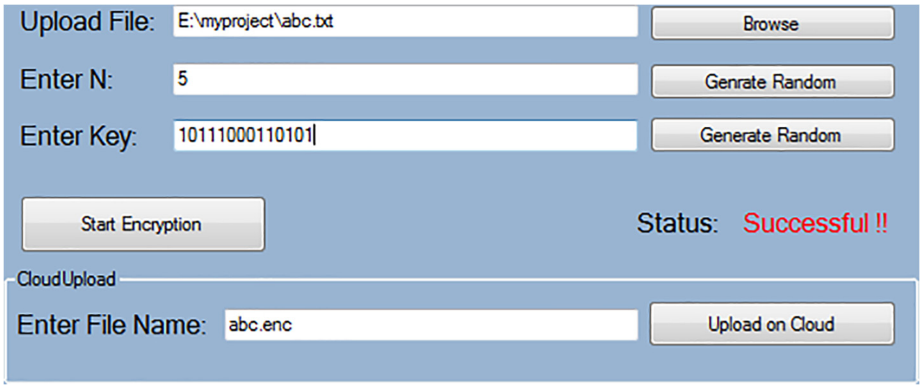

Upload File: E:\myproject\abc.txt
Enter N: 5
Enter Key: 10111000110101
Status: Successful !!
CloudUpload
Enter File Name: abc.enc

Fig. 3. The security framework interface for encryption.

Table 1
Initial encoding of ASCII characters as 7-bit combinations for n = 0.

0000000-y	0000001-\	0000010-l	0000011-,	0000100-;	0000101
0000110- W	0000111-a	0001000-J	0001001-$	0001010-‘	0001011
0001100-{	0001101-g	0001110-7	0001111-r	0010000-w	0010001
0010010-b	0010011-E	0010100-C	0010101-P	0010110-U	0010111
0011000-z	0011001-2	0011010-m	0011011-.	0011100-“	0011101
0011110-X	0011111-+	0100000-K	0100001-#	0100010-space	0100011
0100100-[	0100101-h	0100110-8	0100111-&	0101000-x	0101001
0101010-c	0101011-F	0101100-*	0101101-s	0101110-V	0101111
0110000-A	0110001-3	0110010-n	0110011-Q	0110100-,	0110101
0110110-Y	0110111-=	0111000-L	0111001-?	0111010-‘	0111011
0111100-}	0111101-i	0111110-9	0111111-@	1000000	1000001
1000010-d	1000011-G	1000100-o	1000101-t	1000110	1000111
1001000-B	1001001-4	1001010-M	1001011-R	1001100	1001101
1001110-Z	1001111-_	1010000-<	1010001-/	1010010	1010011
1010100-]	1010101-j	1010110-^	1010111-!	1011000	1011001
1011010-e	1011011-H	1011100-p	1011101-u	1011110	1011111
1100000-(	1100001-5	1100010-N	1100011-S	1100100	1100101
1100110-0	1100111–	1101000->	1101001-:	1101010	1101011
1101100-|	1101101-k	1101110-%	1101111-~	1110000	1110001
1110010-f	1110011-I	1110100-q	1110101-v	1110110	1110111
1111000-D	1111001-6	1111010-O	1111011-T	1111100	1111101
1111110-1	1111111-)				

Table 2
Final Encoding Table.

000000-F	000010-O	000100-c	000110-+
001000-g	001010-t	001100-W	001110-N
010000-Z	010010-X	010100-y	010110-v
011000-I	011010-3	011100-6	011110-R
100000-r	100010-V	100100-G	100110-8
101000-l	101010-7	101100-0	101110-B
110000-C	110010-J	110100-u	110110-4
111000-j	111010-1	111100-z	111110-x
000001-S	000011-a	000101-P	000111-U
001001-b	001011-i	001101-k	001111-2
010001-L	010011-M	010101-5	010111-/
011001-w	011011-s	011101-T	011111-H
100001-Q	100011-Y	100101-e	100111-p
101001-o	101011-h	101101-q	101111-E
110001-A	110011-D	110101-K	110111-f
111001-m	111011-n	111101-9	111111-d

nature as in this the first encoding table i.e. Table 1 changes every time, the data is encrypted. The entries inside Table 1 are shifted according to the value of n. So, even if the intruders have access to the two encoding tables, they still cannot decrypt the data as the values of Table 1 keeps on changing, thus making it highly unbreakable. After shifting the values in Table 1, the data owner converts the plaintext into binary sequences using Table 1 and then applies transformation using the encryption key. Then, as a final step, the binary data is converted into the final ciphertext using Table 2. On the client side, when a user wants to access the data stored onto the cloud, first of all, he/she has to prove his/her authentication and then, he/she gains access to the cloud data, decryption key and a random number N. Depending upon the value of N, the values in Table 1 are shifted and then decryption process is applied using the decryption key.

In this work, table 1 represents all the 96 ASCII characters as 7-bit binary sequences. The ASCII characters are allocated to the 7 bit sequences on entirely random basis irrespective of their binary value. Since in all other encryption algorithms, each character is represented as 8-bit binary number, it would be hard for the intruder to speculate that the characters have been represented by 7 bits and not 8. Table 2 is the random version of base64 encoding. Here, each 6-bit sequence is allocated to a character from base64 encoding. But, the allocation of characters is purely on the random basis.

Our approach initially, uses a 14-bit, but before actually encrypting the data, a 7-bit key has to be derived from it depending upon the first bit. This further enhances the security of the presented technique, thus making it a multifold cryptographic technique. The encryption and decryption steps discussed in this section are performed by the security framework level of the proposed system model.

5.1. The encryption process

The encryption process involved in this approach is a client-side encryption i.e the data owner encrypts the data before uploading it to the cloud. The cloud provider has no access to the original data. Even if the malicious user gains access to the cloud data he would not be able to access the original unencrypted data. The encryption process involves several steps. First of all, a random number N is input. Depending upon the value of N, the characters in Table1 are shuffled. Then, the plaintext is converted into bit sequence using these shuffled values of Table 1. After this, a random 14-bit key is generated. If the initial bit of the key is 1, all the bits at odd locations are extracted and if the first bit is 0, all the even positioned bits are extracted. The resulting 7-bit key is the actual key involved in the encryption process. Then the XOR operation is performed between the plaintext bit sequence and the 7-bit key. The resultant bit sequence is then divided into blocks of 7 bits and then substitution is performed. The position of each block is checked in Table 1 and subsequently, each block is replaced by the bit sequence present at (127-i)th position in Table 1. In the next step, transposition is applied. The bit sequence is divided into the two halves, the left shift is applied on the first half and the right shift on the second half. Afterwards, the two halves are interchanged. As the final step, the resulting bit sequence is converted into ciphertext characters using Table2.

The Algorithm 1 below presents the stepwise explanation of the encryption process.

Algorithm 1: Encryption Process

```
1. Begin:
2. Read the plain text;
3. Input a random number N;
4. While (N! = 0) do
    a. Shuffle the value of Table 1 according to the value of n.
    b. Decrement N;
5. End while
6. Convert plaintext into 7-bit sequence using Table 1.
7. Input the 14 bit key
    a. Extract 7 bit key depending upon the first bit
8. Perform XOR between the key and 7-bit sequence
9. Divide the resulting bit sequence into blocks of 7 bits
10. Now find out the position i of each block from Table 1
11. Substitute each block with the block located at (127-i)th
 position in Table 1
12. Divide the resulting bit sequence into two halves
13. Left shift the first half and right shift the second half
14. Interchange the two halves
15. Divide the resulting bit sequence into blocks of 6 bits
16. if length of the resulting bit sequence is not divisible by
 6, pad zero's at the end of last block
17. Generate ciphertext using Table 2.
18. End.
```

5.2. The decryption process

The decryption process is carried out at the user's end. First of all, the user authentication takes place at the security framework. Once the identity of the user is proved, he provides the name of the file that he needs to access. The data obtained from the cloud is in the encrypted format which needs to be decrypted. For decryption, the user gets the decryption key and value of N for the data he is authorized to access. The decryption process is the exact reverse of encryption. First of all the ciphertext is read. Then, the 14-bit key is read and converted into 7-bit key depending upon the initial bit of the key; same as done during encryption (see Section 4.1). The characters at Table 1 are then shuffled according to the value of N. Then, the ciphertext is converted into bit sequence using Table 2. Then, the ciphertext is converted into bit sequence using Table 2. The resulting ciphertext is then split up into two halves. The right shift is applied on the first half and left shift on the other half. The two halves are then interchanged. The resulting bit sequence is then chunked as 7-bit blocks. Each block is then replaced with the (127-i)th block from Table 1. Then, the XOR operation is performed between the resulting bit sequence and the 7-bit key. As the final step, the bit sequence is converted into plaintext using Table 1.

The Algorithm 2 below provides the stepwise explanation of the decryption process.

```
Algorithm 2: Decryption Process
1. Begin:
2. Read the cipher text;
3. Read the 14 bit key
    a. Extract 7 bit key depending upon the first bit
4. Read the given number N
5. While (N!=0) do
    a. Shuffle the value of Table 1 according to the value of n.
    b. Decrement n.
6. End while
7. Convert the cipher text into bit sequence using Table 2.
8. Divide the resulting bit sequence into two halves
9. Right shift the first half and left shift the second half
10. Interchange the two halves
11. Divide the resulting bit sequence into blocks of 7 bits
 and remove the extra bits
12. Now find out the position i of each block from Table 1
13. Substitute each block with the block located at (127-i)th
 position in Table 1
14. Perform XOR between the key and 7-bit sequence
15. Generate plain text using Table 1
16. End
```

5.3. Implementation

The proposed algorithm has been implemented in java on core i5 processor 3210 M @ 2.67 Ghz, 4 GB RAM, 64-bit operating system. The program acknowledges the plaintext as an input. After a thriving implementation, the data has been produced in encrypted form (shown in Fig. 4). Further, the size of cipher text generated in bytes for the size of plaintext input by the user has been calculated. The total time taken for encrypting the given plaintext has also been recorded for the various techniques.

For example:

Plaintext:

“Good, better, best. Never let it rest. Till your good is better and your better is best.”

Value of n = 2

Key: 10101010101011

Ciphertext: hYZRG26+YZIHK8Yj4wy7DyQ1O7+UhYZRG2YweXhkXU9wlxk887rGaLlk/UA0Jo58o5Azskk5SLm/kpDyQmYW7U95Zxe887rlddKW+Sj

5.4. Enhancements over the existing DNA cryptography

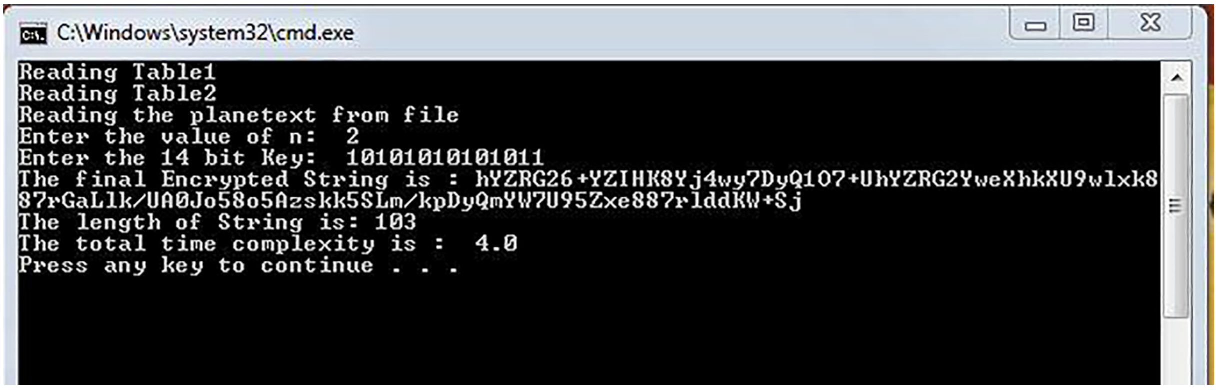
In DNA cryptography, all the characters are represented in terms of the four DNA bases A, T, C, G. Each base in turn is represented by two bits for example A as 00, T as 01, C as C as 10 and G as 11. So, this in whole makes each character of 8 bits which is the conventional representation of a character. But, in our approach each character is represented in terms of 7 bits which results in reduced size of ciphertext as compared to DNA Cryptography. Moreover, in DNA cryptography the first encoding table can only represent 96 characters there is no provision for adding some additional characters in future. In our proposed approach, the first encoding table (see Table 1), can be expanded to accommodate 32 additional characters. In most of the DNA cryptographic algorithms, a second encoding table is used which converts the DNA sequence into amino acids by grouping together three DNA bases. This table uses characters from A to Z to represent various amino acids. Each character of this table represents multiple amino acids. The final ciphertext in these cases is the sequence of amino acids. This creates problem at the time of decryption, as it is not clear which amino acid is to be replaced with the given character of the ciphertext. This ambiguity has been resolved in the second encoding table (see Table 2) used in our approach. Moreover, in DNA cryptography the amino acids are represented by only characters from A-Z, therefore the ciphertext would consists of these 26 upper case characters. Therefore, it becomes suspicious that the data has been encrypted using some DNA cryptographic method. All these shortcomings have been removed in our approach.

5.5. Security Analysis:

The proposed algorithm is secure against chosen plaintext attacks (CPA). A symmetric algorithm is said to be secure against CPA if for all the probabilistic polynomial-time, the adversaries A have negligible chances of guessing the right plaintext.

Let us consider an experiment $SymK^{CPA}_{A,\varepsilon}$ consisting of following phases:

- Training Phase: The Adversary *A* is given access to the BDNA Encryption system in this step. Adapting to the situation, *A* submits its query messages and obtains their encryptions.
- Challenge Phase: *A* submits two equal length challenge plaintexts, p_0 and p_1 to the Challenger. Here *A* is free to query any message of its preference. The Challenger acquires the encryption key, k, from the key generator and then picks random number N according to which it shuffles the values of Table 1. He then randomly picks up the value of bit b ϵ {0.1}, and encrypts the resultant challenge plaintext, say p_b, and sends the ciphertext to *A*.



```
Reading Table1
Reading Table2
Reading the planetext from file
Enter the value of n:  2
Enter the 14 bit Key:  10101010101011
The final Encrypted String is : hYZRG26+YZIHK8Yj4wy7DyQ1O7+UhYZRG2YweXhkXU9wlxk8
87rGaLlk/UA0Jo58o5Azskk5SLm/kpDyQmYW7U95Zxe887rlddKW+Sj
The length of String is: 103
The total time complexity is :  4.0
Press any key to continue . . .
```

Fig. 4. Experimental Results.

- Post-Challenge Training Phase: *A* can utilize the Encryption System even after the challenge. Now also *A* can adaptively submit its query messages and obtains their ciphertexts.
- Response Phase: *A* finally submits its guess regarding encrypted plaintext, in the form of a bit, b'. A wins the experiment if it guesses the correct value of b'.

The symmetric key algorithm ε= {KGen, Enc, Dec}is said to be secure against CPA if for all probabilistic polynomial time Adversaries there is negligible function negf(n) such that

$$\Pr\left[SymK_{A,\varepsilon}^{CPA} = 1\right] \leq \frac{1}{2} + negf(n) \tag{1}$$

If an encryption scheme is CPA-secure, then calling the Encryption algorithm two times with the same plaintext and key must lead to different ciphertexts. Since, in our proposed scheme the value of N is selected randomly on each encryption, the entries inside encoding Table 1 are shuffled for each plaintext, even if the adversary submits the same challenge plaintext twice, the resulting ciphertext would be different. Therefore, it can be concluded that our proposed scheme is CPA-secure.

Let us consider the scenario that *A* submits two challenge plaintexts p_0 and p_1 such that $p_0 = p_1$. According to our proposed scheme,

$$\varepsilon(k, N, p0) \neq \varepsilon(k, N, p1) \tag{2}$$

This is because the value of N changes for every encryption making Table 1 dynamic in nature. *A* has negligible chances of guessing the right plaintext. Hence, we conclude that our scheme is secure against Chosen Plaintext Attacks.

5.5.1. Security of BDNA vs. DNA

While the data is being transmitted using DNA cryptography, the ciphertext itself reveals that the encryption algorithm used is DNA as the data would be transmitted either in the form of ATCG or if it has been converted into amino acids, in that case the ciphertext would be represented only in terms of characters from A-Z. While in case of BDNA, the characters are being represented using an enhanced version of base64 encoding which is being used in most of the encryption algorithms, so the ciphertext does not reveal the type of encryption algorithm used. Secondly, since we are using an enhanced version of base64 encoding, therefore it is difficult to surpass this level of encryption. But, even if the intruder somehow decodes the base64 encoding, in the next step the obvious thing is to work on 8 bits a time as the conventional representation of characters is 8 bits. But, in BDNA we use 7 bits to represent the character which further adds to the security of our proposed algorithm. While in DNA cryptography, once the amino acids have been decoded, the next step simply represents each character in terms of 8 bits which is the obvious representation of the characters.

6. Experimental results:

In this section, the comparison of the proposed technique has been made with the existing symmetric key techniques such as DNA, AES, DES, and Blowfish on the basis of the size of ciphertext produced for different plaintext sizes, time required for encrypting the given size of plaintext and the throughput of each algorithm. The observations have been recorded in Tables 3–5.

6.1. Ciphertext size

Since, this technique has been designed for storing data in the cloud; therefore cloud storage has been taken into account while designing this approach, so that after encryption the size of data does not increase too much. Furthermore, the transmission time is directly proportional to the amount of data sent. Since, in client-side encryption, the data is encrypted before uploading it onto the cloud, therefore, the size of data uploaded directly effects the transmission time. We have evaluated our approach and other approaches and the results show that our approach produces smaller sized ciphertext as compared to the existing encryption techniques. The results have been shown in Table 3 and Fig. 5 below:

Table 5
Throughput of various techniques.

Algorithm	Throughput (kb/sec)
BDNA	180.4
Blowfish	159.6
AES	126.8
DES	33.32
DNA	20.92

Table 3
Comparison of Ciphertext Size among various techniques for a given Plaintext size.

Plaintext Size in KB	Ciphertext Size in KB				
	BlowFish	AES	DES	DNA	BDNA
5	6.76	6.86	6.86	6.67	5.83
10	13.52	13.70	13.70	13.33	11.66
15	20.27	20.55	20.55	20.0	17.5
20	27.03	27.39	27.39	26.67	23.34
25	33.78	34.25	34.25	33.33	29.17
30	40.5	41.09	41.09	40.0	35.0

Table 4
Comparison of encryption time among various techniques for a given plaintext size.

Plaintext Size in KB	Encryption Time in seconds				
	BlowFish	AES	DES	DNA	BDNA
5	0.054	0.053	0.152	0.200	0.048
10	0.074	0.081	0.283	0.483	0.066
15	0.097	0.106	0.405	0.625	0.090
20	0.130	0.172	0.571	0.972	0.106
25	0.133	0.198	0.785	1.250	0.123
30	0.170	0.218	0.955	1.487	0.149

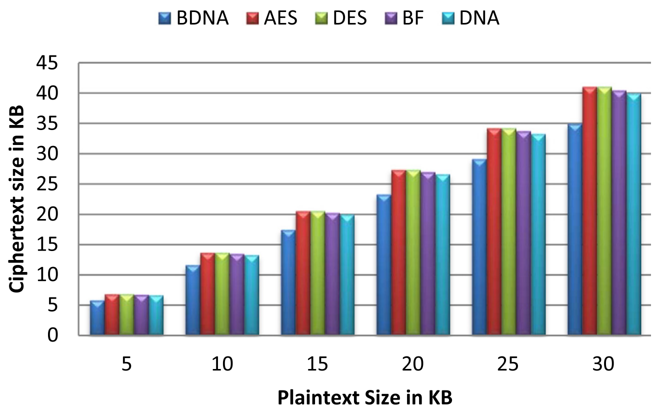


Fig. 5. Analysis of cipher text size generated by BDNA and other symmetric algorithms.

6.2. Encryption time

Encryption time can be defined as the time involved in converting plaintext into ciphertext. The efficiency of an encryption algorithm is inversely proportional to the encryption time. Lesser the encryption time of an algorithm, higher would be its efficiency. In this section, we have compared the encryption time of BDNA with the other symmetric algorithm. The results demonstrate that

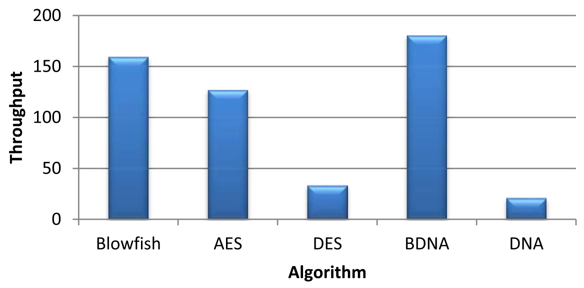


Fig. 7. Comparison of throughput of various symmetric algorithms with BDNA.

as compared to AES, DES, and Blowfish, BDNA takes less time for encrypting the same plaintext. Hence, BDNA is more efficient than these algorithms. The results have been illustrated below in Table 4 and Fig. 6.

6.3. Throughput

The Performance of an algorithm can be analyzed directly through its throughput. The Performance of an algorithm is directly proportional to the throughput i.e. greater the throughput of the algorithm, higher will be the performance. The formula for calculating the throughput of an encryption algorithm is as follows:

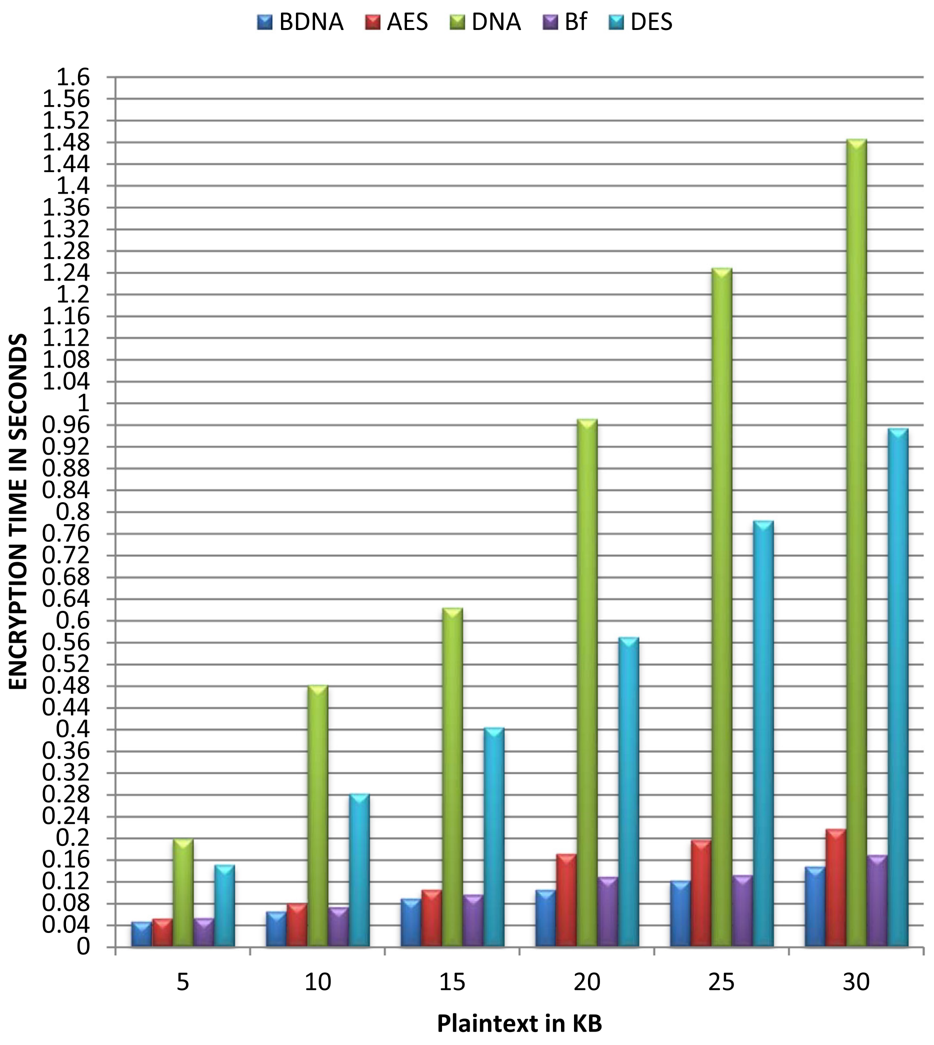


Fig. 6. Comparison of Encryption time taken by BDNA and other symmetric encryption techniques.

Throughput = Plaintext size/Encryption Time

Based upon the recorded encryption time in Table 4, the throughput for all the techniques has been calculated using the above formula. Table 5 and Fig. 7 clearly demonstrate that the throughput of BDNA is higher than all other techniques. Hence, it can be concluded that BDNA performs better than other symmetric key algorithms.

7. Conclusions

In cloud computing, distributed resources are shared among users in an open environment by means of networks. Thus, the data is accessible to the users from anywhere. In spite of various benefits of cloud storage, there are still lots of hindrances regarding security and privacy of the data that need to be resolved. Although, the cloud facilitates flexible and easy to access data storage and management, there are still possibilities of unauthorized attacks and malicious activities. The cloud server may store confidential and sensitive data. Therefore, the security of the data is of prime concern. Cryptography techniques proffer a secure means for private data storage at the third party (cloud) by means of encryption and grant its corresponding key to the authorized user only. In this paper, a novel symmetric key cryptographic technique has been proposed which is inspired by DNA cryptography. Our proposed scheme uses dynamic encoding tables that are random in nature which leads to higher security. The security analysis proves that the proposed scheme is CPA-secure. The experimental results show that our proposed approach outperforms other symmetric key encryption algorithms (DNA, AES, DES, Blowfish) in terms of ciphertext size, encryption time and throughput.

References

Liu, X., Deng, R., Choo, K.K.R., Yang, Y., Pang, H., 2018. Privacy-preserving outsourced calculation toolkit in the cloud. IEEE Trans. Dependable Secure Comput., 1–14

Yang, Y., Liu, X., Zheng, X., Rong, C., Guo, W., 2018. Efficient traceable authorization search system for secure cloud storage. IEEE Trans. Cloud Comput., 1–14

Paladi, N., Gehrmann, C., Michalas, A., 2017. Providing user security guarantees in public infrastructure clouds. IEEE Trans. Cloud Comput. 5, 405–419.

Michalas, A., 2016. In: In: Proceedings of 11th IEEE International Conference on Internet Technology and Secured Transactions (ICITST), pp. 182–187.

Kaaniche, N., Laurent, M., 2017. Data security and privacy preservation in cloud storage environments based on cryptographic mechanisms. Comput. Commun. 111, 120–141.

Nie, T., Song, C., Zhi, X., 2010. In: In: Proceedings of the 2010 IEEE International Conference on Biomedical Engineering and Computer Science (ICBECS), pp. 1–4.

Rouse M. Cloud Encryption (Cloud Storage Encryption). http://searchcloudstorage.techtarget.com/definition/cloud-storage-encryption (2014). (Accessed June 23, 2017).

Kamara, S., Papamanthou, C., Roeder, T., 2012. In: In: Proceedings of the 2012 ACM conference on Computer and communications security, pp. 965–976.

Dowsley, R., Michalas, A., Nagel, M., Paladi, N., 2017. A survey on design and implementation of protected searchable data in the cloud. Comput. Sci. Rev. 26, 17–30.

Sohal, M., Sharma, S., 2017. Enhancement of cloud security using DNA inspired multifold cryptographic technique. Int. J. Secur. Its Appl. 11 (12), 15–26.

Sharma, M., Garg, R.B., 2016. DES: The oldest symmetric block key encryption algorithm. Proceedings of the IEEE International Conference on System Modeling & Advancement in Research Trends (SMART), 53–58.

Meyers, R.K., Desoky, A.H., 2008. In: In: Proceedings of the 2008 IEEE International Symposium on Signal Processing and Information Technology (ISSPIT), pp. 346–351.

Kumar, P., Rana, S.B., 2016. Development of modified AES algorithm for data security.Optik-Int. J. Light Electron Opt. 127 (4), 2341–2345.

Rajput, S., Dhobi, J.S., Gadhavi, L.J., 2016. In: Enhancing Data Security Using AES Encryption Algorithm in Cloud Computing. Technology for Intelligent Systems, Springer, pp. 135–143.

Bhardwaj, A., Subrahmanyam, G.V.B., Avasthi, V., Sastry, H., 2016. Security algorithms for cloud computing. Procedia Comput. Sci. 85, 535–542.

Pansotra, A., Singh, S., 2015. Cloud security algorithms. Int. J. of Secur. and Its Appl. 9 (10), 353–360.

Furht B., Data Encryption Standard (DES) and Advanced Encryption Standard (AES). Encyclopedia of Multimedia. Springer, Boston, MA. (2006). doi: 10.1007/0-387-30038-4_46.

Biryukov A., Cannière D.C., 2011. Data Encryption Standard (DES). In: van Tilborg H.C.A., Jajodia S. (Eds.) Encyclopedia of Cryptography and Security. Springer, Boston, MA. doi: 10.1007/978-1-4419-5906-5_568.

Thiyagarajan B., Kamalakannan R., 2014. Data integrity and security in cloud environment using AES algorithm. In: Proceedings of 2014 IEEE International Conference on Information Communication and Embedded Systems (ICICES), pp. 1–5.

Wang, J., Kissel, Z.A., 2015. Data Encryption Algorithms. In: In: Introduction to Network Security: Theory and Practice. John Wiley & Sons Singapore Pte. Ltd, Singapore. https://doi.org/10.1002/9781119113102.ch2.

Rachh, R.R., Mohan, P.V.A., Anami, B.S., 2012. Efficient implementations for AES encryption and decryption. Circuits Syst. Signal Process 31, 1765–1785.

Bruce Schneier, John Wiley & Sons, Inc. https://mrajacse.files.wordpress.com/2012/01/applied-cryptography-2nd-ed-b-schneier.pdf. 1996. (Accessed June 23 2017).

Khatri, N., Kshirsagar, V.K., 2014. Blowfish algorithm. J. Comput. Eng. 16 (2), 80–83.

Alabaichi, A., Ahmad, F., Mahmod, R., 2013. In: In: 2013 IEEE International Conference on Informatics and Applications (ICIA), pp. 12–18.

Cornwell, J.W., Columbus, G.A., 2012. Blowfish survey. GA Columbus State University, Department of Computer Science Columbus, pp. 1–6.

Marwan, M., Kartit, A., Ouahmane, H., 2016. In: In: 2nd IEEE International Conference on Cloud Computing Technologies and Applications (CloudTech), pp. 88–94.

Marwan, M., Kartit, A., Ouahmane, H., 2017. Design a Secure Framework for Cloud-Based Medical Image Storage. Applications.

Marwan, M., Kartit, A., Ouahmane, H., 2018. A framework to secure medical image storage in cloud computing environment. J. Electronic Commerce Organizations 16 (1), 1–16.

Hossain, E.M.S., Alam, K.M.R., Biswas, M.R., Morimoto, Y., 2017. In: In: 2017 IEEE International Conference on Computer and Information Technology (ICCIT), pp. 270–275.