ISSN:0975-9646

An Approach to Improve the Security of Online System using Crypto System

Saurav Mallik, Sutapa Majee,Arun Kanti Manna,Md. Headayetullah
Department of Computer Science and Engineering, Dr. B. C. Roy Engineering College, Durgapur, India

Abstract—**Presently, the chief problem facing Internet banking is the security of online transactions. That's why, 3-tier security is used to make the transaction more secure. Among the 3-Tier Security layers, Application layer security comes first as it is the topmost layer in OSI model. The visitors always assume about the security and safety of these transactions, and so they are steadily lose their trust on the bank due to increase of various online Internet attacks. Now, it is trying to design an approach to enhance the Application Layer Security in case of online security application. It helps to make a latest idea about online banking transactions as well as increases trust and security over the existing theory, by challenge-response mutual authentication process. This paper says about possible preventions of Password Guessing by *padding a random string in the different position* of the password depending upon *UTC (Coordinated Universal Time) time and Nearest Prime Number Concept.* We also propose an algorithm on *Multiple Key Block Cipher Symmetric Encryption (MKBCSE) algorithm* which is advantageous in many respect than other existing symmetric key encryption technique. Different *Key generation* and *NOB variable* have major important role here, because the key here changes after a certain number of bits (NOB). Key comprises of various components and is a combination of various server and client related information. This makes it hard for the attacker to guess the key.**

Keyword:-Random string Padding, Multiple key symmetric encryption with block cipher, NOB.

I. INTRODUCTION

In today's world of increasingly sophisticated cyber attacks, application-layer security threats are top of mind with many network administrators, security consultants. The loss of network and application access can cost enterprises dearly in lost revenue and employee productivity. Today's security infrastructure must address the new wave of application-layer security attacks and application abuse.

Presently, the chief problem facing Online banking is the security of online transactions. That's why, 3-tier security[1,4,5] is used to make the transaction more secure. Among the 3-Tier Security layers, Application layer security comes first as it is the topmost layer in OSI model. The visitors always assume about the security and safety of these transactions, and so they are steadily lose their trust on the bank due to increase of various online Internet attacks. There are mainly two types of attacks [3]: local attacks (example: Trojan-horse), remote attacks (example: password guessing, dictionary search, phishing, pharming etc.). Now, it is trying to design an approach to enhance the Application Layer Security in case of online security application. It helps to make a latest idea about online banking transactions as well as increases trust and security over the existing theory, by challenge-response mutual authentication process.

In Internet Banking[1,2], a hash-encrypted password is not entirely secure. An attacker can hack hashed password when transferring it from client to server. So, in this paper, as an extra protection, we ***pad a random string (***given by user) ***in different position of hashed string***. This padding position is calculated depending on ***UTC time and Nearest prime number***. So, it is difficult guess the password. We also propose an algorithm of ***Multiple Key Block Cipher Symmetric Encryption (MKBCSE),*** where, after a particular bit number (NOB), different keys are generated used for encryption and decryption. So, here key is more secure than other existing encryption technique. Key is made of server[2,3] and client related information. This makes it hard for the attacker to guess the key. The rest of the paper is well thought out as follows: Section II describes the previous research review. Section III depicts proposed possible preventions of password guessing. Section IV discusses experimental results and benefits of this proposed work. Finally conclusions are summed up in Section V.

II. LITERATURE REVIEW

A. Existing salt value padding technique:-

In cryptography[4], a **salt** consists of random bits[5] that are used as one of the inputs to a one-way function. The other input is usually a password or passphrase. The output of the one-way function can be stored rather than the password, and still be used to authenticate users. A salt can also be combined with a password by a key derivation function to generate a key for use with a cipher or other cryptographic algorithm.

In a one-way function, salt value is always padded at the end of the password. That means position of padding is fixed. And also size of salt value is also fixed. So, it is possible to guess the salt size and can get the original password.

B. Limitations of existing Symmetric key algorithms:-

The existing symmetric key encryption algorithms(like DES, 3DES, AES, Blowfish) has some shortcomings.

A brief note on the operational mechanism and overheads posed by some popular symmetric key algorithms are discussed below:-

B.1 **DES/3DES**:

DES[3] applies a symmetric 56-bit key to each 64-bit block of data. The process can run in several modes and involves 16 rounds of operations. DES is breakable as the key size is too less. Hence Triple DES (3DES)[7] an Enhancement of DES emerged as a stronger method. Triple DES encrypts the data three times and uses a different key for at least one of the three passes for giving a cumulative key of size 112-168 bits. If we consider a triple length key to consist of three 56-bit keys K1, K2, K3 then encryption is as following order :- encryption with K1, decryption with K2, encryption with K3. Whereas, decryption is the reverse process of encryption, so the decryption of the cipher text will be as follows:- decryption with K3, encryption with K2, decryption with K1. The computational overhead at the sender and receiver ends, are as high as 3DES that involves three times the operations of normal DES, with each DES contributing 16 rounds of operation [4]. Further the increasing key size increases the process run time complexity. Moreover the keys are of fixed size and do not change with sessions, they has to be changed periodically between the parties for achieving more message confidentiality.

B.2 **AES:**

An Advanced Encryption Standard (AES)[6] has basically three different configurations with respect to the number of rounds and key sizes. It has 10 rounds for 128-bit keys, 12 rounds for 192-bit keys, and 14 rounds for 256-bit keys. By 2006, the best known attacks were on 7 rounds for 128-bit keys, 8 rounds for 192-bit keys, and 9 rounds for 256-bit keys [6]. It is worth mentioning here that any increase in the number of iterations or rounds and key size is a burden for both the sender and the receiver. Even the key is fixed and it doesn't change with the respective session. So, using a single key for long time decreases the message confidentiality.

B.3 **Blowfish Encryption:-**

Blowfish encrypts data in 8-byte blocks. The algorithm consists of two parts: a key-expansion part and a data-encryption part. Key expansion converts a variable-length key of at most 56 bytes (448 bits) into several sub key arrays totaling 4168 bytes. Blowfish has 16 rounds. Each round consists of a key dependent permutation, and a key and data-dependent substitution. All operations are XORs and additions on 32-bit words [4]. Blowfish Encryption is modified version of DES with simplified operations like XORs and additions. The run time complexity of Blowfish is much lesser than that of DES and AES, using the same concept, but suffers the same problem as DES. Session key uses multiple keys for ciphering and deciphering the message for a period of time. In session key mechanism after a particular time t, both the sender and receiver will change the keys which are being used to cipher and decipher. The problem with this technique is that if there is lack of synchronization between the sender and receiver, one end might have changed the key whereas other might be using the previous key. But instead of using a time stamp to change a key, we consider here the change of key after encrypting some number of bits (i.e. some piece of message) and similarly on receiver side it changes the key after decrypting same number of bits. We emphasize here that this novel idea is more suitable for this process of encryption or decryption which has been justified through experimental results.

III. PROPOSED POSSIBLE PREVENTIONS OF PASSWORD GUESSING

There are three possible ideas about preventions of password guessing which are given below:-

- Padding salt value,
- Padding a random string in different positions of hashed message string,
- Different Keys generation.

A. *Proposed Conceptual Algorithm of Padding salt value and Random String:-*

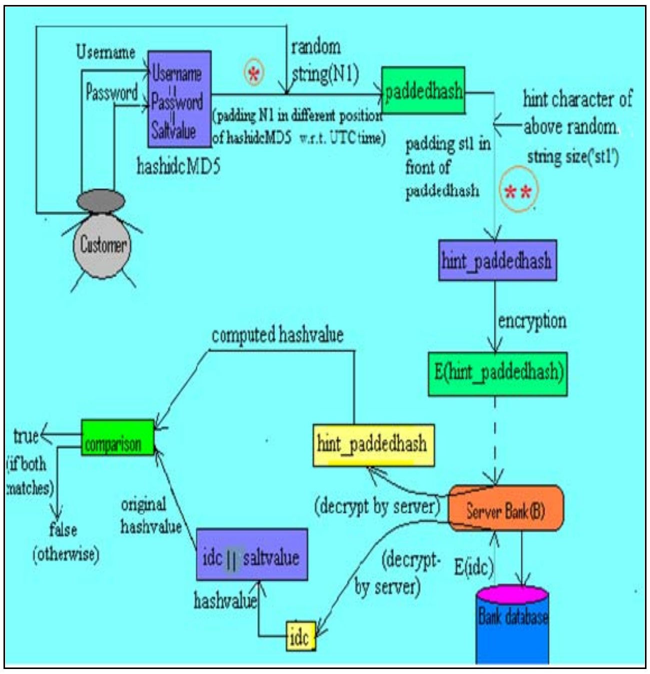


Figure 1:**Challenged-based Password Verification Technique**

Now, our first target is to design an algorithm on challenged based password verification technique(in Figure 1), where we are trying to pad a random string which will be padded in different position of hashed string(from the Figure 1). This padding position totally depends on UTC time and Nearest Prime number concept.

- ***Algorithm part '*'(from Figure1):-***

<u>Step1:</u>- Taking a random string (N1) from user with a variable length. Find the size of that string as 'size(N1)';

<u>Step2:</u>-Find UTC time (with hour ,minute, second) and UTC date (with date, month, year). Then add some extra time value with UTC time. After that calculate a time value(T1) and a date value(D1), where

$$\mathbf{T1 = ((minute+12)*60+(second+14))>>3}$$
$$\mathbf{D1=((month+6)*30+(day+9));}$$

Then find ***nearest prime number of (T1)*** referred as **'PrimeT1'** and also find ***nearest prime number of (D1)*** referred as **'PrimeD1'.**

<u>Step 3</u>:- Calculate the position of padding :-

$$Pad\_position=(2^{PrimeD1} + 3^{PrimeT1})\% \ size(hashstring+1);$$

- ***Algorithm part '**' (from Figure 1):-***

Step:- Find ***(PrimeT1/size(N1));*** Then add extra value in it and find corresponding ASCII character (refer as 'st1').

B. Multiple key block cipher symmetric encryption (mkbcse) algorithm:-

On the basis of the above mentioned (in *Section II.B.*) shortcomings, of the popular crypto algorithms, we propose "Multiple Key Block Cipher Symmetric Encryption" (MKBCSE) algorithm which overcome those. Our algorithm is under symmetric key block ciphering and uses a 128 bits key. NOB(number of bits) is a variable, which will decide, after how many number of bits the key should change. We consider 128 bits key size and n bits NOB (where $7<=n<=128$). Here, $128<=NOB<=(2^n-1)$ and NOB must be multiples of 128. The key and NOB is known to both the sender and receiver via exchange of an initial message. For message integrity, we will use message authentication code (MAC) using MD5. 'bit_count' is used as a variable to count the number of bits to be encrypted and decrypted. The bit_count will be compared to NOB. If it is equal to the NOB, a new key will be generated.

B.1. Key Generation Algorithm:-

We assume that S is a server,N_i presents each node in the network. The key and NOB will be generated at the server end and then exchange between server and the client. This process consists of generating MIN_i (client node related), MIN_s (server related), $SRMP_{Ni}$ (based on screen resolution and mouse position) and T_i (a time component).Based on $SRMP_{Ni}$ and T_i , a random number of 128 bits, R_i is generated. The MIN_i and MIN_s make use of MAC address, IP address, host name of server S and node N_i respectively. So, key (Figure:2) is a combination of various server and client related information. This makes it hard for attacker to guess the key.

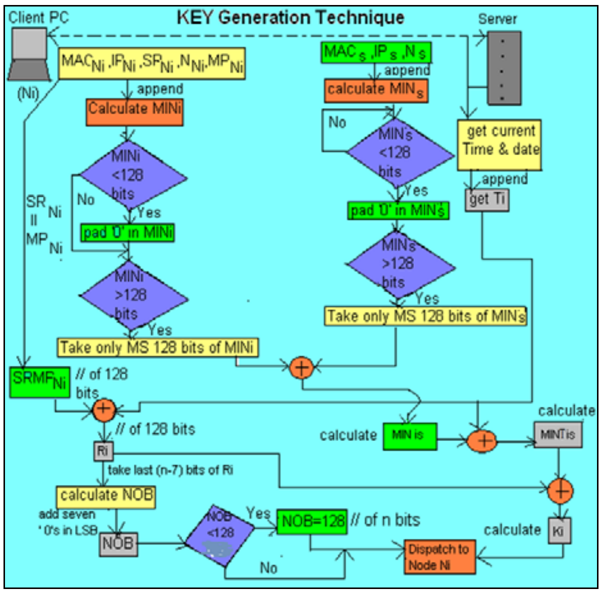


Figure 2:Key generation technique in MKBCSE

- Steps are as follows:-

```
KeyGeneration (n)
{
            /* acquiring Ni's information */
1. Acquire node Ni's MAC address ($MAC_{Ni}$), IP address ($IP_{Ni}$), Host Name ($N_{Ni}$), Screen Resolution ($SR_{Ni}$),Mouse Position ($MP_{Ni}$).
            /*generate $MIN_i$ of 128 bits*/
2. Generate $MIN_i$ variable by appending $MAC_{Ni}$, $IP_{Ni}$, $N_{Ni}$.
3. If($MIN_i$<128 bits)
       Pad '0' in the $MIN_i$;
   else goto step 3.
4.If($MIN_i$>128 bits)
       take 128 bits from MSB side of $MIN_i$.
            /* acquiring server's information */
5. Acquire Server MAC Address ($MAC_s$), IP address ($IP_s$), ServerName ($N_s$).
6.Find current Date,Time from server when connecting $N_i$
            /*generating $MIN_s$ of 128 bits*/
7. Generate $MIN_s$ variable by appending $MAC_s$, $IP_s$ , $N_s$.
8. If ($MIN_s$<128 bits)
       Pad '0' in $MIN_s$ ;
   else   goto step 8.
9.If($MIN_s$>128 bits)
       take 128 bits from MSB side of $MIN_s$.
            /*generate Ti of 128 bits*/
10. Find Ti by appending time, date components as string.
            /* generate random number Ri */
11. Get 128 bits $SRMP_{Ni}$ a variable by appending 64 bits $SR_{Ni}$ with 64 bits $MP_{Ni}$;
12.  Ri=$SRMP_{Ni}$ (XOR) $T_i$.      /* Here, Ri is 128 bits */
            /* calculating n bits NOB variable from Ri */
13.   NOB = last n-7 bits of Ri.
14. Append seven '0's in to the LSB side of NOB;
            /* Here, no. of bits in (NOB) = n bits*/
15. If (NOB<128)   NOB=128;   /*($128<=NOB<=2^n-1$)*/
            /*$MIN_{is}$ , $MINT_{is}$ are variables*/
16. $MIN_{is}$=$MIN_i$(XOR)$MIN_s$ ;
17.$MINT_{is}$=$MIN_{is}$(XOR)$T_i$.
18. $K_i$ = $MINT_{is}$ (XOR) Ri;   /* generating key Ki*/
19. Dispatch key Ki  and NOB to Ni;    /* dispatch key */
20. END
}   /*'n' is an agreed value between client and server*/
```

B.2. Encryption Algorithm:-

Encryption algorithm divides whole message into 120 bits blocks and each block is appended by 8 bits message authentication code(MAC). Now,these 128 bits blocks are encrypted by key. After every NOB bits of message, the key is changed and encrypted by new key. This procedure is repeated till end of message. The steps are as follows:

```
Encryption (Key K, NOB, PlainText)
{
1. Cipher_Text = null;
2.M=1st 120 bits of PlainText;
                  /*pad'0'at end of M, if M!=120 bits*/
3. bit_count = 0;
4. If (M == null)
       (goto step 12);
   else   (goto step 5);
5. 8 bits MAC=MD(NOB,M); /*MD=Message digest */
6. 128 bits(M+MAC)=append120 bits M with 8 bits MAC
7. 128 bits C=128 bits (M + MAC) (XOR) 128 bits key K
8. Cipher_Text =Append C with Cipher_Text;
9. bit_count = bit_count + 128;
10.M=next 120 bits of PlainText;
                  /*pad'0'at end of M, if M !=120 bits*/
11. If (bit_count != NOB)
         goto step 4;
    else  {  /*new key generation*/
             Perform shift operation at K;
             K = K + NOB;
             K = K (XOR) NOB;
             goto step 3;
          }
12. END
}
```

***B.3.* Decryption Algorithm:-**

Decryption algorithm divides whole cipher text into 128-bit blocks which are decrypted by same multiple keys (which was used for encryption). After every NOB bits, key will be changed and this procedure is repeated till encrypted message ends. The steps are as follows:

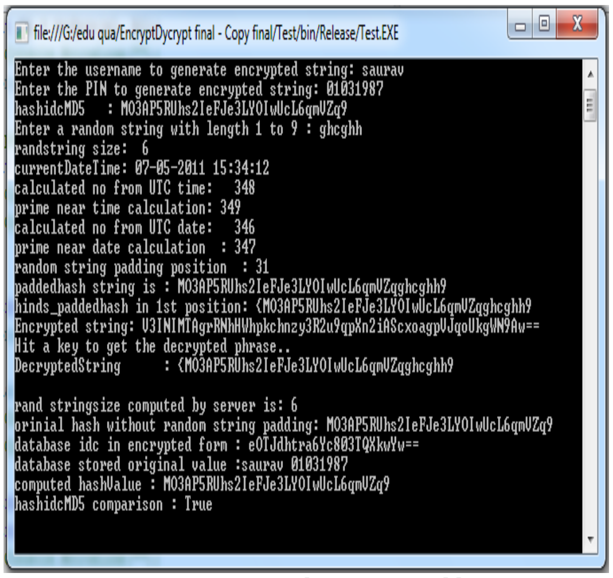
```
Decryption (Key K, NOB, CipherText)
{
1. Plain_Text = null;
2. C = First 128 bits of CipherText;
3. bit_count = 0;
4. If (C == null)
       (goto step 11);
   else (goto step 5);
5. 128 bits (M + MAC) = C (XOR) 128 bits Key K.
6. Get 120 bits M, 8 bits MAC from 128 bits(M+MAC);
7. Append M with Plain_Text;
8. C = next 128 bits of CipherText ;
9. 8 bits MAC' = MD (NOB, M);
10. If (MAC' == MAC)
   { /*Message integrity verified*/
       bit_count+ =  128;
      If (bit_count != NOB)
          goto step 4;
      else {   /*new key generation*/
               Perform shift operation at k;
               K = K + NOB;
               K = K (XOR) NOB;
               goto step 3;
           }
   }
   else   Take proper action;  /*Message not integrated*/
 11. END
}
```

IV. EXPERIMENTS, RESULTS AND BENEFITS

We have used C# language in domain of ASP.NET 3.5 as a coding standard. To perform the simulation, we have used desktop with Windows XP sp2, Visual Studio 2008, Intel Dual Core 2.53 GHz processor and 2 GB RAM.

IV.A. *Experiment 1:-*

We have implemented our 1st algorithm, i.e. padding a random string in different position of the hashed string in the figure. This padding position depends on UTC time and Nearest Prime number. The snapshots of outputs are given in Figures 3 & 4.

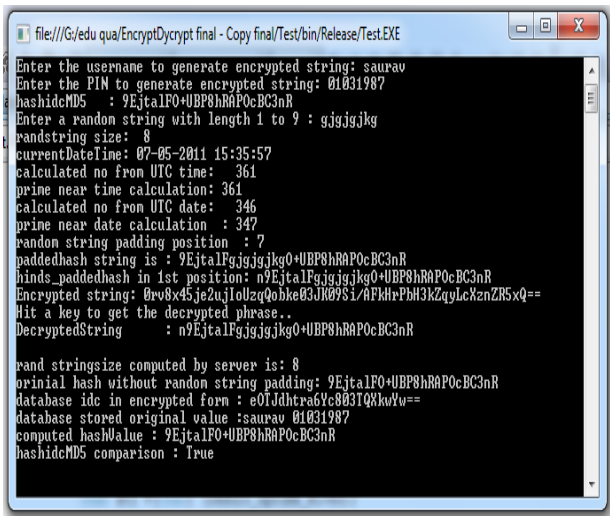


file:///G:/edu qua/EncryptDycrypt final - Copy final/Test/bin/Release/Test.EXE

```
Enter the username to generate encrypted string: saurav
Enter the PIN to generate encrypted string: 01031987
hashidcMD5   : MO3AP5RUhs2IeFJe3LYOIwUcL6qmVZq9
Enter a random string with length 1 to 9 : ghcghh
randstring size:  6
currentDateTime: 07-05-2011 15:34:12
calculated no from UTC time:   348
prime near time calculation: 349
calculated no from UTC date:   346
prime near date calculation  : 347
random string padding position  : 31
paddedhash string is : MO3AP5RUhs2IeFJe3LYOIwUcL6qmVZqghcghh9
hinds_paddedhash in 1st position: <MO3AP5RUhs2IeFJe3LYOIwUcL6qmVZqghcghh9
Encrypted string: V3INIMTAgrRNhHWhpkchnzy3R2u9qpXn2iAScxoagpVJqoUkgWN9Aw==
Hit a key to get the decrypted phrase..
DecryptedString      : <MO3AP5RUhs2IeFJe3LYOIwUcL6qmVZqghcghh9

rand stringsize computed by server is: 6
orinial hash without random string padding: MO3AP5RUhs2IeFJe3LYOIwUcL6qmVZq9
database idc in encrypted form : eOTJdhtra6Yc803TQXkwYw==
database stored original value :saurav 01031987
computed hashValue : MO3AP5RUhs2IeFJe3LYOIwUcL6qmVZq9
hashidcMD5 comparison : True
```

Figure 3: Random string padding



```
file:///G:/edu qua/EncryptDycrypt final - Copy final/Test/bin/Release/Test.EXE
Enter the username to generate encrypted string: saurav
Enter the PIN to generate encrypted string: 01031987
hashidcMD5   : 9EjtalFO+UBP8hRAPOcBC3nR
Enter a random string with length 1 to 9 : gjgjgjkg
randstring size:  8
currentDateTime: 07-05-2011 15:35:57
calculated no from UTC time:   361
prime near time calculation: 361
calculated no from UTC date:   346
prime near date calculation  : 347
random string padding position  : 7
paddedhash string is : 9EjtalFgjgjgjkgO+UBP8hRAPOcBC3nR
hinds_paddedhash in 1st position: n9EjtalFgjgjgjkgO+UBP8hRAPOcBC3nR
Encrypted string: 0rv8x45je2ujIoUzqQobke03JK09Si/AFkHrPbH3kZqyLcXznZR5xQ==
Hit a key to get the decrypted phrase..
DecryptedString      : n9EjtalFgjgjgjkgO+UBP8hRAPOcBC3nR

rand stringsize computed by server is: 8
orinial hash without random string padding: 9EjtalFO+UBP8hRAPOcBC3nR
database idc in encrypted form : eOTJdhtra6Yc803TQXkwYw==
database stored original value :saurav 01031987
computed hashValue : 9EjtalFO+UBP8hRAPOcBC3nR
hashidcMD5 comparison : True
```

Figure 4:Random string padding

It is difficult for attacker to predict the padding position of random string as that position is calculated depending on nearest prime number and UTC time.

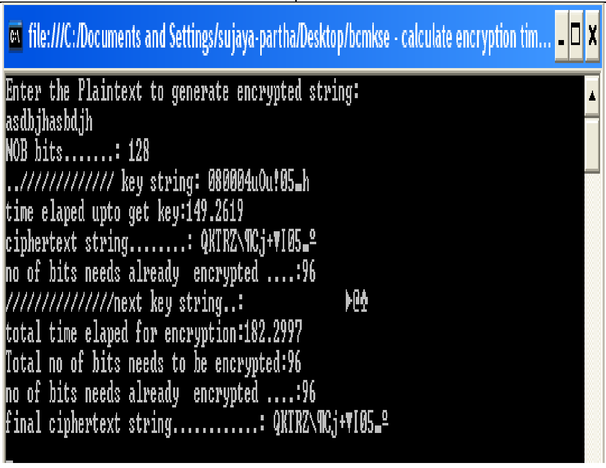
IV.B. *Experiment 2:-*

We have implemented our algorithm and get different encryption time values for encrypting different size of plaintext. Those time data sets are given in table 1.

Figure 5 & 6 depict the snapshots of output of time taken for MKBCSE encryption of data of different sizes.

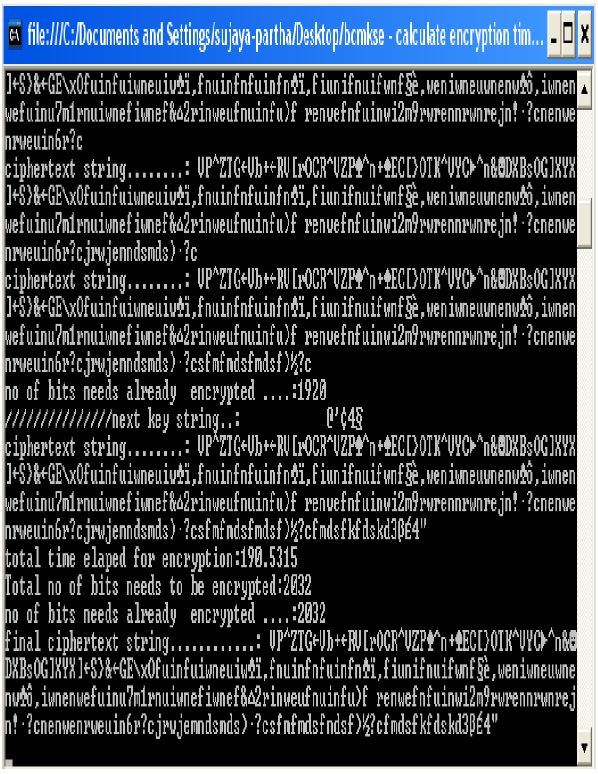
Table 1: Time taken for MKBCSE encryption

Data(bit) of different sizes	MKBCSE Encryption Times (millisec)
0	0
32	182.1224
96	182.2997
336	182.8862
640	185.9429
944	187.8948
1280	188.9976
1600	189.6899
1920	190.0884
1992	190.2382
2032	190.5315



```
file:///C:/Documents and Settings/sujaya-partha/Desktop/bcmkse - calculate encryption tim...
Enter the Plaintext to generate encrypted string:
asdbjhasbdjh
NOB bits.......: 128
..//////////// key string: 080004u0u!05▂h
time elaped upto get key:149.2619
ciphertext string........: QKTRZ\¶Cj+▼I05▂º
no of bits needs already  encrypted ....:96
//////////////next key string..:          ▶☺☻
total time elaped for encryption:182.2997
Total no of bits needs to be encrypted:96
no of bits needs already  encrypted ....:96
final ciphertext string...........: QKTRZ\¶Cj+▼I05▂º
```

Figure 5: Time taken for MKBCSE data Encryption



```
file:///C:/Documents and Settings/sujaya-partha/Desktop/bcmkse - calculate encryption tim...
]←S}&←GE\xOfuinfuiwneuiw♠ï,fnuinfnfuinfn♠ï,fiunifnuifwnf§è,weniwneuwnenw♠ô,iwnen
wefuinu7m1rnuiwnefiwnef&⌂2rinweufnuinfu)f renwefnfuinwi2m9rwrennrwnrejn!·?cnenwe
nrweuin6r?c
ciphertext string........: UP^ZTG←Ub+←RU[rOCR^UZP♠^n+♠EC[}OTK^UYC▶^n&●DXBsOG]XYX
]←S}&←GE\xOfuinfuiwneuiw♠ï,fnuinfnfuinfn♠ï,fiunifnuifwnf§è,weniwneuwnenw♠ô,iwnen
wefuinu7m1rnuiwnefiwnef&⌂2rinweufnuinfu)f renwefnfuinwi2m9rwrennrwnrejn!·?cnenwe
nrweuin6r?cjrwjemndsmds)·?c
ciphertext string........: UP^ZTG←Ub+←RU[rOCR^UZP♠^n+♠EC[}OTK^UYC▶^n&●DXBsOG]XYX
]←S}&←GE\xOfuinfuiwneuiw♠ï,fnuinfnfuinfn♠ï,fiunifnuifwnf§è,weniwneuwnenw♠ô,iwnen
wefuinu7m1rnuiwnefiwnef&⌂2rinweufnuinfu)f renwefnfuinwi2m9rwrennrwnrejn!·?cnenwe
nrweuin6r?cjrwjemndsmds)·?csfmfmdsfmdsf)½?c
no of bits needs already  encrypted ....:1920
///////////////next key string..:           @'¢4§
ciphertext string........: UP^ZTG←Ub+←RU[rOCR^UZP♠^n+♠EC[}OTK^UYC▶^n&●DXBsOG]XYX
]←S}&←GE\xOfuinfuiwneuiw♠ï,fnuinfnfuinfn♠ï,fiunifnuifwnf§è,weniwneuwnenw♠ô,iwnen
wefuinu7m1rnuiwnefiwnef&⌂2rinweufnuinfu)f renwefnfuinwi2m9rwrennrwnrejn!·?cnenwe
nrweuin6r?cjrwjemndsmds)·?csfmfmdsfmdsf)½?cfmdsfkfdskd3ßÉ4"
total time elaped for encryption:190.5315
Total no of bits needs to be encrypted:2032
no of bits needs already  encrypted ....:2032
final ciphertext string...........: UP^ZTG←Ub+←RU[rOCR^UZP♠^n+♠EC[}OTK^UYC▶^n&●
DXBsOG]XYX]←S}&←GE\xOfuinfuiwneuiw♠ï,fnuinfnfuinfn♠ï,fiunifnuifwnf§è,weniwneuwne
nw♠ô,iwnenwefuinu7m1rnuiwnefiwnef&⌂2rinweufnuinfu)f renwefnfuinwi2m9rwrennrwnrej
n!·?cnenwenrweuin6r?cjrwjemndsmds)·?csfmfmdsfmdsf)½?cfmdsfkfdskd3ßÉ4"
```

Figure 6: Time taken for MKBCSE data Encryption

Above values are plotted in a graph (in Figure 7) and we calculate an approximate encryption rate value.

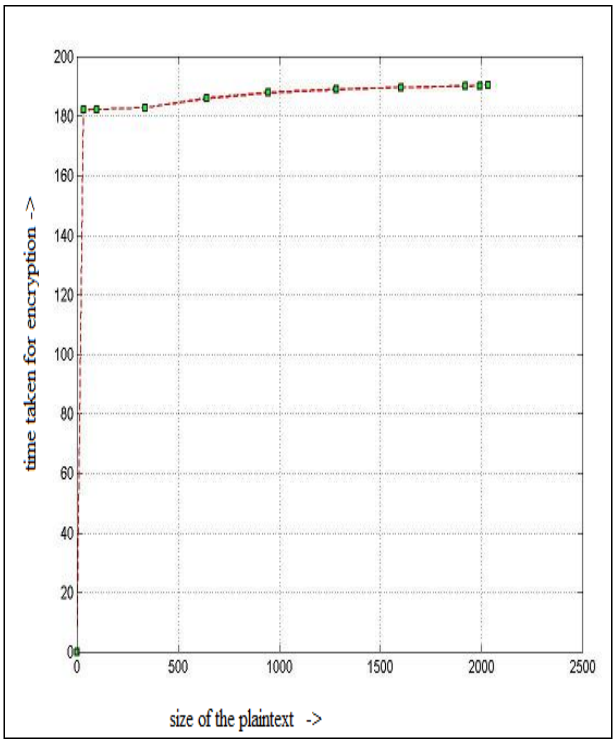


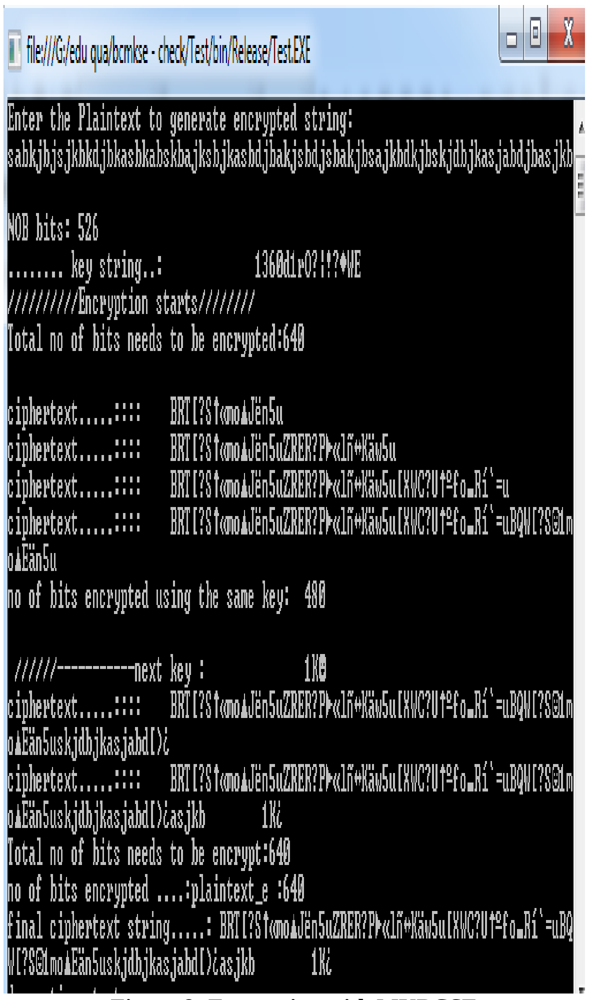
Figure 7:Encrypting times for different plaintext size

Then we compare it with DES, 3DES and AES. The time taken for encryption of data are given in Table 2.

Table 2:Comparative study with other existing symmetric encryption algorithm

Algorithm	Encryption Time (second/Kilobit)
DES(56 bit key)	0.296957406
3DES(56bit three key)	0.765329835
AES(128 bit key)	0.264974528
Blowfish(128 bit key)	0.204341515
Block Cipher Multiple key Symmetric Encryption (128 bit key)	0.177459598
Proposed MKSEBC(128 bit key)	0.172733488 (approximate)

So, it is clear that the time required to encrypt 1Kbit data taken by MKBCSE algorithm is lesser than the existing symmetric key algorithms. Figure 8 & Figure 9 depicts the snapshots for MKBCSE Encryption and Decryption respectively.



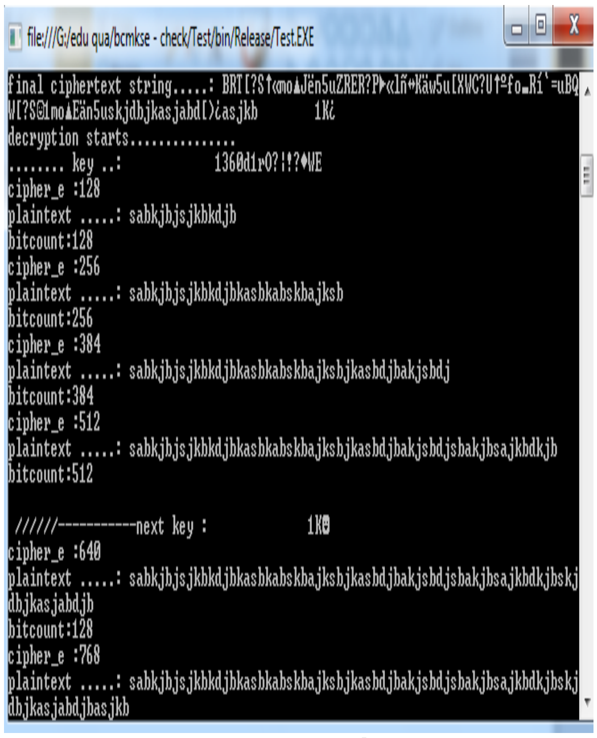
```
Enter the Plaintext to generate encrypted string:
sabkjbjsjkbkdjbkasbkabskbajksbjkasbdjbakjsbdjsbakjbsajkbdkjbskjdbjkasjabdjbasjkb

NOB bits: 526
........ key string..:          1360d1r0?¦!?♦WE
//////////Encryption starts////////
Total no of bits needs to be encrypted:640


ciphertext.....::::   BRT[?S↑«mo▲Jën5u
ciphertext.....::::   BRT[?S↑«mo▲Jën5uZRER?P►«lñ↔Käw5u
ciphertext.....::::   BRT[?S↑«mo▲Jën5uZRER?P►«lñ↔Käw5u[XWC?U↑º-fo▬Rí`=u
ciphertext.....::::   BRT[?S↑«mo▲Jën5uZRER?P►«lñ↔Käw5u[XWC?U↑º-fo▬Rí`=uBQW[?S☺1m
o▲Eän5u
no of bits encrypted using the same key:  480


 //////----------next key :            1K☻
ciphertext.....::::   BRT[?S↑«mo▲Jën5uZRER?P►«lñ↔Käw5u[XWC?U↑º-fo▬Rí`=uBQW[?S☺1m
o▲Eän5uskjdbjkasjabd[)¿
ciphertext.....::::   BRT[?S↑«mo▲Jën5uZRER?P►«lñ↔Käw5u[XWC?U↑º-fo▬Rí`=uBQW[?S☺1m
o▲Eän5uskjdbjkasjabd[)¿asjkb       1K¿
Total no of bits needs to be encrypt:640
no of bits encrypted ....:plaintext_e :640
final ciphertext string.....: BRT[?S↑«mo▲Jën5uZRER?P►«lñ↔Käw5u[XWC?U↑º-fo▬Rí`=uBQ
W[?S☺1mo▲Eän5uskjdbjkasjabd[)¿asjkb       1K¿
```

Figure 8: Encryption with MKBCSE



```
final ciphertext string.....: BRT[?S↑«mo▲Jën5uZRER?P►«lñ↔Käw5u[XWC?U↑º-fo▬Rí`=uBQ
W[?S☺1mo▲Eän5uskjdbjkasjabd[)¿asjkb       1K¿
decryption starts...............
........ key ..:           1360d1r0?¦!?♦WE
cipher_e :128
plaintext .....: sabkjbjsjkbkdjb
bitcount:128
cipher_e :256
plaintext .....: sabkjbjsjkbkdjbkasbkabskbajksb
bitcount:256
cipher_e :384
plaintext .....: sabkjbjsjkbkdjbkasbkabskbajksbjkasbdjbakjsbdj
bitcount:384
cipher_e :512
plaintext .....: sabkjbjsjkbkdjbkasbkabskbajksbjkasbdjbakjsbdjsbakjbsajkbdkjb
bitcount:512

 //////----------next key :            1K☻
cipher_e :640
plaintext .....: sabkjbjsjkbkdjbkasbkabskbajksbjkasbdjbakjsbdjsbakjbsajkbdkjbskj
dbjkasjabdjb
bitcount:128
cipher_e :768
plaintext .....: sabkjbjsjkbkdjbkasbkabskbajksbjkasbdjbakjsbdjsbakjbsajkbdkjbskj
dbjkasjabdjbasjkb
```

Figure 9: Decryption with MKBCSE

- ***Advantages of the MKBCSE algorithm***: -

1. ***Encryption time of MKBCSE is less than existing DES, 3DES, AES, Blowfish, Block Cipher Multiple key Symmetric Encryption.***
2. Due to change of keys after random bits, it is very hard to perform cryptanalysis to deduce the secret keys.
3. Due to 128-bit key and n-bit NOB, cipher becomes more secure. As a total (2^{128} +$2^{n)}$ number of permutations are possible where 128 >= n>=7. So, brute force attack is much time taking, nearly 1.079x1028 year when n=7;
4. No need of key exchange. So it reduces network traffic.
5. It involves few XORs, SHIFTs, additions, comparisons and appends operations. So, it works faster.

V. CONCLUSION

In this paper, it is structured to describe about possible preventions of password guessing & brute-force attack by ***padding a random string in the different position*** of the password. The proposed MKBCSE key generation, encryption and decryption algorithms are efficient because they are simple and easily implementable. ***The algorithm achieves less encryption time than existing DES, 3DES, AES, Blowfish, Block Cipher Multiple key Symmetric Encryption***, which has been justified in the previous section. This paper presents a new way of using multiple keys concept without increasing message overhead. Instead of developing a complex algorithm by involving complex & time taking operations, this paper emphasizes to think about logically complex algorithm with simple operations. More modification in experimental section of this logic will be the future work of this paper.

ACKNOWLEDGMENT

We want to thank our parents and supervisor Professor Md. Headayetullah for their needful encouragement to this work.

REFERENCES

[1] Hole, K.J., Moen, V., Tjøstheim, T (2006). Case study: Online banking security. IEEE Security & Privacy 4(2), 14–20.
[2] Hole, K.J.: Tjøstheim, T., Moen, V., Netland, L., Espelid, Y., Klingsheim, and A.N.: Next generation internet banking in Norway, submitted to IEEE Security & Privacy, 2007.
[3] FIPS Publication 46-3, "Data Encryption Standard (DES)" ,U.S. DoC/NIST, October 25, 1999.
[4] B. Schneier, Applied Cryptography, John Wiley & Sons, New York, 1994.
[5] ***Saurav Mallik***, Sutapa Majee, Md. Headayetullah, "Proposed Novel Conceptual 3-Tier Security Model for Internet Banking System", ICMLC 2011, Vol. 4, page:250-254, Singapore.
[6] FIPS Publication 197, "Advanced Encryption Standard (AES)." U.S. DoC/NIST, November 26, 2001.
[7] Candid American National Standard for Financial Services X9.52- 1998, "Triple Data Encryption Algorithm Modes of Operation." American Bankers Association, Washington, D.C., July 29, 1998.

AUTHORS

Saurav Mallik is M.Tech student of Dr. B. C. Roy Engineering College in Computer Science and Engineering under West Bengal University of Technology, India.

Sutapa Majee is M.Tech student of Dr. B. C. Roy Engineering College in Computer Science and Engineering under West Bengal University of Technology, India.

Arun Kanti Manna is M.Tech student of Dr. B. C. Roy Engineering College in Computer Science and Engineering under West Bengal University of Technology, India.

Md.Headayetullah is an Assistant Professor of department of Computer Science and Engineering and Information Technology of Dr. B.C Roy Engineering College, Durgapur. He has submitted his PhD Thesis in the department of Computer Science and Engineering. He has more than 6 Intern national publications in Reputed Journal. He guided so many students in PG Level. Professor Headayetullah is working as reviewer of so many journals and books. Professor, Headayetullah is the members of IAENG and IACSIT respectively.