Manpreet Chavan
10
TE COMPS

EXPERIMENT 3

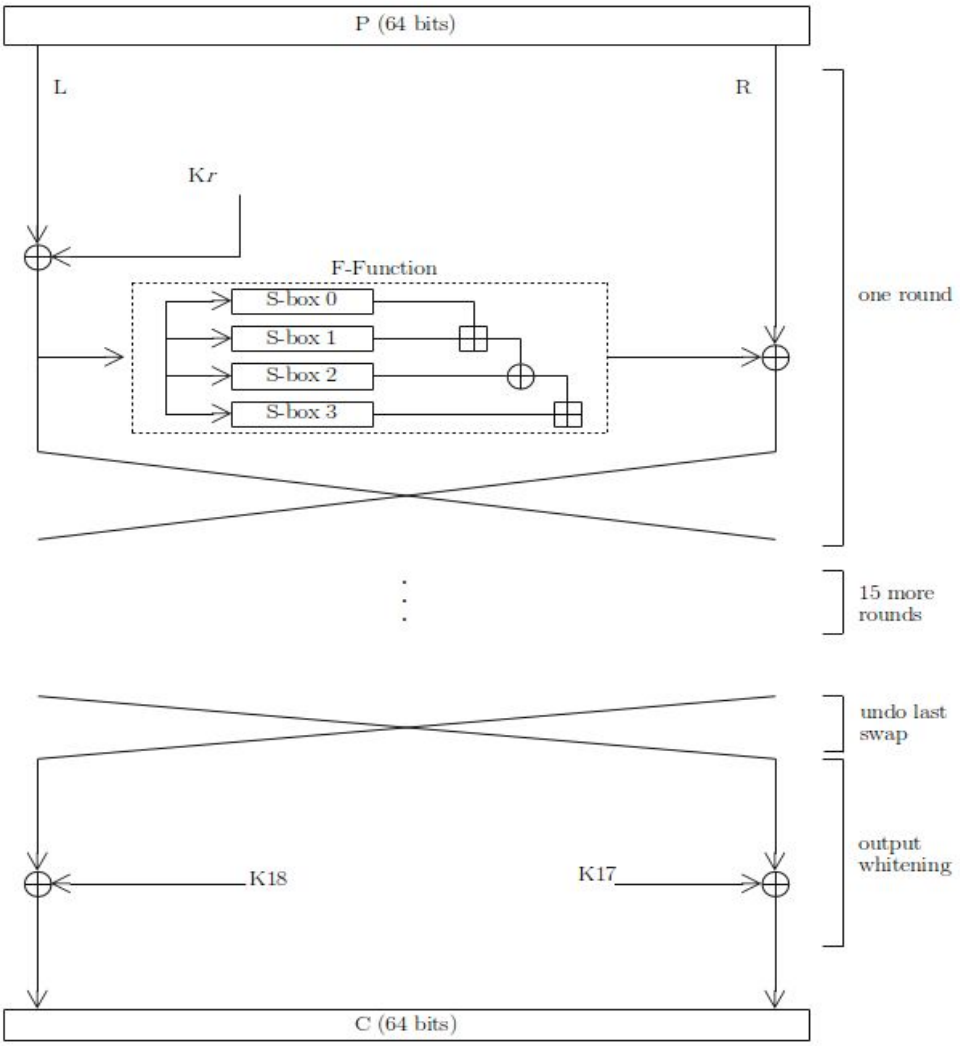
Aim : To implement blowfish algorithm.

Theory :

Introduction :
Blowfish is a symmetric block cipher that can be used as a drop-in replacement for DES or IDEA. It takes a variable-length key, from 32 bits to 448 bits, making it ideal for both domestic and exportable use. Blowfish was designed in 1993 by Bruce Schneier as a fast, free alternative to existing encryption algorithms. Since then it has been analyzed considerably, and it is slowly gaining acceptance as a strong encryption algorithm. Blowfish is unpatented and license-free, and is available free for all uses.

General overview :

Blowfish has a 64-bit block size and a variable key length from 32 bits up to 448 bits.[3] It is a 16-round Feistel cipher and uses large key-dependent S-boxes. In structure it resembles CAST-128, which uses fixed S-boxes.



P=Plaintext; C=Ciphertext; Kx= P-array-entry x

$\oplus$ = xor $\boxplus$ = addition mod 2^32

(Figure 1 :The Feistel structure of Blowfish)

The adjacent diagram shows Blowfish's encryption routine. Each line represents 32 bits. There are five subkey-arrays: one 18-entry P-array (denoted as K in the diagram, to avoid confusion with the Plaintext) and four 256-entry S-boxes (S0, S1, S2 and S3).

Every round *r* consists of 4 actions:

Action 1	XOR the left half (L) of the data with the *r* th P-array entry
Action 2	Use the XORed data as input for Blowfish's F-function
Action 3	XOR the F-function's output with the right half (R) of the data
Action 4	Swap L and R

The F-function splits the 32-bit input into four eight-bit quarters, and uses the quarters as input to the S-boxes. The S-boxes accept 8-bit input and produce 32-bit output. The outputs are added modulo 2^{32} and XORed to produce the final 32-bit output (see image in the upper right corner)

After the 16th round, undo the last swap, and XOR L with K18 and R with K17 (output whitening).

Decryption is exactly the same as encryption, except that P1, P2, …, P18 are used in the reverse order. This is not so obvious because xor is commutative and associative. A common misconception is to use inverse order of encryption as decryption algorithm (i.e. first XORing P17 and P18 to the ciphertext block, then using the P-entries in reverse order).

Blowfish's key schedule starts by initializing the P-array and S-boxes with values derived from the hexadecimal digits of pi, which contain no obvious pattern (see nothing up my sleeve number). The secret key is then, byte by byte, cycling the key if necessary, XORed with all the P-entries in order. A 64-bit all-zero block is then encrypted with the algorithm as it stands. The resultant ciphertext replaces P_1 and P_2. The same ciphertext is then encrypted again with the new

subkeys, and the new ciphertext replaces P_3 and P_4. This continues, replacing the entire P-array and all the S-box entries. In all, the Blowfish encryption algorithm will run 521 times to generate all the subkeys - about 4KB of data is processed.

Because the P-array is 576 bits long, and the key bytes are XORed through all these 576 bits during the initialization, many implementations support key sizes up to 576 bits. The reason for that is a discrepancy between the original Blowfish description, which uses 448-bit key, and its reference implementation, which uses 576-bit key. The test vectors for verifying third party implementations were also produced with 576-bit keys. When asked which Blowfish version is the correct one, Bruce Schneier answered: "*The test vectors should be used to determine the one true Blowfish*".

Another opinion is that the 448 bits limit is here to ensure that every bit of every subkey depends on every bit of the key,[] as the last four values of the P-array don't affect every bit of the ciphertext. This point should be taken in consideration for implementations with a different number of rounds, as even though it increases security against an exhaustive attack, it weakens the security guaranteed by the algorithm. And given the slow initialization of the cipher with each change of key, it is granted a natural protection against brute-force attacks, which doesn't really justify key sizes longer than 448 bits.

Algorithm :

```
uint32_t P[18];
uint32_t S[4][256];

uint32_t f (uint32_t x) {
  uint32_t h = S[0][x >> 24] + S[1][x >> 16 & 0xff];
  return ( h ^ S[2][x >> 8 & 0xff] ) + S[3][x & 0xff];
}
```

Encryption :

```
void encrypt (uint32_t & L, uint32_t & R) {
  for (int i=0 ; i<16 ; i += 2) {
```

```
        L ^= P[i];
        R ^= f(L);
        R ^= P[i+1];
        L ^= f(R);
  }
  L ^= P[16];
  R ^= P[17];
  swap (L, R);
}
```

Decryption:

```
void decrypt (uint32_t & L, uint32_t & R) {
  for (int i=16 ; i > 0 ; i -= 2) {
        L ^= P[i+1];
        R ^= f(L);
        R ^= P[i];
        L ^= f(R);
  }
  L ^= P[1];
  R ^= P[0];
  swap (L, R);
}

 // ...
 // initializing the P-array and S-boxes with values derived from pi; omitted in the example
 // ...
{
  for (int i=0 ; i<18 ; ++i)
        P[i] ^= key[i % keylen];
  uint32_t L = 0, R = 0;
  for (int i=0 ; i<18 ; i+=2) {
        encrypt (L, R);
        P[i] = L; P[i+1] = R;
  }
  for (int i=0 ; i<4 ; ++i)
        for (int j=0 ; j<256; j+=2) {
        encrypt (L, R);
        S[i][j] = L; S[i][j+1] = R;
```

```
        }
}
```

Code :

```
import java.util.*;
class Blowfish
{
   static String[] pArray;
   static String[] s0Array;
   static String[] s1Array;
   static String[] s2Array;
   static String[] s3Array;
   static String[] NewpArr;


   static int lengthp;
   static int lengths0;
   static int lengths1;
   static int lengths2;
   static int lengths3;

   static String plaintext,test1,temp,temp2;
   public static void main(String args[])
   {
      int[] p = new int[]{0x243F6A88, 0x85A308D3, 0x13198A2E, 0x03707344,
                          0xA4093822, 0x299F31D0, 0x082EFA98, 0xEC4E6C89,
                          0x452821E6, 0x38D01377, 0xBE5466CF, 0x34E90C6C,
                           0xC0AC29B7, 0xC97C50DD, 0x3F84D5B5, 0xB5470917,
                          0x9216D5D9, 0x8979FB1B};

      int[] s0 = new int[]{0xD1310BA6, 0x98DFB5AC, 0x2FFD72DB, 0xD01ADFB7,
                           0xB8E1AFED, 0x6A267E96, 0xBA7C9045, 0xF12C7F99,
                           0x24A19947, 0xB3916CF7, 0x0801F2E2, 0x858EFC16,
```

0x636920D8, 0x71574E69, 0xA458FEA3, 0xF4933D7E,
0x0D95748F, 0x728EB658, 0x718BCD58, 0x82154AEE,
0x7B54A41D, 0xC25A59B5, 0x9C30D539, 0x2AF26013,
0xC5D1B023, 0x286085F0, 0xCA417918, 0xB8DB38EF,
0x8E79DCB0, 0x603A180E, 0x6C9E0E8B, 0xB01E8A3E,
0xD71577C1, 0xBD314B27, 0x78AF2FDA, 0x55605C60,
0xE65525F3, 0xAA55AB94, 0x57489862, 0x63E81440,
0x55CA396A, 0x2AAB10B6, 0xB4CC5C34, 0x1141E8CE,
0xA15486AF, 0x7C72E993, 0xB3EE1411, 0x636FBC2A,
0x2BA9C55D, 0x741831F6, 0xCE5C3E16, 0x9B87931E,
0xAFD6BA33, 0x6C24CF5C, 0x7A325381, 0x28958677,
0x3B8F4898, 0x6B4BB9AF, 0xC4BFE81B, 0x66282193,
0x61D809CC, 0xFB21A991, 0x487CAC60, 0x5DEC8032,
0xEF845D5D, 0xE98575B1, 0xDC262302, 0xEB651B88,
0x23893E81, 0xD396ACC5, 0x0F6D6FF3, 0x83F44239,
0x2E0B4482, 0xA4842004, 0x69C8F04A, 0x9E1F9B5E,
0x21C66842, 0xF6E96C9A, 0x670C9C61, 0xABD388F0,
0x6A51A0D2, 0xD8542F68, 0x960FA728, 0xAB5133A3,
0x6EEF0B6C, 0x137A3BE4, 0xBA3BF050, 0x7EFB2A98,
0xA1F1651D, 0x39AF0176, 0x66CA593E, 0x82430E88,
0x8CEE8619, 0x456F9FB4, 0x7D84A5C3, 0x3B8B5EBE,
0xE06F75D8, 0x85C12073, 0x401A449F, 0x56C16AA6,
0x4ED3AA62, 0x363F7706, 0x1BFEDF72, 0x429B023D,
0x37D0D724, 0xD00A1248, 0xDB0FEAD3, 0x49F1C09B,
0x075372C9, 0x80991B7B, 0x25D479D8, 0xF6E8DEF7,
0xE3FE501A, 0xB6794C3B, 0x976CE0BD, 0x04C006BA,
0xC1A94FB6, 0x409F60C4, 0x5E5C9EC2, 0x196A2463,
0x68FB6FAF, 0x3E6C53B5, 0x1339B2EB, 0x3B52EC6F,
0x6DFC511F, 0x9B30952C, 0xCC814544, 0xAF5EBD09,
0xBEE3D004, 0xDE334AFD, 0x660F2807, 0x192E4BB3,
0xC0CBA857, 0x45C8740F, 0xD20B5F39, 0xB9D3FBDB,
0x5579C0BD, 0x1A60320A, 0xD6A100C6, 0x402C7279,
0x679F25FE, 0xFB1FA3CC, 0x8EA5E9F8, 0xDB3222F8,

```
                0x3C7516DF, 0xFD616B15, 0x2F501EC8, 0xAD0552AB,
                0x323DB5FA, 0xFD238760, 0x53317B48, 0x3E00DF82,
                0x9E5C57BB, 0xCA6F8CA0, 0x1A87562E, 0xDF1769DB,
                0xD542A8F6, 0x287EFFC3, 0xAC6732C6, 0x8C4F5573,
                0x695B27B0, 0xBBCA58C8, 0xE1FFA35D, 0xB8F011A0,
                0x10FA3D98, 0xFD2183B8, 0x4AFCB56C, 0x2DD1D35B,
                0x9A53E479, 0xB6F84565, 0xD28E49BC, 0x4BFB9790,
                0xE1DDF2DA, 0xA4CB7E33, 0x62FB1341, 0xCEE4C6E8,
                0xEF20CADA, 0x36774C01, 0xD07E9EFE, 0x2BF11FB4,
                0x95DBDA4D, 0xAE909198, 0xEAAD8E71, 0x6B93D5A0,
                0xD08ED1D0, 0xAFC725E0, 0x8E3C5B2F, 0x8E7594B7,
                0x8FF6E2FB, 0xF2122B64, 0x8888B812, 0x900DF01C,
                0x4FAD5EA0, 0x688FC31C, 0xD1CFF191, 0xB3A8C1AD,
                0x2F2F2218, 0xBE0E1777, 0xEA752DFE, 0x8B021FA1,
                0xE5A0CC0F, 0xB56F74E8, 0x18ACF3D6, 0xCE89E299,
                0xB4A84FE0, 0xFD13E0B7, 0x7CC43B81, 0xD2ADA8D9,
                0x165FA266, 0x80957705, 0x93CC7314, 0x211A1477,
                0xE6AD2065, 0x77B5FA86, 0xC75442F5, 0xFB9D35CF,
                0xEBCDAF0C, 0x7B3E89A0, 0xD6411BD3, 0xAE1E7E49,
                0x00250E2D, 0x2071B35E, 0x226800BB, 0x57B8E0AF,
                0x2464369B, 0xF009B91E, 0x5563911D, 0x59DFA6AA,
                0x78C14389, 0xD95A537F, 0x207D5BA2, 0x02E5B9C5,
                0x83260376, 0x6295CFA9, 0x11C81968, 0x4E734A41,
                0xB3472DCA, 0x7B14A94A, 0x1B510052, 0x9A532915,
                0xD60F573F, 0xBC9BC6E4, 0x2B60A476, 0x81E67400,
                0x08BA6FB5, 0x571BE91F, 0xF296EC6B, 0x2A0DD915,
                0xB6636521, 0xE7B9F9B6, 0xFF34052E, 0xC5855664,
            0x53B02D5D, 0xA99F8FA1, 0x08BA4799, 0x6E85076A};

int[] s1 = new int[]{0x4B7A70E9, 0xB5B32944, 0xDB75092E, 0xC4192623,
                0xAD6EA6B0, 0x49A7DF7D, 0x9CEE60B8, 0x8FEDB266,
                0xECAA8C71, 0x699A17FF, 0x5664526C, 0xC2B19EE1,
                0x193602A5, 0x75094C29, 0xA0591340, 0xE4183A3E,
```

0x3F54989A, 0x5B429D65, 0x6B8FE4D6, 0x99F73FD6,
0xA1D29C07, 0xEFE830F5, 0x4D2D38E6, 0xF0255DC1,
0x4CDD2086, 0x8470EB26, 0x6382E9C6, 0x021ECC5E,
0x09686B3F, 0x3EBAEFC9, 0x3C971814, 0x6B6A70A1,
0x687F3584, 0x52A0E286, 0xB79C5305, 0xAA500737,
0x3E07841C, 0x7FDEAE5C, 0x8E7D44EC, 0x5716F2B8,
0xB03ADA37, 0xF0500C0D, 0xF01C1F04, 0x0200B3FF,
0xAE0CF51A, 0x3CB574B2, 0x25837A58, 0xDC0921BD,
0xD19113F9, 0x7CA92FF6, 0x94324773, 0x22F54701,
0x3AE5E581, 0x37C2DADC, 0xC8B57634, 0x9AF3DDA7,
0xA9446146, 0x0FD0030E, 0xECC8C73E, 0xA4751E41,
0xE238CD99, 0x3BEA0E2F, 0x3280BBA1, 0x183EB331,
0x4E548B38, 0x4F6DB908, 0x6F420D03, 0xF60A04BF,
0x2CB81290, 0x24977C79, 0x5679B072, 0xBCAF89AF,
0xDE9A771F, 0xD9930810, 0xB38BAE12, 0xDCCF3F2E,
0x5512721F, 0x2E6B7124, 0x501ADDE6, 0x9F84CD87,
0x7A584718, 0x7408DA17, 0xBC9F9ABC, 0xE94B7D8C,
0xEC7AEC3A, 0xDB851DFA, 0x63094366, 0xC464C3D2,
0xEF1C1847, 0x3215D908, 0xDD433B37, 0x24C2BA16,
0x12A14D43, 0x2A65C451, 0x50940002, 0x133AE4DD,
0x71DFF89E, 0x10314E55, 0x81AC77D6, 0x5F11199B,
0x043556F1, 0xD7A3C76B, 0x3C11183B, 0x5924A509,
0xF28FE6ED, 0x97F1FBFA, 0x9EBABF2C, 0x1E153C6E,
0x86E34570, 0xEAE96FB1, 0x860E5E0A, 0x5A3E2AB3,
0x771FE71C, 0x4E3D06FA, 0x2965DCB9, 0x99E71D0F,
0x803E89D6, 0x5266C825, 0x2E4CC978, 0x9C10B36A,
0xC6150EBA, 0x94E2EA78, 0xA5FC3C53, 0x1E0A2DF4,
0xF2F74EA7, 0x361D2B3D, 0x1939260F, 0x19C27960,
0x5223A708, 0xF71312B6, 0xEBADFE6E, 0xEAC31F66,
0xE3BC4595, 0xA67BC883, 0xB17F37D1, 0x018CFF28,
0xC332DDEF, 0xBE6C5AA5, 0x65582185, 0x68AB9802,
0xEECEA50F, 0xDB2F953B, 0x2AEF7DAD, 0x5B6E2F84,
0x1521B628, 0x29076170, 0xECDD4775, 0x619F1510,

```
                0x13CCA830, 0xEB61BD96, 0x0334FE1E, 0xAA0363CF,
                0xB5735C90, 0x4C70A239, 0xD59E9E0B, 0xCBAADE14,
                0xEECC86BC, 0x60622CA7, 0x9CAB5CAB, 0xB2F3846E,
                0x648B1EAF, 0x19BDF0CA, 0xA02369B9, 0x655ABB50,
                0x40685A32, 0x3C2AB4B3, 0x319EE9D5, 0xC021B8F7,
                0x9B540B19, 0x875FA099, 0x95F7997E, 0x623D7DA8,
                0xF837889A, 0x97E32D77, 0x11ED935F, 0x16681281,
                0x0E358829, 0xC7E61FD6, 0x96DEDFA1, 0x7858BA99,
                0x57F584A5, 0x1B227263, 0x9B83C3FF, 0x1AC24696,
                0xCDB30AEB, 0x532E3054, 0x8FD948E4, 0x6DBC3128,
                0x58EBF2EF, 0x34C6FFEA, 0xFE28ED61, 0xEE7C3C73,
                0x5D4A14D9, 0xE864B7E3, 0x42105D14, 0x203E13E0,
                0x45EEE2B6, 0xA3AAABEA, 0xDB6C4F15, 0xFACB4FD0,
                0xC742F442, 0xEF6ABBB5, 0x654F3B1D, 0x41CD2105,
                0xD81E799E, 0x86854DC7, 0xE44B476A, 0x3D816250,
                0xCF62A1F2, 0x5B8D2646, 0xFC8883A0, 0xC1C7B6A3,
                0x7F1524C3, 0x69CB7492, 0x47848A0B, 0x5692B285,
                0x095BBF00, 0xAD19489D, 0x1462B174, 0x23820E00,
                0x58428D2A, 0x0C55F5EA, 0x1DADF43E, 0x233F7061,
                0x3372F092, 0x8D937E41, 0xD65FECF1, 0x6C223BDB,
                0x7CDE3759, 0xCBEE7460, 0x4085F2A7, 0xCE77326E,
                0xA6078084, 0x19F8509E, 0xE8EFD855, 0x61D99735,
                0xA969A7AA, 0xC50C06C2, 0x5A04ABFC, 0x800BCADC,
                0x9E447A2E, 0xC3453484, 0xFDD56705, 0x0E1E9EC9,
                0xDB73DBD3, 0x105588CD, 0x675FDA79, 0xE3674340,
                0xC5C43465, 0x713E38D8, 0x3D28F89E, 0xF16DFF20,
            0x153E21E7, 0x8FB03D4A, 0xE6E39F2B, 0xDB83ADF7};

int[] s2 = new int[]{0xE93D5A68, 0x948140F7, 0xF64C261C, 0x94692934,
            0x411520F7, 0x7602D4F7, 0xBCF46B2E, 0xD4A20068,
            0xD4082471, 0x3320F46A, 0x43B7D4B7, 0x500061AF,
            0x1E39F62E, 0x97244546, 0x14214F74, 0xBF8B8840,
            0x4D95FC1D, 0x96B591AF, 0x70F4DDD3, 0x66A02F45,
```

0xBFBC09EC, 0x03BD9785, 0x7FAC6DD0, 0x31CB8504,
0x96EB27B3, 0x55FD3941, 0xDA2547E6, 0xABCA0A9A,
0x28507825, 0x530429F4, 0x0A2C86DA, 0xE9B66DFB,
0x68DC1462, 0xD7486900, 0x680EC0A4, 0x27A18DEE,
0x4F3FFEA2, 0xE887AD8C, 0xB58CE006, 0x7AF4D6B6,
0xAACE1E7C, 0xD3375FEC, 0xCE78A399, 0x406B2A42,
0x20FE9E35, 0xD9F385B9, 0xEE39D7AB, 0x3B124E8B,
0x1DC9FAF7, 0x4B6D1856, 0x26A36631, 0xEAE397B2,
0x3A6EFA74, 0xDD5B4332, 0x6841E7F7, 0xCA7820FB,
0xFB0AF54E, 0xD8FEB397, 0x454056AC, 0xBA489527,
0x55533A3A, 0x20838D87, 0xFE6BA9B7, 0xD096954B,
0x55A867BC, 0xA1159A58, 0xCCA92963, 0x99E1DB33,
0xA62A4A56, 0x3F3125F9, 0x5EF47E1C, 0x9029317C,
0xFDF8E802, 0x04272F70, 0x80BB155C, 0x05282CE3,
0x95C11548, 0xE4C66D22, 0x48C1133F, 0xC70F86DC,
0x07F9C9EE, 0x41041F0F, 0x404779A4, 0x5D886E17,
0x325F51EB, 0xD59BC0D1, 0xF2BCC18F, 0x41113564,
0x257B7834, 0x602A9C60, 0xDFF8E8A3, 0x1F636C1B,
0x0E12B4C2, 0x02E1329E, 0xAF664FD1, 0xCAD18115,
0x6B2395E0, 0x333E92E1, 0x3B240B62, 0xEEBEB922,
0x85B2A20E, 0xE6BA0D99, 0xDE720C8C, 0x2DA2F728,
0xD0127845, 0x95B794FD, 0x647D0862, 0xE7CCF5F0,
0x5449A36F, 0x877D48FA, 0xC39DFD27, 0xF33E8D1E,
0x0A476341, 0x992EFF74, 0x3A6F6EAB, 0xF4F8FD37,
0xA812DC60, 0xA1EBDDF8, 0x991BE14C, 0xDB6E6B0D,
0xC67B5510, 0x6D672C37, 0x2765D43B, 0xDCD0E804,
0xF1290DC7, 0xCC00FFA3, 0xB5390F92, 0x690FED0B,
0x667B9FFB, 0xCEDB7D9C, 0xA091CF0B, 0xD9155EA3,
0xBB132F88, 0x515BAD24, 0x7B9479BF, 0x763BD6EB,
0x37392EB3, 0xCC115979, 0x8026E297, 0xF42E312D,
0x6842ADA7, 0xC66A2B3B, 0x12754CCC, 0x782EF11C,
0x6A124237, 0xB79251E7, 0x06A1BBE6, 0x4BFB6350,
0x1A6B1018, 0x11CAEDFA, 0x3D25BDD8, 0xE2E1C3C9,

0x44421659, 0x0A121386, 0xD90CEC6E, 0xD5ABEA2A,
0x64AF674E, 0xDA86A85F, 0xBEBFE988, 0x64E4C3FE,
0x9DBC8057, 0xF0F7C086, 0x60787BF8, 0x6003604D,
0xD1FD8346, 0xF6381FB0, 0x7745AE04, 0xD736FCCC,
0x83426B33, 0xF01EAB71, 0xB0804187, 0x3C005E5F,
0x77A057BE, 0xBDE8AE24, 0x55464299, 0xBF582E61,
0x4E58F48F, 0xF2DDFDA2, 0xF474EF38, 0x8789BDC2,
0x5366F9C3, 0xC8B38E74, 0xB475F255, 0x46FCD9B9,
0x7AEB2661, 0x8B1DDF84, 0x846A0E79, 0x915F95E2,
0x466E598E, 0x20B45770, 0x8CD55591, 0xC902DE4C,
0xB90BACE1, 0xBB8205D0, 0x11A86248, 0x7574A99E,
0xB77F19B6, 0xE0A9DC09, 0x662D09A1, 0xC4324633,
0xE85A1F02, 0x09F0BE8C, 0x4A99A025, 0x1D6EFE10,
0x1AB93D1D, 0x0BA5A4DF, 0xA186F20F, 0x2868F169,
0xDCB7DA83, 0x573906FE, 0xA1E2CE9B, 0x4FCD7F52,
0x50115E01, 0xA70683FA, 0xA002B5C4, 0x0DE6D027,
0x9AF88C27, 0x773F8641, 0xC3604C06, 0x61A806B5,
0xF0177A28, 0xC0F586E0, 0x006058AA, 0x30DC7D62,
0x11E69ED7, 0x2338EA63, 0x53C2DD94, 0xC2C21634,
0xBBCBEE56, 0x90BCB6DE, 0xEBFC7DA1, 0xCE591D76,
0x6F05E409, 0x4B7C0188, 0x39720A3D, 0x7C927C24,
0x86E3725F, 0x724D9DB9, 0x1AC15BB4, 0xD39EB8FC,
0xED545578, 0x08FCA5B5, 0xD83D7CD3, 0x4DAD0FC4,
0x1E50EF5E, 0xB161E6F8, 0xA28514D9, 0x6C51133C,
0x6FD5C7E7, 0x56E14EC4, 0x362ABFCE, 0xDDC6C837,
0xD79A3234, 0x92638212, 0x670EFA8E, 0x406000E0};

int[] s3 = new int[]{0x3A39CE37, 0xD3FAF5CF, 0xABC27737, 0x5AC52D1B,
0x5CB0679E, 0x4FA33742, 0xD3822740, 0x99BC9BBE,
0xD5118E9D, 0xBF0F7315, 0xD62D1C7E, 0xC700C47B,
0xB78C1B6B, 0x21A19045, 0xB26EB1BE, 0x6A366EB4,
0x5748AB2F, 0xBC946E79, 0xC6A376D2, 0x6549C2C8,
0x530FF8EE, 0x468DDE7D, 0xD5730A1D, 0x4CD04DC6,

0x2939BBDB, 0xA9BA4650, 0xAC9526E8, 0xBE5EE304,
0xA1FAD5F0, 0x6A2D519A, 0x63EF8CE2, 0x9A86EE22,
0xC089C2B8, 0x43242EF6, 0xA51E03AA, 0x9CF2D0A4,
0x83C061BA, 0x9BE96A4D, 0x8FE51550, 0xBA645BD6,
0x2826A2F9, 0xA73A3AE1, 0x4BA99586, 0xEF5562E9,
0xC72FEFD3, 0xF752F7DA, 0x3F046F69, 0x77FA0A59,
0x80E4A915, 0x87B08601, 0x9B09E6AD, 0x3B3EE593,
0xE990FD5A, 0x9E34D797, 0x2CF0B7D9, 0x022B8B51,
0x96D5AC3A, 0x017DA67D, 0xD1CF3ED6, 0x7C7D2D28,
0x1F9F25CF, 0xADF2B89B, 0x5AD6B472, 0x5A88F54C,
0xE029AC71, 0xE019A5E6, 0x47B0ACFD, 0xED93FA9B,
0xE8D3C48D, 0x283B57CC, 0xF8D56629, 0x79132E28,
0x785F0191, 0xED756055, 0xF7960E44, 0xE3D35E8C,
0x15056DD4, 0x88F46DBA, 0x03A16125, 0x0564F0BD,
0xC3EB9E15, 0x3C9057A2, 0x97271AEC, 0xA93A072A,
0x1B3F6D9B, 0x1E6321F5, 0xF59C66FB, 0x26DCF319,
0x7533D928, 0xB155FDF5, 0x03563482, 0x8ABA3CBB,
0x28517711, 0xC20AD9F8, 0xABCC5167, 0xCCAD925F,
0x4DE81751, 0x3830DC8E, 0x379D5862, 0x9320F991,
0xEA7A90C2, 0xFB3E7BCE, 0x5121CE64, 0x774FBE32,
0xA8B6E37E, 0xC3293D46, 0x48DE5369, 0x6413E680,
0xA2AE0810, 0xDD6DB224, 0x69852DFD, 0x09072166,
0xB39A460A, 0x6445C0DD, 0x586CDECF, 0x1C20C8AE,
0x5BBEF7DD, 0x1B588D40, 0xCCD2017F, 0x6BB4E3BB,
0xDDA26A7E, 0x3A59FF45, 0x3E350A44, 0xBCB4CDD5,
0x72EACEA8, 0xFA6484BB, 0x8D6612AE, 0xBF3C6F47,
0xD29BE463, 0x542F5D9E, 0xAEC2771B, 0xF64E6370,
0x740E0D8D, 0xE75B1357, 0xF8721671, 0xAF537D5D,
0x4040CB08, 0x4EB4E2CC, 0x34D2466A, 0x0115AF84,
0xE1B00428, 0x95983A1D, 0x06B89FB4, 0xCE6EA048,
0x6F3F3B82, 0x3520AB82, 0x011A1D4B, 0x277227F8,
0x611560B1, 0xE7933FDC, 0xBB3A792B, 0x344525BD,
0xA08839E1, 0x51CE794B, 0x2F32C9B7, 0xA01FBAC9,

```
            0xE01CC87E, 0xBCC7D1F6, 0xCF0111C3, 0xA1E8AAC7,
            0x1A908749, 0xD44FBD9A, 0xD0DADECB, 0xD50ADA38,
            0x0339C32A, 0xC6913667, 0x8DF9317C, 0xE0B12B4F,
            0xF79E59B7, 0x43F5BB3A, 0xF2D519FF, 0x27D9459C,
            0xBF97222C, 0x15E6FC2A, 0x0F91FC71, 0x9B941525,
            0xFAE59361, 0xCEB69CEB, 0xC2A86459, 0x12BAA8D1,
            0xB6C1075E, 0xE3056A0C, 0x10D25065, 0xCB03A442,
            0xE0EC6E0E, 0x1698DB3B, 0x4C98A0BE, 0x3278E964,
            0x9F1F9532, 0xE0D392DF, 0xD3A0342B, 0x8971F21E,
            0x1B0A7441, 0x4BA3348C, 0xC5BE7120, 0xC37632D8,
            0xDF359F8D, 0x9B992F2E, 0xE60B6F47, 0x0FE3F11D,
            0xE54CDA54, 0x1EDAD891, 0xCE6279CF, 0xCD3E7E6F,
            0x1618B166, 0xFD2C1D05, 0x848FD2C5, 0xF6FB2299,
            0xF523F357, 0xA6327623, 0x93A83531, 0x56CCCD02,
            0xACF08162, 0x5A75EBB5, 0x6E163697, 0x88D273CC,
            0xDE966292, 0x81B949D0, 0x4C50901B, 0x71C65614,
            0xE6C6C7BD, 0x327A140A, 0x45E1D006, 0xC3F27B9A,
            0xC9AA53FD, 0x62A80F00, 0xBB25BFE2, 0x35BDD2F6,
            0x71126905, 0xB2040222, 0xB6CBCF7C, 0xCD769C2B,
            0x53113EC0, 0x1640E3D3, 0x38ABBD60, 0x2547ADF0,
            0xBA38209C, 0xF746CE76, 0x77AFA1C5, 0x20756060,
            0x85CBFE4E, 0x8AE88DD8, 0x7AAAF9B0, 0x4CF9AA7E,
            0x1948C25C, 0x02FB8A8C, 0x01C36AE4, 0xD6EBE1F9,
            0x90D4F869, 0xA65CDEA0, 0x3F09252D, 0xC208E69F,
            0xB74E6132, 0xCE77E25B, 0x578FDFE3, 0x3AC372E6};

    lengthp = p.length; // length of P-Array (hex array)
    lengths0 = s0.length; // length of S0-Array (hex  array)
    lengths1 = s1.length; //length of S1-Array (hex array)
    lengths2 = s2.length; // length of S2-Array (hex array)
    lengths3 = s3.length; //length of S3=Array(hex array)

pArray = new String[lengthp]; // new P-array in binary format
```

```
        s0Array = new String[lengths0];// new S0-array in binary format
        s1Array = new String[lengths1]; // new S1- array in binary format
        s2Array = new String[lengths2];// new S2-array in binary format
        s3Array = new String[lengths3];// new S3-array in binary format

        pArray = Binaryconv(p,lengthp);//Called the conversion method from hexa decimal to
Binary
        s0Array= Binaryconv(s0,lengths0);
        s1Array=Binaryconv(s1,lengths1);
        s2Array=Binaryconv(s2,lengths2);
        s3Array=Binaryconv(s3,lengths3);

//**************************************************************************
*
        System.out.println("p array in binary is ");
        for(int i=0;i<lengthp;i++)
        System.out.println(pArray[i]);
        //
//**************************************************************************
*
        // System.out.println("S0 array in binary is ");
        // for(int i=0;i<lengths0;i++)
        //  System.out.println(s0Array[i]);
        // //
//**************************************************************************
**
        // System.out.println("S1 array in binary is ");
        // for(int i=0;i<lengths1;i++)
        //  System.out.println(s1Array[i]);
        // //
//**************************************************************************
****
        // System.out.println("S2 array in binary is ");
```

```
        // for(int i=0;i<lengths2;i++)
        //  System.out.println(s2Array[i]);
        // //
//*************************************************************************
*****
        // System.out.println("S3 array in binary is ");
        // for(int i=0;i<lengths3;i++)
        //  System.out.println(s3Array[i]);
        //
//*************************************************************************
*****
        int c=0;
           Scanner sc= new Scanner(System.in);
           String XL=new String(); // 1st part of plaintext(32bits)
           String XR=new String();// 2nd part of plaintext(32bits)

           String XL1=new String();//1st part of ciphertext(32bits) for Decryption
           String XR1=new String();//2nd part of cipertext(32bits) for Encryption

           System.out.println("Enter the plain text \n");
           plaintext = sc.nextLine();
           System.out.println("plaintext is \n"+plaintext);
           byte[] bytespt = plaintext.getBytes(); // plaintext converted to bytes
           StringBuilder binarypt = new StringBuilder();
           for (byte b : bytespt)
           {
              c++;
                    int val = b;
                    for (int i = 0; i < 8; i++)
                    {
                    binarypt.append((val & 128) == 0 ? 0 : 1); //appended with 8 bits each
                    val <<= 1;
                    }
```

```
        }
        System.out.println("Binary of plaintext is :"+binarypt);

        XL = binarypt.substring(0,32);
        XR = binarypt.substring(32,64);

        String pt = XL+XR;
        System.out.println("XL is "+XL);
        System.out.println("XR is "+XR);

        String CipherText = encryption(XL,XR);

        XL1 = CipherText.substring(0,32);
        XR1 = CipherText.substring(32,64);

        System.out.println("XL is "+XL1);
        System.out.println("XR is "+XR1);

        String PlainText = decryption(XL1,XR1);

        if(pt.equals(PlainText))
        {
           System.out.println("Sucessfull");
        }
        else
        {
           System.out.println("Unsucessfull");
        }
}
public static String encryption(String XL,String XR)
{
   String L,R,NewL="",NewR="";
   L=XL;
```

```
        R=XR;
        for(int i=0;i<16;i++)
        {
            System.out.println("*****************
ROUND"+i+"************************");
            NewL = XOR(L,pArray[i]);
            System.out.println("_____________________________________________________");
            System.out.println("XL XOR P["+(i+1)+"] Is :" +NewL);
            String templ = function(NewL);
            NewR = XOR(templ,R);
            System.out.println("_____________________________________________________");
            System.out.println("XR After XOR(XL)"+(i+1)+" Is :" +NewR);

            L=NewR;
            R=NewL;
            System.out.println("_____________________________________________________");
            System.out.println("XL After Swapping is :"+L);
            System.out.println("XR After Swapping is :"+R);
        }
            L=NewL;
            R=NewR; // Undo last Swap
            System.out.println("_____________________________________________________");
            System.out.println("XL After Final Swapping is :"+L);
            System.out.println("XR After Final Swapping is :"+R);
            System.out.println("_____________________________________________________");

            System.out.println("XL XOR P[17] Final");
            String finalXL = XOR(NewL,pArray[17]);

            System.out.println("XR XOR P[16] Final");
            String finalXR = XOR(NewR,pArray[16]);

            String cipher = finalXL+finalXR;
```

```
        System.out.println("FINAL CIPHER IS: "+cipher);
        return cipher;
    }
    public static String decryption(String XL1,String XR1)
    {
        String L,R,NewL="",NewR="";
        L=XL1;
        R=XR1;
        System.out.println("********************** DECRYPTION
**************************");
        for(int i=17;i>=2;i--)
        {
            System.out.println("*****************
ROUND"+i+"************************");
            NewL = XOR(L,pArray[i]);
            System.out.println("_____________________________________________________");
            System.out.println("XL XOR P["+(i+1)+"] "+(i+1)+" Is :" +NewL);
            String templ = function(NewL);
            NewR = XOR(templ,R);
            System.out.println("_____________________________________________________");
            System.out.println("XR After XOR(XL)"+(i+1)+" Is :" +NewR);

            L=NewR;
            R=NewL;
            System.out.println("_____________________________________________________");
            System.out.println("XL After Swapping is :"+L);
            System.out.println("XR After Swapping is :"+R);
        }
            L=NewL;
            R=NewR;
            System.out.println("_____________________________________________________");
            System.out.println("XL After Final Swapping is :"+L);
            System.out.println("XR After Final Swapping is :"+R);
```

```
        System.out.println("_____");

        System.out.println("XL XOR P[18] Final");
        String finalXL = XOR(NewL,pArray[0]);

        System.out.println("XR XOR P[17] Final");
        String finalXR = XOR(NewR,pArray[1]);

        String plaintext1 = finalXL+finalXR;
        System.out.println("FINAL PlainText IS: "+plaintext1);
        return plaintext1;
}
public static String function(String XL)
{
   String xl1,xl2,xl3,xl4;

   xl1=XL.substring(0,8);
   xl2=XL.substring(8,16);
   xl3=XL.substring(16,24);
   xl4=XL.substring(24,32);

   System.out.println("_____");
   System.out.println("X1 "+xl1 +"|X2 "+xl2+"|X3 "+xl3+"|X4 "+xl4);

   String xl11=keyExp(xl1,s0Array); // For key Expansion from 8 bits to 32 bits
   String xl21=keyExp(xl2,s1Array);
   String xl31=keyExp(xl3,s2Array);
   String xl41=keyExp(xl4,s3Array);

   String tempxor = AddMod(xl11,xl21);
   String tempxor1 = XOR(tempxor,xl31);
   String tempxor2 = AddMod(tempxor1,xl41);
   System.out.println("_____");
```

```
        System.out.println("XL After Function is "+ tempxor2);
        return tempxor2;
    }

    public static String XOR(String Value1, String Value2)
    {
        String SB1="";
        for(int j=0; j<32;j++)
        {
            SB1 =
SB1+(Integer.parseInt(""+Value1.charAt(j))^Integer.parseInt(""+Value2.charAt(j))); // "" used
so that character gets appended with ""
        }
        return SB1;
    }

    public static String AddMod(String Value1, String Value2)
    {
        String result = "";
            int s = 0;// carry
            for(int i = 31; i>=0;i--)
            {
                int
add=Integer.parseInt(""+Value1.charAt(i))+Integer.parseInt(""+Value2.charAt(i))+s; // "" used
so that character gets appended with ""
            s=add/2;
            if(add>=2)
            add=add-2;
        result=String.valueOf(add)+result;
            }
                return result;
            }
    public static String keyExp(String xleft,String sarray[])
```

```
    {
        int keyfind = Integer.parseInt(xleft,2); //decimal value for the 8 bit left

        String index = sarray[keyfind]; // finding the corresponding value index
        return index;

    }

    public static String[] Binaryconv(int conv[], int len)
    {
        String[] convArray=new String[len];
        for(int i=0;i<len;i++)
            convArray[i]=String.format("%32s", Integer.toBinaryString(conv[i])).replace(' ','0'); //
replace Each empty place from right with 0

        return convArray;
    }
}
/*
Output:-E:\TE\VI\CSS> javac blowfish.java
E:\TE\VI\CSS> java blowfish
p array in binary is
00100100001111110110101010001000
10000101101000110000100011010011
00010011000110011000101000101110
00000011011100000111001101000100
10100100000010010011100000100010
00101001100111110011000111010000
00001000001011101111101010011000
11101100010011100110110010001001
01000101001010000010000111100110
00111000110100000001001101110111
10111110010101000110011011001111
```

00110100111010010000110001101100
11000000101011000010100110110111
11001001011111000101000011011101
00111111100001001101010110110101
10110101010001110000100100010111
10010010000101101101010111011001
10001001011110011111101100011011
Enter the plain text

mornings
plaintext is
mornings
Binary of plaintext is
:011011010110111101110010011011100110100101101110011001110110011
XL is 01101101011011110111001001101110
XR is 01101001011011100110011101110011
********** ROUND0**********

XL XOR P[1] Is :010010010101000000011000111001 10

X1 01001001|X2 01010000|X3 00011000|X4 11100110

XL After Function is 00111111000000110001000000101011

XR After XOR(XL)1 Is :010101100110110101110111010110 00

XL After Swapping is :010101100110110101110111010110 00
XR After Swapping is :010010010101000000011000111001 10
********** ROUND1**********

XL XOR P[2] Is :110100111100111001111111100010 11

X1 11010011|X2 11001110|X3 01111111|X4 10001011

XL After Function is 01101101100000000110011001101110

XR After XOR(XL)2 Is :00100100110100000111111010001000

XL After Swapping is :00100100110100000111111010001000

XR After Swapping is :11010011110011100111111110001011

********** ROUND2**********

XL XOR P[3] Is :00110111110010011111010010100110

X1 00110111|X2 11001001|X3 11110100|X4 10100110

XL After Function is 10010100010010011101111011101110

XR After XOR(XL)3 Is :01000111100001111010000101100101

XL After Swapping is :01000111100001111010000101100101

XR After Swapping is :00110111110010011111010010100110

********** ROUND3**********

XL XOR P[4] Is :01000100111101111101001000100001

X1 01000100|X2 11110111|X3 11010010|X4 00100001

XL After Function is 11101010001101100111111001010000

XR After XOR(XL)4 Is :11011101111111111000101011110110

XL After Swapping is :11011101111111111000101011110110

XR After Swapping is :01000100111101111101001000100001

********** ROUND4**********

XL XOR P[5] Is :01111001111101101011001011010100

X1 01111001|X2 11110110|X3 10110010|X4 11010100

XL After Function is 11111110101010010100001001111000

XR After XOR(XL)5 Is :10111010010111101001000001011001

XL After Swapping is :10111010010111101001000001011001

XR After Swapping is :01111001111101101011001011010100

********** ROUND5**********

XL XOR P[6] Is :10010011110000011010000110001001

X1 10010011|X2 11000001|X3 10100001|X4 10001001

XL After Function is 10110100010100101010110011010100

XR After XOR(XL)6 Is :11001101101001000001111000000000

XL After Swapping is :11001101101001000001111000000000

XR After Swapping is :10010011110000011010000110001001

********** ROUND6**********

XL XOR P[7] Is :11000101100010101110010010011000

X1 11000101|X2 10001010|X3 11100100|X4 10011000

XL After Function is 00111001001101100001000010001011

XR After XOR(XL)7 Is :10101010111101111011000100000010

XL After Swapping is :10101010111101111011000100000010

XR After Swapping is :11000101100010101110010010011000
********** ROUND7**********

XL XOR P[8] Is :01000110101110011101110110001011

X1 01000110|X2 10111001|X3 11011101|X4 10001011

XL After Function is 10100011100000111101011000101011

XR After XOR(XL)8 Is :01100110000010010011001010110011

XL After Swapping is :01100110000010010011001010110011
XR After Swapping is :01000110101110011101110110001011
********** ROUND8**********

XL XOR P[9] Is :00100011001000010001001101010101

X1 00100011|X2 00100001|X3 00010011|X4 01010101

XL After Function is 11101101000001000011001110011000

XR After XOR(XL)9 Is :10101011101111011111011000010011

XL After Swapping is :10101011101111011111011000010011
XR After Swapping is :00100011001000010001001101010101
********** ROUND9**********

XL XOR P[10] Is :10010011011011011111110101100100

X1 10010011|X2 01101101|X3 11111101|X4 01100100

XL After Function is 11110000000001111101000100010000

XR After XOR(XL)10 Is :11010011001001101100001001000101

XL After Swapping is :11010011001001101100001001000101

XR After Swapping is :10010011011011011111110101100100

********** ROUND10**********

XL XOR P[11] Is :01101101011100101010010010001010

X1 01101101|X2 01110010|X3 10100100|X4 10001010

XL After Function is 10101100110101011100000111011100

XR After XOR(XL)11 Is :00111111101110000011110010111000

XL After Swapping is :00111111101110000011110010111000

XR After Swapping is :01101101011100101010010010001010

********** ROUND11**********

XL XOR P[12] Is :00001011010100010011000011010100

X1 00001011|X2 01010001|X3 00110000|X4 11010100

XL After Function is 10010001010011101010111000111100

XR After XOR(XL)12 Is :11111100001111000000101010110110

XL After Swapping is :11111100001111000000101010110110

XR After Swapping is :00001011010100010011000011010100

********** ROUND12**********

XL XOR P[13] Is :00111100100100000010001100000001

X1 00111100|X2 10010000|X3 00100011|X4 00000001

XL After Function is 00100101010100110010011111101001

XR After XOR(XL)13 Is :00101110000000100001011100111101

XL After Swapping is :00101110000000100001011100111101

XR After Swapping is :00111100100100000010001100000001

********** ROUND13**********

XL XOR P[14] Is :11100111011111100100011111100000

X1 11100111|X2 01111110|X3 01000111|X4 11100000

XL After Function is 01010101111000100100001010100101

XR After XOR(XL)14 Is :01101001011100100110000110100100

XL After Swapping is :01101001011100100110000110100100

XR After Swapping is :11100111011111100100011111100000

********** ROUND14**********

XL XOR P[15] Is :01010110111101101011010000010001

X1 01010110|X2 11110110|X3 10110100|X4 00010001

XL After Function is 00101111100100011010000110000011

XR After XOR(XL)15 Is :11001000111011111110011001100011

XL After Swapping is :11001000111011111110011001100011

XR After Swapping is :01010110111101101011010000010001

********** ROUND15**********

XL XOR P[16] Is :01111101101010001110111101110100

X1 01111101|X2 10101000|X3 11101111|X4 01110100

XL After Function is 01000000110110010001000010010110

XR After XOR(XL)16 Is :00010110001011111010010010000111

XL After Swapping is :00010110001011111010010010000111

XR After Swapping is :01111101101010001110111101110100

XL After Final Swapping is :01111101101010001110111101110100

XR After Final Swapping is :00010110001011111010010010000111

XL XOR P[17] Final

XR XOR P[16] Final

FINAL CIPHER IS:

11110100110100010001010001101111100001000011100101110001010111 10

XL is 11110100110100010001010001101111

XR is 10000100001110010111000101011110

********** DECRYPTION **********

********** ROUND17**********

XL XOR P[18] 18 Is :01111101101010001110111101110100

X1 01111101|X2 10101000|X3 11101111|X4 01110100

XL After Function is 01000000110110010001000010010110

XR After XOR(XL)18 Is :11000100111000000110000111001000

XL After Swapping is :11000100111000000110000111001000

XR After Swapping is :01111101101010001110111101110100

********** ROUND16**********

XL XOR P[17] 17 Is :01010110111101101011010000010001

X1 01010110|X2 11110110|X3 10110100|X4 00010001

XL After Function is 00101111100100011010000110000011

XR After XOR(XL)17 Is :01010010001110010100111011110111

XL After Swapping is :01010010001110010100111011110111

XR After Swapping is :01010110111101101011010000010001

********** ROUND15**********

XL XOR P[16] 16 Is :11100111011111100100011111100000

X1 11100111|X2 01111110|X3 01000111|X4 11100000

XL After Function is 01010101111000100100001010100101

XR After XOR(XL)16 Is :00000011000101001111011010110100

XL After Swapping is :00000011000101001111011010110100

XR After Swapping is :11100111011111100100011111100000

********** ROUND14**********

XL XOR P[15] 15 Is :00111100100100000010001100000001

X1 00111100|X2 10010000|X3 00100011|X4 00000001

XL After Function is 00100101010100110010011111101001

XR After XOR(XL)15 Is :11000010001011010110000000001001

XL After Swapping is :11000010001011010110000000001001

XR After Swapping is :00111100100100000010001100000001

********** ROUND13**********

XL XOR P[14] 14 Is :00001011010100010011000011010100

X1 00001011|X2 01010001|X3 00110000|X4 11010100

XL After Function is 10010001010011101010111000111100

XR After XOR(XL)14 Is :10101101110111101000110100111101

XL After Swapping is :10101101110111101000110100111101

XR After Swapping is :00001011010100010011000011010100

********** ROUND12**********

XL XOR P[13] 13 Is :01101101011100101010010010001010

X1 01101101|X2 01110010|X3 10100100|X4 10001010

XL After Function is 10101100110101011100000111011100

XR After XOR(XL)13 Is :10100111100001001111000100001000

XL After Swapping is :10100111100001001111000100001000

XR After Swapping is :01101101011100101010010010001010

********** ROUND11**********

XL XOR P[12] 12 Is :10010011011011011111110101100100

X1 10010011|X2 01101101|X3 11111101|X4 01100100

XL After Function is 11110000000001111101000100010000

XR After XOR(XL)12 Is :10011101011101010111010110011010

XL After Swapping is :10011101011101010111010110011010

XR After Swapping is :10010011011011011111110101100100

********** ROUND10**********

XL XOR P[11] 11 Is :00100011001000010001001101010101

X1 00100011|X2 00100001|X3 00010011|X4 01010101

XL After Function is 11101101000001000011001110011000

XR After XOR(XL)11 Is :01111110011010011100111011111100

XL After Swapping is :01111110011010011100111011111100

XR After Swapping is :00100011001000010001001101010101

********** ROUND9**********

XL XOR P[10] 10 Is :01000110101110011101110110001011

X1 01000110|X2 10111001|X3 11011101|X4 10001011

XL After Function is 10100011100000111101011000101011

XR After XOR(XL)10 Is :10000000101000101100010101111110

XL After Swapping is :10000000101000101100010101111110

XR After Swapping is :01000110101110011101110110001011

********** ROUND8**********

XL XOR P[9] 9 Is :11000101100010101110010010011000

X1 11000101|X2 10001010|X3 11100100|X4 10011000

XL After Function is 00111001001101100001000010001011

XR After XOR(XL)9 Is :01111111100011111100110100000000

XL After Swapping is :01111111100011111100110100000000

XR After Swapping is :11000101100010101110010010011000

********** ROUND7**********

XL XOR P[8] 8 Is :10010011110000011010000110001001

X1 10010011|X2 11000001|X3 10100001|X4 10001001

XL After Function is 10110100010100101010110011010100

XR After XOR(XL)8 Is :01110001110110000100100001001100

XL After Swapping is :01110001110110000100100001001100

XR After Swapping is :10010011110000011010000110001001

********** ROUND6**********

XL XOR P[7] 7 Is :01111001111101101011001011010100

X1 01111001|X2 11110110|X3 10110010|X4 11010100

XL After Function is 11111110101010010100001001111000

XR After XOR(XL)7 Is :01101101011010001110001111110001

XL After Swapping is :01101101011010001110001111110001

XR After Swapping is :01111001111101101011001011010100

********** ROUND5**********

XL XOR P[6] 6 Is :01000100111101111101001000100001

X1 01000100|X2 11110111|X3 11010010|X4 00100001

XL After Function is 11101010001101100111111001010000

XR After XOR(XL)6 Is :10010011110000001100110010000100

XL After Swapping is :10010011110000001100110010000100
XR After Swapping is :01000100111101111101001000100001
********** ROUND4**********

XL XOR P[5] 5 Is :00110111110010011111010010100110

X1 00110111|X2 11001001|X3 11110100|X4 10100110

XL After Function is 10010100010010011101111011101110

XR After XOR(XL)5 Is :11010000101111100000110011001111

XL After Swapping is :11010000101111100000110011001111
XR After Swapping is :00110111110010011111010010100110
********** ROUND3**********

XL XOR P[4] 4 Is :11010011110011100111111110001011

X1 11010011|X2 11001110|X3 01111111|X4 10001011

XL After Function is 01101101100000000110011001101110

XR After XOR(XL)4 Is :01011010010010011001001011001000

_____

XL After Swapping is :01011010010010011001001011001000

XR After Swapping is :11010011110011100111111110001011

********** ROUND2**********

_____

XL XOR P[3] 3 Is :01001001010100000001100011100110

_____

X1 01001001|X2 01010000|X3 00011000|X4 11100110

_____

XL After Function is 00111111000000110001000000101011

_____

XR After XOR(XL)3 Is :11101100110011010110111110100000

_____

XL After Swapping is :11101100110011010110111110100000

XR After Swapping is :01001001010100000001100011100110

_____

XL After Final Swapping is :01001001010100000001100011100110

XR After Final Swapping is :11101100110011010110111110100000

_____

XL XOR P[18] Final

XR XOR P[17] Final

FINAL PlainText IS:

01101101011011110111001001101110011010010110111001100111011100 11

Successfull

*/

Conclusion :

I implemented the Blowfish algorithm which is a better data encryption than DES in terms of cryptographic security.