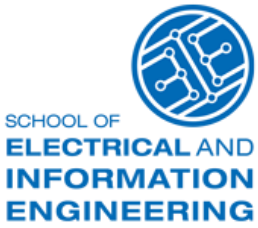
Laboratory: Data Encryption Algorithm

Benjamin Palay: 1815593

University of the Witwatersrand, Electrical (Information) Engineering YoS3, ELEN3015A

26th April 2021



Abstract

This report discusses and analyses the design and implementation of the Data Encryption Standard written in MATLAB. The algorithm accepts an input of 64 binary bits and a 56-bit key and outputs a 64-bit ciphertext block that has been encrypted by the algorithm. The decryption algorithm also takes in a 64-bit ciphertext and outputs the 64-bit plaintext. Both algorithms are successfully implemented.

1. Introduction

The objective of this laboratory is to gain practical exposure to the Data Encryption Standard (or DES) and to its associated algorithm (DEA – which is commonly referred to as DES). DES is a symmetric method of encryption, meaning that the same key is used to encrypt and decrypt a message [1]. It is a block cipher which encrypts 64-bit blocks of plaintext into 64-bit blocks of ciphertext [2]. The security of the system rests almost entirely on the 56-bit key used to encrypt the data [1]. A brief outline of the algorithm: the 64-bit input is permuted and split in half. There are then 16 rounds performed on the right half in which bits are manipulated, permuted, and combined with a 48-bit subkey that has been generated for each round. The halves are then switched, and the next round implemented. A more detailed outline is provided in the design section below.

The success criteria for this laboratory include obtaining the correct outputs for given inputs, gaining exposure to cryptographic techniques, as well as gaining skills in engineering and software. One of the constraints of this laboratory is that the coding must be done exclusively in MATLAB.

In this report, a background on DES is explained, followed by my design and implementation of the algorithm. All implementation is done using MATLAB. The results are shown, and an analysis and discussion of the efficiency and efficacy of the algorithm is considered.

1.1. Background

When the US government needed a standard way of encrypting sensitive information, the Data Encryption Standard was submitted by cryptographers at IBM in 1973[3]. For more than 20 years, it was also used to provide security for bank transactions, emails, login passwords and many other applications [3]. However, DES was proved to be inadequate for important systems mainly because of the relatively small key size [4]. Several challenges were held to determine how long it would take to break DES [5]. By 1999 it took less than 1 day to crack it [5]. Therefore, in 2002, the Advanced Encryption Standard (AES) replaced DES in the US government and other important areas of needed security [4]. AES can encrypt more information than DES and does so in a more complicated way, resulting in greater security[4].

Nonetheless, DES is an important precursor to AES and has been used as a basis for designing other important algorithms, such as hash function and pseudorandom number generators [3]. DES has 5 main modes, namely Electronic Code Book (ECB), Cipher Block Chaining (CBC), Cipher Feedback (CFB), Output Feedback (OFB), and Message Authentication Code (MAC) [3]. These are used when there is more than one 64-bit block of input data.

2. Design and Implementation

DES uses a 56-bit key to encrypt 64-bit blocks of plaintext data into 64-bit blocks of ciphertext. This key is often in a 64-bit form, but the parity bits are removed when an initial key permutation is performed, resulting in the desired 56-bit key. This implementation in MATLAB is shown in Appendix A. For each round of the algorithm, a different 48-bit subkey is generated from the 56-bit key. This is done by splitting the key into two 28-bit halves and left-shifting the bits in each half either by one or two, depending on the round index. The halves are then joined, becoming the current key, and then they are permuted and compressed according to a permutation box. This implementation is shown in Appendix B.

After the 64-bit input block is split into two 32-bit halves, 16 rounds are performed on the right half. Before the rounds are executed, the expansion (E), substitution (S), and permutation (P) boxes need to be initialized, as shown in Appendix C1. For each round, indexed as 'i', the corresponding subkey is used. First, the left half of the input is made to equal the right half. The 32-bit right half is expanded and the bits are duplicated to give 48 bits according to the Expansion Permutation box/table. These 48-bits are combined with the round subkey (Ki) using a bitwise exclusive-or (XOR) and then split into eight 6-bit numbers, each of which correspond to an S-box. For each 6-bit number, the first and last bits give the row in decimal format, while bits 2-5 give the column in decimal format. The decimal number in the corresponding position in the given S-box is used to convert the 6 bits into 4-bits, which are then combined to return a 32-bit number. This number is permuted according to the P-box and then combined with the left half using a bitwise exclusive-or. The round implementation is shown in Appendix C2.

Finally, after the 16 rounds, the halves are switched, combined and permuted according to the final permutation box. The output of this permutation is the ciphertext. The encryption function with this final implementation is shown in Appendix D.

The decryption algorithm is the same as the encryption algorithm, except that the subkeys are in the reverse order. This is achieved by shifting them right either 0, 1 or 2 times depending on the round index. The subkey generating function and the decryption algorithm are shown in Appendix E and F respectively.

One of the disadvantages of DES is that because of the way the key is transformed to obtain a subkey for each round, certain initial keys are called 'weak', meaning that the same subkey is used for some or all of the rounds of the algorithm, making it less secure [1]. For example, when the key '1F1F 1F1F 0E0E 0E0E' is used, all the subkeys generated are identical (as obtained from my code the subkey is: '000000000000000000000000111111111111111111111111'). This key is therefore called a weak key. When the key '1FFE1FFE0EFE0EFE' is used, only 2 subkeys are generated, making it a semi-weak key. Using the key '1FFEFE1F0EFEFE0E' generates 4 subkeys, making it a possibly weak key.

Nonetheless, these weak keys do not significantly affect the design and implementation of DES, as 64 weak keys out of a total 2^{56} possible keys is negligible, and one could check if the randomly generated key is weak or not before utilising it [1].

3. Results

The implementation of the designed algorithm is successful when compared with an online DES calculator [6] for several inputs. Another way to check the effectiveness is to run the decryption algorithm for the obtained ciphertext using the same key and comparing it to the original plaintext.

For example, running the following in the command window:

>>key = 0x0123456789ABCDEF;

>>A = 0xADFE123423453456;

>>key = subkey(key); %removing parity bits

>>ciphertext = encryption(key,A)

Yields:

ciphertext = '10110000110001111111010011000111011101001001000001100101000100011'

which in hexadecimal format is B0C7F4C774906513. This is the same ciphertext obtained by the online DES calculator.

When running the decryption algorithm with the same key and A = 0xB0C7F4C774906513 (the ciphertext), the output is plaintext='10101101111111100001001000110100001000110100010100110100010100110' which is ADFE123423453456 in hexadecimal. Notice how the plaintext obtained is identical to the original plaintext. Therefore, there is a high confidence that the algorithms were designed and implemented successfully.

Several more inputs and obtained outputs are shown in Appendix G below. Each of these were compared with the known calculator as well as decrypted and compared to the original plaintext. They were all found to be correct.

4. Analysis and Discussion

The results obtained have been shown to meet the success criteria as outlined in section 1. For each of the four tests shown in this report, as well as for many others, the outputs are identical to those from the DES online calculator [6] for the same given inputs. By using the obtained ciphertext as the input to the decryption algorithm and checking that the output is identical to the original plaintext, the implementation is further confirmed. The output is very accurate for both the encryption and decryption functions, as they are always exactly identical to the expected results.

Nonetheless, this design and implementation could be improved significantly by decreasing the number of operations performed each time it is run. For example, the expansion, substitution, and permutation boxes could be initialised and stored elsewhere, instead of re-initialising them each time and using unnecessary operations. Furthermore, this algorithm works only for specific input sizes (64/56-bit keys and 64-bit inputs). It would not be effective for other sizes. Moreover, it cannot combine blocks for a more extended output.

5. Conclusion

The design and implementation of the Data Encryption Standard was successful. When given a 64-bit plaintext input and 56-bit key, the encryption algorithm produces a 64-bit ciphertext. When the decryption algorithm is run using this ciphertext and the same key, the output is the original 64-bit plaintext.

6. References

[1] Schneier, B., 1996. *Applied Cryptography: Protocols, Algorithms and Source Code in C*. John Wiley & Sons.

[2] GeeksforGeeks. 2021. *Data encryption standard (DES) | Set 1 - GeeksforGeeks*. [online] Available at: <https://www.geeksforgeeks.org/data-encryption-standard-des-set-1/> [Accessed 20 April 2021].

[3] Coppersmith, D., Holloway, C., Matyas, S. and Zunic, N., 1997. The data encryption standard. *Information Security Technical Report*, 2(2), pp.22-24.

[4] Craven, C., 2017. [online] Available at: <https://www.sdxcentral.com/security/definitions/what-is-data-encryption-standard-des> [Accessed 20 April 2021].

[5] Precisely. 2020. *AES vs. DES Encryption: Why AES has replaced DES, 3DES and TDEA*. [online] Available at: <https://www.precisely.com/blog/data-security/aes-vs-des-encryption-standard-3des-tdea> [Accessed 20 April 2021].

[6] Emvlab.org. 2021. *DES Calculator*. [online] Available at: <https://emvlab.org/descalc/> [Accessed 20 April 2021].

7. Appendices

Appendix A: Function for removing the parity bits from the 64-bit key, creating a 56-bit key

```
function K = subkey(key)
%creates 56 bit key from 64 bit key

key = dec2bin(key,64);

KeyPermuteTo56=[57,49,41,33,25,17,9,1,58,50,42,34,26,18,10,2,59,51,43,35,27,19,11,3,60,52,44,36,63,55,47,39,31,23,15,7,62,54,46,38,30,22,14,6,61,53,45,37,29,21,13,5,28,20,12,4];
key = key(KeyPermuteTo56);


L = [key(1:28)];  %split 56bit input in half
R = [key(29:56)];
K = [L,R];
K = bin2dec(K);

end
```

Appendix B: Function for generating the specific 48-bit subkey from the 56-bit key

```
function K = subkey56(key,index)
%subkey for 56 bit input to get 48 bit output

key = dec2bin(key,56);

[n m] = size(key);

L = [key(1:28)];  %split 56bit input in half
R = [key(29:56)];
K = [L,R];

for i=1:index
    if (i==1)|(i==2)|(i==9)|(i==16)  %shift left 1

    L = [L(2:28),L(1)];
    R = [R(2:28),R(1)];
    else                             %shift left 2
        L = [L(3:28),L(1:2)];
        R = [R(3:28),R(1:2)];
    end
end

K = [L,R];
%now need to compress and permutate
PermuteCompress =[14,17,11,24,1,5,3,28,15,6,21,10,23,19,12,4,26,8,16,7,27,20,13,2,41,52,31,37,47,55,30,40,51,45,33,48,44,49,39,56,34,53,46,42,50,36,29,32];

K=K(PermuteCompress);

end
```

Appendix C: Function for round implementation

C1) Splitting input into two halves and initializing the Expansion, Substitution and Permutation Boxes

```
function [LH, RH] = rounds(A,key,index)
%A is the 64-bit input block,  key is 56 bit input
%output is right hand and left hand 32bit blocks after 'index' rounds
A = dec2bin(A,64);
[n m] = size(A)

IP = [58, 50, 42, 34, 26, 18, 10, 2, 60, 52, 44, 36, 28, 20, 12, 4, 62, 54, 46, 38, 30, 22, 14, 6, 64, 56, 48, 40, 32, 24, 16, 8, 57, 49, 41, 33, 25, 17, 9, 1, 59, 51, 43, 35, 27, 19, 11, 3, 61, 53, 45, 37, 29, 21, 13, 5, 63, 55, 47, 39, 31, 23, 15, 7];

A = A(IP); %initial permutation

LH = [A(1:32)];
RH = [A(33:64)];

Expansion1=[32,1,2,3,4,5; 4,5,6,7,8,9; 8,9,10,11,12,13; 12,13,14,15,16,17; 16,17,18,19,20,21; 20,21,22,23,24,25; 24,25,26,27,28,29; 28,29,30,31,32,1];%for SBOX (need 8x6 array)
Expansion2=[32,1,2,3,4,5, 4,5,6,7,8,9, 8,9,10,11,12,13, 12,13,14,15,16,17, 16,17,18,19,20,21, 20,21,22,23,24,25, 24,25,26,27,28,29, 28,29,30,31,32,1];%for bitXOR (need 1x48 bit array)
%S-boxes
S1=[14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7; 0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8; 4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0; 15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13];
S2=[15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10; 3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5; 0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15; 13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9];
S3=[10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8; 13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1; 13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7; 1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12];
S4=[7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15; 13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9; 10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4; 3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14];
S5=[2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9; 14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6; 4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14; 11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3];
S6=[12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11; 10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8; 9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6; 4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13];
S7=[4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1; 13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6; 1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2; 6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12];
S8=[13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7; 1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2; 7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8; 2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11];
PBOX=[16,7,20,21,29,12,28,17,1,15,23,26,5,18,31,10,2,8,24,14,32,27,3,9,19,13,30,6,22,11,4,25];
FinalP=[40,8,48,16,56,24,64,32,39,7,47,15,55,23,63,31,38,6,46,14,54,22,62,30,37,5,45,13,53,21,61,29,36,4,44,12,52,20,60,28,35,3,43,11,51,19,59,27,34,2,42,10,50,18,58,26,33,1,41,9,49,17,57,25];
 RH1 = 0b00000000000000000000000000000000; %32 bit array to use to populate after Sboxes
 RH1= dec2bin(RH1,32);
```

C2) Loop for doing each round (expanding and substituting, permuting and combinations)

```
for i=1:index
        Ki = subkey56(key,i) ;%using other function to find subkey for given index (56 btkey)

 temp = LH;
 LH = RH;
 RHE1 = RH(Expansion1); %expand and duplicate to 48 bits, 8 blocks of 6 bits, used for SBOX
 RHE2 = RH(Expansion2); %expand and duplicate to 1x48 bits

  EightBySix=[1,2,3,4,5,6;7,8,9,10,11,12;13,14,15,16,17,18;19,20,21,22,23,24;25,26,27,28,29,30;31,32,33,34,35,36;37,38,39,40,41,42;43,44,45,46,47,48];
  LinearArray=[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32];
  RH=mod(RHE2+Ki(1,:),2);
  RH = RH(EightBySix);

for s=1:8
    row = (RH(s,1)*2)+(RH(s,6)*1) +1; %bit 1 and 6 of each 6 bit gives row in decimal
    col = (RH(s,2)*8)+(RH(s,3)*4)+(RH(s,4)*2)+(RH(s,5)*1)+1;%bits 2-5 give column in decimal
    if s==1
        S=S1;
    end
     if s==2
        S=S2;
     end
     if s==3
        S=S3;
     end
     if s==4
        S=S4;
     end
     if s==5
        S=S5;
     end
     if s==6
        S=S6;
     end
     if s==7
        S=S7;
     end
     if s==8
        S=S8;
     end
    Bl = S(row,col);
   Bl = dec2bin(Bl,4); %4 bits from SBOX
    [RH1(((s*4)-3):(s*4))] = [Bl(1:4)]; %add together to get 32 bit array
end

  RH=RH1(PBOX) ;%Pbox permutation
  Rl =mod(RH+temp(1,:),2) ; %Xor with L(n-1)
  Rl = sprintf('%d',Rl); %make binary representation
  RH=Rl;

end


end
```

Appendix D: Function for the encryption and half-switching

```
function ciphertext = encryption(key,A)

i=16; %so that the 'rounds' function below returns the final Left and Right sides after 16 rounds.

[LH, RH] = rounds(A,key,i);

% final switch
tempH = RH;
RH = LH;
LH = tempH;

% The final permutation block is given by:
FinalP=[40,8,48,16,56,24,64,32,39,7,47,15,55,23,63,31,38,6,46,14,54,22,62,30,37,5,45,13,53,21,61,29,36,4,44,12,52,20,60,28,35,3,43,11,51,19,59

T = [LH,RH]; %combining the two 32 bit sides into 1.
ciphertext = T(FinalP);

end
```

Appendix E: Function for generating the decryption subkey

```
function K = subkeyDecrypt(key,index)
  key = dec2bin(key,56); %for 56bit key

 [n m] = size(key);

 L = [key(1:28)];  %split 56bit input in half
 R = [key(29:56)];
 K = [L,R];

for i=1:index
 if (i==1)
     L =L;
     R=R;

 elseif ((i==2)|(i==9)|(i==16))
          L = [L(28),L(1:27)];
     R = [R(28),R(1:27)];%shift right 1

 %      if ((i==3)|(i==4)|(i==5)|(i==6)|(i==7)|(i==8)|(i==10)|(i==11)|(i==12)|(i==13)|(i==14)|(i==15))
  else  %shift right 2
           L = [L(27:28),L(1:26)];
     R = [R(27:28),R(1:26)];
     end
 end

 K = [L,R];
 %now need to compress and permutate
 PermuteCompress =[14,17,11,24,1,5,3,28,15,6,21,10,23,19,12,4,26,8,16,7,27,20,13,2,41,52,31,37,47,55,30,40,51,45,33,48,44,49,39,56,34,53,46,42,50,36,29,32];

 K=K(PermuteCompress);

 end
```

Appendix F: Function for the decrypting DES (the box initialisations are done elsewhere for convenience)

```
function plaintext = decryption(key,A)

A = dec2bin(A,64);
[n m] = size(A);

A = A(IP); %initial permutation

LH = [A(1:32)];
RH = [A(33:64)];

for i=1:16

     Ki = subkeyDecrypt(key,i) ;%using other function to find subkey for given index (56 btkey)

 temp = LH;
 LH = RH;
 RHE1 = RH(Expansion1); %expand and duplicate to 48 bits, 8 blocks of 6 bits, used for SBOX
 RHE2 = RH(Expansion2); %expand and duplicate to 1x48 bits
  EightBySix=[1,2,3,4,5,6;7,8,9,10,11,12;13,14,15,16,17,18;19,20,21,22,23,24;25,26,27,28,29,30;31,32,33,34,35,36;37,38,39,40,41,42;43,44,45,46,47,48];
  LinearArray=[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32];
  RH=mod(RHE2+Ki(1,:),2);
  RH = RH(EightBySix);


for s=1:8
    row = (RH(s,1)*2)+(RH(s,6)*1) +1; %bit 1 and 6 of each 6 bit gives row in decimal
    col = (RH(s,2)*8)+(RH(s,3)*4)+(RH(s,4)*2)+(RH(s,5)*1)+1;%bits 2-5 give column in decimal
    if s==1
        S=S1;
    end
     if s==2
        S=S2;
     end
     if s==3
        S=S3;
     end
     if s==4
        S=S4;
     end
     if s==5
        S=S5;
     end
      if s==6
        S=S6;
      end
      if s==7
        S=S7;
      end
      if s==8
        S=S8;
      end


    B1 = S(row,col);
    B1 = dec2bin(B1,4); %4 bits from SBOX
    [RH1(((s*4)-3):(s*4))] = [B1(1:4)]; %add together to get 32 bit array

end

  RH=RH1(PBOX) ;%Pbox permutation
  R1 =mod(RH+temp(1,:),2) ; %Xor with L(n-1)
  R1 = sprintf('%d',R1); %make binary representation
  RH=R1;

end
% final switch
tempH = RH;
RH = LH;
LH = tempH;

T = [LH,RH];
plaintext = T(FinalP);

end
```

Appendix G:

G1) function for testing the inputs and obtained outputs for 3 tests

```
function testingInputs

%%Test1:
i=16;
A = 0xDE3428900987345E;
%or A=0b1101111000110100001010001001000000001001100001110011010001011110
key = 0x067EFA234DDD3456;
key = subkey(key);
ciphertext1 = encryption(key,A)

A = 0b1010111101001111111110000010100111000000110010110001001010010011;
%This is the obtained ciphertext
key = 0x067EFA234DDD3456;
key = subkey(key);
plaintext1 = decryption(key,A)

%%Test2:
i=16;
A = 0b1010101100100011110011010100010111101111011110000111100001100101;
%or A=0xAB23CD45EF787865
key = 0b1000011010111011111110100010110001010110101110110111011110111011;
%or key =0x067EFA234DDD3456;
key = subkey(key);
ciphertext2 = encryption(key,A)

A = 0b1111111111110010000100100110001101100011101000111101000110110111;
key = 0b1000011010111011111110100010110001010110101110110111011110111011;
key = subkey(key);
plaintext2 = decryption(key,A)

%%Test3:
i=16;
A = 0b0010001111011110010001011111011010100010011100010100111111011001;
%or A=0x23DE45F6A2714FD9
key = 0x00034EF56782ADC2;
%or key =0b1101001110111101010110011110000010101011011000010
key = subkey(key);
ciphertext3 = encryption(key,A)

A = 0b1110100000111001011101110000100001111010101111110100101000001100;
key = 0x00034EF56782ADC2;
key = subkey(key);
plaintext3 = decryption(key,A)
```

G2) outputs shown in the command window from running 'testingInputs.m'

```
Command Window

ciphertext3 =

    '1110100000111001011101110000100001111010101111110100101000001100'

plaintext3 =

    '0010001111011110010001011111011010100010011100010100111111011001'

>> testingInputs

ciphertext1 =

    '1010111101001111111110000010100111000000110010110001001010010011'

plaintext1 =

    '1101111000110100001010001001000000001001100001110011010001011110'

ciphertext2 =

    '1111111111110010000100100110001101100011101000111101000110110111'

plaintext2 =

    '1010101100100011110011010100010111101111011110000111100001100101'

ciphertext3 =

    '1110100000111001011101110000100001111010101111110100101000001100'

plaintext3 =

    '0010001111011110010001011111011010100010011100010100111111011001'
```