**ECE 673 – ADVANCED CRYPTOGRAPHY**

**HOMEWORK 2**

Mehmet Sinan INCI 393784824

System spec;

Intel Core i7 4702MQ @ 2.2 GHz 16 GB 1666 MHz DDR3 RAM running Windows 8

Development Environment;

Matlab 2013b

Outline of the algorithm is as follows;

**Inputs** *P, plaintext, C cyphertext*

**Output** An *x* such that ***DESx(P)=C*** or failure

*Using the formula P(S)>= mt / N and m = t = r = 2k/3 we calculate the m(row count) as 1024. P(S) here being the probability of success*

1. Initialise *hellman table with random starting points and m(1024) rows*
2. While  *last6bits!*=*0* then add new element to the chain
   1. *When the last 6bits are zero, save the last cyphertext as EP until all rows are done*

*SP = K0 = Starting Point*

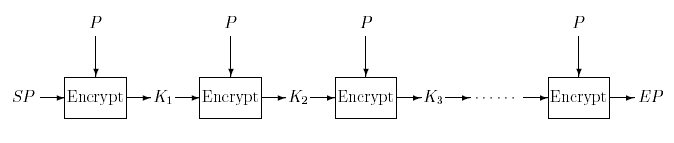
*K1 = E(P, SP)*

*K2 = E(P, K1)*

*:*

*:*

*EP = Kt = E(P, Kt−1) = End Point*



1. While  *last6bits!*=*0 then randkey=****DESrandkey(P)***
2. If *randkey* == *EPx* then ***secretkey* ∈ chainx**

Using the aforementioned algorithm and the spec, it took 55 minutes to create a table with 1024 rows. After creating the table I checked if randkeydp was equal to any of the EPs. The randkeydp was found after the creation of 4th hellman table. So, we can say that the total attack on DES for 30 bits took 4 hours and 1024\*2\*64=16 KB of memory with the distinguished points method.

**Cryptanalysis of DES 64(56bits) using Time-Memory Trade-off Implemented in Matlab**

**function [ output ] = reduct( input )**

% this reduction function gets rid of the parity bits in the cypertext

% and returns a 56 bit value

% it also turns the first 26 bits of the key into zeros since that is

% the scenario for the assignment

output(:,:)=input([1:7,9:15,17:23,25:31,33:39,41:47,49:55,57:63]);

output(1:26) = zeros;

end

**function [ K ] = addparity ( key )**

% this function calculates and adds the proper party bits to a 56bit

% key

K = reshape(key,7,8)'; % taking the transpose of the matrix

K(:,8) = mod(sum(K,2),2); % calculating the parity

% note these eight bits of key are never used in encryption

K = reshape(K',1,64);

end

**The DES Time Memory Trade-off Implementation**

% MSI TMTO implementation for DES 56

clc;

clear all; % clearing all the workspace variables

plain = round(rand(8,8)); % creating a random plaintext

plain = reshape(plain',1,64);

randkey = round(rand(8,7)); % creating a random key that will be our secret key

randkey(:,8) = mod(sum(randkey,2),2);

randkey = reshape(randkey',1,64);

hellman\_table = zeros(1024,1,64);

% the table that'll hold the starting points, SPs and ending points, EPs

display('creating the hellman table with random starting points');

for i=1:1024

key = round(rand(8,7));

key(:,8) = mod(sum(key,2),2);

key = reshape(key',1,64);

hellman\_table(i,1,:)=key; % first column of the table stores random keys

% each row corresponds to a new chain

end

display('table created!')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% ENDPOINT CALCULATION % ENDPOINT CALCULATION % ENDPOINT CALCULATION %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for j=1:1024 % using the formula P(S)>= mt / N we calculate the m(row count)

% P(S) here being the probability of success

j

encrypted = hellman\_table(j,:);

key56 = reduct(encrypted);

notzero=1;

% chainlength=0; % i was planning on storing the chainlenght for

% each chain as well but then i decided not to because the matlab

% time analysis showed that it slows down the process considerably

while notzero % when the last 6 bits are not zero,

% which indicates that the distinguished point has not been reached

% the operation continues

encrypted=DES(plain,'ENC',key56);

key56 = reduct(encrypted); % zeroing the first 26 bits and eliminating the parity

notzero = any(key56(51:56));

% chainlength=chainlength+1;

end

% sprintf('length of this chain is %d', chainlength);

hellman\_table(j+1,2,:)=addparity(key56); % adding parity bits when an EP is found

% and storing it in the table

end

display('hellman table calculation is done');

save('table'); % saving all the variables since creating the table takes about 1 hour

% and it's too risky to keep them in volatile memory

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Distinguished point search for the key

notzero=1;

chainlength=0;

encrypted=randkey;

randkey56 = reduct(encrypted);

while notzero % here we encrypt until we find the distinguished point for the known cyphertext

encrypted=DES(plain,'ENC',randkey56);

randkey56 = reduct(encrypted);

notzero = any(randkey56(51:56));

chainlength=chainlength+1;

end

randdp=addparity(randkey56);

chainlength

sorted\_table=sort(hellman\_table,2); % sort the table according to the EPs

match=0;

l=0; % l here is our row counter

while(~match) % when a match is found between EPs and the randkeyEP, it meaans

% that the secret key is in the l th

l=l+1;

match=isequal(sorted\_table(l,2,:),randdp(:));

end

display('match is found!');

**The DES Implementation**

function [varargout] = DES(input64,mode,key)

%DES: Data Encryption Standard

% Encrypt/Decrypt a 64-bit message using a 64-bit key using the Feistel Network

% -------------------------------------------------------------------------

% Inputs:

% input64 = a 64-bit message

% mode = either 'ENC' encryption or 'DEC' decryption (default 'ENC')

% key = a 56/64-bit key (optional under 'ENC', but mandatory under 'DEC')

% Outputs:

% varargout{1} = output64, a 64-bit message after encryption/decryption

% varargout{2} = a 64-bit key, if a 64-bit key is not provided as an input

% -------------------------------------------------------------------------

% Demos:

% plaintext = round(rand(1,64));

% [ciphertext,key] = DES(plaintext); % Encryption syntex 1

% [ciphertext1,key] = DES(plaintext,'ENC'); % Encryption syntex 2

% deciphertext1 = DES(ciphertext1,'DEC',key);% Decryption syntex

%

% key56 = round(rand(1,56));

% [ciphertext2,key64] = DES(plaintext,'ENC',key56);% Encryption syntex 3 (56-bit key)

% deciphertext2 = DES(ciphertext2,'DEC',key64); % Decryption syntex (64-bit key)

% ciphertext3 = DES(plaintext,'ENC',key64); % Encryption syntex 3 (64-bit key)

% deciphertext3 = DES(ciphertext3,'DEC',key56); % Decryption syntex (56-bit key)

%

% % plot results

% subplot(4,2,1),plot(plaintext),ylim([-.5,1.5]),xlim([1,64]),title('plaintext')

% subplot(4,2,2),plot(ciphertext),ylim([-.5,1.5]),xlim([1,64]),title('ciphertext')

% subplot(4,2,3),plot(deciphertext1),ylim([-.5,1.5]),xlim([1,64]),title('deciphertext1')

% subplot(4,2,4),plot(ciphertext1),ylim([-.5,1.5]),xlim([1,64]),title('ciphertext1')

% subplot(4,2,5),plot(deciphertext2),ylim([-.5,1.5]),xlim([1,64]),title('deciphertext2')

% subplot(4,2,6),plot(ciphertext2),ylim([-.5,1.5]),xlim([1,64]),title('ciphertext2')

% subplot(4,2,7),plot(deciphertext3),ylim([-.5,1.5]),xlim([1,64]),title('deciphertext3')

% subplot(4,2,8),plot(ciphertext3),ylim([-.5,1.5]),xlim([1,64]),title('ciphertext3')

% -------------------------------------------------------------------------

% NOTE:

% 1. If a 64-bit key is provided, then its bit parities will be checked. If

% a 56-bit key is provided, then it is automatically added 8 partity

% checking bits. However, the 8 parity bits are never used in

% DES encryption/decryption process. They are included just for the

% completeness of a DES implementation.

% 2. Cipher modes are not provided in this simple script. If you are

% interested or do not know what does cipher modes mean, please go to page

% http://en.wikipedia.org/wiki/Block\_cipher\_modes\_of\_operation

% for details. Please keep in mind that selecting an inappropriate working

% mode may extremely weaken the security of your messages.

% 3. A general description of DES can be found at its wiki page:

% http://en.wikipedia.org/wiki/Data\_Encryption\_Standard

% The detailed cryptographical primitives can be found under the page:

% http://en.wikipedia.org/wiki/DES\_supplementary\_material

% If you want to speed-up the DES code here, you can simply store these

% primitives in memory and call them when you need.

% -------------------------------------------------------------------------

% By Yue (Rex) Wu

% ECE Dept @ Tufts Univ.

% 08/18/2012

% If you find bugs, please email me via ywu03@ece.tufts.edu

% -------------------------------------------------------------------------

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% 0. Initialization %%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% 0.1 check input

error(nargchk(1,3,nargin));

switch nargin

case 1

mode = 'ENC';

K = round(rand(8,7));

K(:,8) = mod(sum(K,2),2); % note these eight bits of key are never used in encryption

K = reshape(K',1,64);

varargout{2} = K;

case 2

switch mode

case 'ENC'

K = round(rand(8,7));

K(:,8) = mod(sum(K,2),2); % note these eight bits of key are never used in encryption

K = reshape(K',1,64);

varargout{2} = K;

case 'DEC'

error('Key has to be provided in decryption mode (DEC)')

otherwise

error('WRONG working mode!!! Select either encrtyption mode: ENC or decryption mode: DEC !!!')

end

case 3

if isempty(setdiff(unique(key),[0,1])) % check provided key type

if numel(key) == 64 % check provided key parity

keyParityCheck = @(k) (sum(mod(sum(reshape(k,8,8)),2))==0);

if keyParityCheck(key) == 1

K = key(:)';

else

error('Key parity check FAILED!!!')

end

elseif numel(key) == 56 % add parity bits

K = reshape(key,7,8)';

K(:,8) = mod(sum(K,2),2); % note these eight bits of key are never used in encryption

K = reshape(K',1,64);

varargout{2} = K;

% display('Key parity bits added')

else

error('Key has to be either 56 or 64-bit long!!!')

end

else

error('Key has to be binary!!!')

end

end

% 0.2 check message length and type

if numel(input64) == 64 && isempty(setdiff(unique(input64),[0,1]))

P = input64;

else

error('Message has to be a 64-bit message!!!')

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% 1. Cryptographical primitives %%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% 1.1 define splitting function

HALF\_L = @(message) message(1:32);

HALF\_R = @(message) message(33:64);

% 1.2 define expansion function

EF = @(halfMessage) [halfMessage([32,4:4:28])',(reshape(halfMessage,4,8))',halfMessage([5:4:29,1])'];

% 1.3 define key mixing (KM)

KM = @(expandedHalfMessage,rK) xor(expandedHalfMessage,reshape(rK,6,8)');

% 1.4 define eight substitution tables

% input: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

st{1} = [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];

st{2} = [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];

st{3} = [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];

st{4} = [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];

st{5} = [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];

st{6} = [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];

st{7} = [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];

st{8} = [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];

% the eight binary s-boxes

for i = 1:8

ST{i} = mat2cell(blkproc(st{i},[1,1],@(x) de2bi(x,4,'left-msb')),ones(1,4),ones(1,16)\*4);

end

% 1.5 define subsitution function (SBOX)

SUBS = @(expandedHalfMessage,blkNo) ST{blkNo}{bi2de(expandedHalfMessage(blkNo,[1,6]),'left-msb')+1,bi2de(expandedHalfMessage(blkNo,[2:5]),'left-msb')+1};

SBOX = @(expandedHalfMessage) [SUBS(expandedHalfMessage,1);SUBS(expandedHalfMessage,2);...

SUBS(expandedHalfMessage,3);SUBS(expandedHalfMessage,4);...

SUBS(expandedHalfMessage,5);SUBS(expandedHalfMessage,6);...

SUBS(expandedHalfMessage,7);SUBS(expandedHalfMessage,8)];

% 1.6 define permutation function (PBOX)

PBOX = @(halfMessage) halfMessage([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]);

% 1.7 define initial permutation (IP)

IP = @(message) message([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]);

% 1.8 define final permutation (FP)

FP = @(message) message([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]);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% 2. key schedule %%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% 2.1 define permuted choice 1 (PC1)

PC1L = @(key64) key64([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]);

PC1R = @(key64) key64([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]);

% 2.2 define permuted choice 2 (PC2)

PC2 = @(key56) key56([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]);

% 2.3 define rotations in key-schedule (RK)

% round# 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6

RK = [1 1 2 2 2 2 2 2 1 2 2 2 2 2 2 1];

% 2.4 define key shift function (KS)

KS = @(key28,s) [key28(s+1:end),key28(1:s)];

% 2.5 define sub-keys for each round

leftHKey = PC1L(K); % 28-bit half key

rightHKey = PC1R(K);% 28-bit half key

for i = 1:16

leftHKey = KS(leftHKey,RK(i));

rightHKey = KS(rightHKey,RK(i));

key56 = [leftHKey ,rightHKey];

subKeys(i,:) = PC2(key56(:));

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% 3. DES main loop %%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% 3.1 initial permutation

C = IP(P);

switch mode

case 'ENC' % if encryption, split 64 message to two halves

L{1} = HALF\_L(C); % left-half 32-bit

R{1} = HALF\_R(C); % right-half 32-bit

case 'DEC' % if decryption, swapping two halves

L{1} = HALF\_R(C);

R{1} = HALF\_L(C);

end

% 3.2 cipher round 1 to 16

for i = 1:16

L{i+1} = R{i}; % half key: 32-bit

expended\_R = EF(R{i}); % expended half key: 32-bit to 48-bit

switch mode

case 'ENC' % if encryption, apply sub-keys in the original order

mixed\_R = KM(expended\_R,subKeys(i,:)); % mixed with sub-key: 48-bit

case 'DEC' % if decryption, apply sub-keys in the reverse order

mixed\_R = KM(expended\_R,subKeys(16-i+1,:)); % mixed with sub-key: 48-bit

end

substituted\_R = SBOX(mixed\_R); % substitution: 48-bit to 32-bit

permuted\_R = PBOX(reshape(substituted\_R',1,32)); % permutation: 32-bit

R{i+1} = xor(L{i},permuted\_R); % Feistel function: 32-bit

end

% 3.3 final permutation

switch mode

case 'ENC'

C = [L{end},R{end}];

case 'DEC'

C = [R{end},L{end}];

end

output64 = FP(C);

varargout{1} = output64;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% END %%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%