SCHOOL OF ELECTRICAL ENGINEERING, UNIVERSITY OF THE WITSWATERSRAND

ELEN3015 - Data and Information Management

Lecturer: Ling Cheng

Lab 2 Data Encryption Algorithm

6 March 2018

Tyson Cross 1239448

QUESTION 2

Code 1: question2.m

```
% ELEN3015 Lab 2, Question 2

clc
clear all

% keys to analyse
key1 = '1F1F1F1F0E0E0E0E';
key2 = '1FFE1FFE0EFE0EFE';
key3 = '1FFEFE1F0EFEFE0E';

% convert to binary row vectors
key_bin1 = hex2binary(key1);
key_bin2 = hex2binary(key2);
key_bin3 = hex2binary(key3);

% initial permutation
key_string1 = permuter(key_bin1, 'parity');
key_string2 = permuter(key_bin2, 'parity');
key_string3 = permuter(key_bin3, 'parity');

% DEA rounds to generate all subkeys
for round_no=1:16
    key_schedule1(round_no,:) = generateSubKey(key_string1,round_no);
    key_schedule2(round_no,:) = generateSubKey(key_string2,round_no);
    key_schedule3(round_no,:) = generateSubKey(key_string3,round_no);
end

% count and classify the subkeys
[count1, classification1] = analyseSubKeys(key_schedule1);
[count2, classification2] =  analyseSubKeys(key_schedule2);
[count3, classification3] =  analyseSubKeys(key_schedule3);

% output
disp(['Key ', key1, ' has ', num2str(count1), ' unique subkey(s). It is a ', classification1, ' key'])
disp(['Key ', key2, ' has ', num2str(count2), ' unique subkey(s). It is a ', classification2, ' key'])
disp(['Key ', key3, ' has ', num2str(count3), ' unique subkey(s). It is a ', classification3, ' key'])
```

When question2.m is run in the workspace, the following output is displayed in the command window:

```
Key 1F1F1F1F0E0E0E0E has 1 unique subkey(s). It is a weak key
Key 1FFE1FFE0EFE0EFE has 2 unique subkey(s). It is a semi weak key
Key 1FFEFE1F0EFEFE0E has 4 unique subkey(s). It is a possibly weak key
```

QUESTION 3

Code 2: question3.m

```
% ELEN3015 Lab 2, Question 3

clc
clear all

% inputs
plaintext_str = '0000000100100011010001010110011111000100110101011110011011101111';
key_str       = '0001001100110100010101110111100110011011101111001101111111110001';
plaintext = plaintext_str - '0';
key_64 = key_str - '0';
key_56 = permuter(key_64, 'parity');                    % discard parity bits and permute
block = permuter(plaintext,'initial');                  % initial permutation

round_no = 1;
subkey = generateSubKey(key_56,round_no);

% perform a DEA round of encryption
[ L_block, R_block ] = DES(block, round_no, subkey);

% output
disp(['Input 64-bit message: ', binary2hex(plaintext),]);
disp(['Input 64-bit key:     ', binary2hex(key_64)])
disp(['Permuted 56-bit key:    ', binary2hex(key_56)])
disp(['48-bit round (', num2str(round_no),') subkey:  ', binary2hex(subkey)])
disp(['Permuted block: ', num2string(block)])
disp(['Left: ', num2string(L_block), '   Right: ', num2string(R_block)])
```

When question3.m is run in the workspace, the following output is displayed in the command window:

```
Input 64-bit message: 0123456789ABCDEF
Input 64-bit key:     133457799BBCDFF1
Permuted 56-bit key:    F0CCAAF556678F
48-bit round (1) subkey:  1B02EFFC7072
Permuted block: 1100110000000000110011001111111111110000101010101111000010101010
Left: 11110000101010101111000010101010   Right: 11101111010001010011001010100010
```

QUESTION 4

Code 3: question4.m

```
% ELEN3015 Lab 2, Question 4

clc
clear all

% inputs
plaintext_str = '0123456789ABCDEF';                         % test block
plaintext = hex2binary(plaintext_str);                      % convert to binary row vector
key_str = '0001001100110100010101110111100110011011101111001101111111110001';
key_64 = key_str - '0';                                     % convert to binary row vector
key_56 = permuter(key_64, 'parity');                        % discard parity bits and permute

%% encryption
block = permuter(plaintext,'initial');                      % initial permutation
for round_no = 1:16                                         % 16 rounds of DES
    subkey = generateSubKey(key_56,round_no);               % subkey generation for each round
    [ L, R ] = DES(block, round_no, subkey);                % DEA (encryption)
    block = [L R];                                          % rejoin the L and R halves
end
cipherblock = permuter(block, 'final');                     % final permutation


%% decryption
out_block = permuter(cipherblock,'initial');                % initial permutation
for round_no=1:16                                           % 16 rounds of DES
    subkey = generateSubKey(key_56,17-round_no);            % subkeys generated in reverse order
    [L, R] = DES(out_block, round_no, subkey);              % DEA (decrpytion)
    out_block = [L R];                                      % rejoin block halves
end
decrypted = permuter(out_block, 'final');                   % final permutation


% output and check
decrypted_str = binary2hex(decrypted);
cipher_str = binary2hex(cipherblock);
disp(['Input 64-bit key:     ', binary2hex(key_64)])
disp(['Encryted ciphertext:  ', cipher_str])
disp(['Original input text:  ', plaintext_str]);
disp(['Decrypted plaintext:  ', decrypted_str])
disp(' ')

% check
if isequal(decrypted_str,plaintext_str)
    disp('The decrypted block matches the plaintext block');
else
    warning('The decrypted block does NOT match the plaintext block');
end
```

When question4.m is run in the workspace, the following output is displayed in the command window:

```
Input 64-bit key:     133457799BBCDFF1
Encryted ciphertext:  85E813540F0AB405
Original input text:  0123456789ABCDEF
Decrypted plaintext:  0123456789ABCDEF

The decrypted block matches the plaintext block
```

APPENDIX I

DES IMPLEMENTATION CODE

Code 4: generateSubKey.m

```
function [ outKey ] = generateSubKey( inKey, i )
% generateSubKey() produces a DES-compliant 48-bit key from an input 56-bit key
% for a specific round number

if i>16 || i<1
    error('Round number not in valid range [1-16]')
end

if length(inKey)≠56
    error('Please provide a 56-bit key')
end

% Take the 56-bit key, split into 2 subkeys
% Each subkey half undergoes 16 rounds of subkey generation:
%    a) bitshift each half by (1 or 2) bits, on a fixed schedule
%    b) combine halves, compress down to a 48-bit key which is stored.

[keyL, keyR] =  splitter(inKey);
for round_no=1:i
keyL = shiftKey(keyL,round_no);
keyR = shiftKey(keyR,round_no);
end

outKey = permuter([keyL keyR],'compression');

end
```

Code 5: analyseSubKeys.m

```
function [count, classification] = analyseSubKeys( key_schedule )
% analyseSubKeys counts and classifies the strength of a subkey schedule

% determine number of unique keys
[n, m] = size(unique(key_schedule,'rows'));
count = n;

% classify the input keys
switch n
    case 1
        classification = 'weak';
    case 2
        classification = 'semi weak';
    case 4
        classification = 'possibly weak';
    otherwise
        classification = 'flat/linear';
end
end
```

Code 6: splitter.m

```
function [ outKey_l, outKey_r ] = splitter(inKey)
% splitter() splits an input row vector in half, into two seperate row vectors of equal length

if mod(length(inKey),2)
    error('Block length must be even')
else len = length(inKey)/2;
end

outKey_l = inKey(1:len);
outKey_r = inKey(len+1:end);
end
```

Code 7: shiftKey.m

```
function [ outKey ] = shiftKey( inKey, round_no)
% Left-shifts (circularly) key entries, 1 or 2 places, depending on the round

if (round_no==1 || round_no==2 || round_no==9 || round_no==16)
    shift = -1;
else shift = -2;
end

outKey = circshift(inKey',[shift,0])';
end
```

Code 8: permuter.m

```
function [ outBlock ] = permuter(inBlock, mode)
% Performs fixed permutation upon input block (64-/56-/48-bit)

% The permutation uses fixed D-tables and size of i/o block,
% determined by the mode argument:
% For blocks: 'initial'(64-bit), 'final'(64-bit), 'expansion' (32->48-bit) or 'pbox'(32-bit)
% For keys: 'parity' (64->56-bit) or 'compression' (56->48-bit)

switch mode
    case 'initial'
        len = 64;
        permutationTable = [58, 50, 42, 34, 26, 18, 10, 02, 60, 52, 44, 36, 28, 20, 12, 04,...
                            62, 54, 46, 38, 30, 22, 14, 06, 64, 56, 48, 40, 32, 24, 16, 08,...
                            57, 49, 41, 33, 25, 17, 09, 01, 59, 51, 43, 35, 27, 19, 11, 03,...
                            61, 53, 45, 37, 29, 21, 13, 05, 63, 55, 47, 39, 31, 23, 15, 07];
    case 'final'
        len = 64;
        permutationTable = [40, 08, 48, 16, 56, 24, 64, 32, 39, 07, 47, 15, 55, 23, 63, 31,...
                            38, 06, 46, 14, 54, 22, 62, 30, 37, 05, 45, 13, 53, 21, 61, 29,...
                            36, 04, 44, 12, 52, 20, 60, 28, 35, 03, 43, 11, 51, 19, 59, 27,...
                            34, 02, 42, 10, 50, 18, 58, 26, 33, 01, 41, 09, 49, 17, 57, 25];

    case 'parity'
        len = 56;
        permutationTable = [57, 49, 41, 33, 25, 17, 09, 01, 58, 50, 42, 34, 26, 18,...
                            10, 02, 59, 51, 43, 35, 27, 19, 11, 03, 60, 52, 44, 36,...
                            63, 55, 47, 39, 31, 23, 15, 07, 62, 54, 46, 38, 30, 22,...
                            14, 06, 61, 53, 45, 37, 29, 21, 13, 05, 28, 20, 12, 04];
    case 'compression'
        len = 48;
        permutationTable = [14, 17, 11, 24, 01, 05, 03, 28, 15, 06, 21, 10,...
                            23, 19, 12, 04, 26, 08, 16, 07, 27, 20, 13, 02,...
                            41, 52, 31, 37, 47, 55, 30, 40, 51, 45, 33, 48,...
                            44, 49, 39, 56, 34, 53, 46, 42, 50, 36, 29, 32];
     case 'expansion'
         len = 48;
        permutationTable = [32, 01, 02, 03, 04, 05, 04, 05, 06, 07, 08, 09,...
                            08, 09, 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, 01];
    case 'pbox'
        len = 32;
        permutationTable = [16, 07, 20, 21, 29, 12, 28, 17,...
                            01, 15, 23, 26, 05, 18, 31, 10,...
                            02, 08, 24, 14, 32, 27, 03, 09,...
                            19, 13, 30, 06, 22, 11, 04, 25];
    otherwise
        error('Specify permutation mode: ''initial'', ''final'', ''parity'', ''compression'', ...
            ''expansion'', or ''pbox''')
end

outBlock = zeros(1,len);

for iter=1:len
    outBlock(iter) = inBlock(permutationTable(iter));
end

end
```

Code 9: DES.m

```
function [ L, R ] = DES( inBlock, round_no, subKey)
% Performs encryption on an input 64-bit block, returns two encrypted 32-bit blocks

% if length(inBlock) < 64
%     inBlock = padarray(inBlock',64-length(inBlock),'post')';
% end

% Round
[L, R] = splitter(inBlock);                     % evenly split the block into two 32-bit halves

temp = R;                                       % store the L half value
R = permuter(R,'expansion');                    % perform the expansion permutation
R = xor(R, subKey);                             % XOR with the round subkey
R = substitution(R);                            % perform non-linear s-box substitution
R = permuter(R,'pbox');                         % perform the p-box permutation
R = xor(R,L);                                   % XOR with the original L half
L = temp;                                       % assign L to the output from the Feistel cipher
if round_no == 16                               % swap the L and R halves on the final round
    [R,L] = deal(L,R);
end

%% output
% disp(['Round ', num2str(round_no), ':'])
% disp(['L: ', num2string(L), '   R: ',num2string(R)])
end
```

Code 10: substitution.m

```
function [ outBlock ] = substitution( inBlock )
% Performs non-linear S-Box substitution on inBlock

substitutionTable =...
    [14, 04, 13, 01, 02, 15, 11, 08, 03, 10, 06, 12, 05, 09, 00, 07;...
     00, 15, 07, 04, 14, 02, 13, 01, 10, 06, 12, 11, 09, 05, 03, 08;...
     04, 01, 14, 08, 13, 06, 02, 11, 15, 12, 09, 07, 03, 10, 05, 00;...
     15, 12, 08, 02, 04, 09, 01, 07, 05, 11, 03, 14, 10, 00, 06, 13];

substitutionTable(:,:,2) =...
    [15, 01, 08, 14, 06, 11, 03, 04, 09, 07, 02, 13, 12, 00, 05, 10;...
     03, 13, 04, 07, 15, 02, 08, 14, 12, 00, 01, 10, 06, 09, 11, 05;...
     00, 14, 07, 11, 10, 04, 13, 01, 05, 08, 12, 06, 09, 03, 02, 15;...
     13, 08, 10, 01, 03, 15, 04, 02, 11, 06, 07, 12, 00, 05, 14, 09];

substitutionTable(:,:,3) =...
    [10, 00, 09, 14, 06, 03, 15, 05, 01, 13, 12, 07, 11, 04, 02, 08;...
     13, 07, 00, 09, 03, 04, 06, 10, 02, 08, 05, 14, 12, 11, 15, 01;...
     13, 06, 04, 09, 08, 15, 03, 00, 11, 01, 02, 12, 05, 10, 14, 07;...
     01, 10, 13, 00, 06, 09, 08, 07, 04, 15, 14, 03, 11, 05, 02, 12];

substitutionTable(:,:,4) =...
    [07, 13, 14, 03, 00, 06, 09, 10, 01, 02, 08, 05, 11, 12, 04, 15;...
     13, 08, 11, 05, 06, 15, 00, 03, 04, 07, 02, 12, 01, 10, 14, 09;...
     10, 06, 09, 00, 12, 11, 07, 13, 15, 01, 03, 14, 05, 02, 08, 04;...
     03, 15, 00, 06, 10, 01, 13, 08, 09, 04, 05, 11, 12, 07, 02, 14];

substitutionTable(:,:,5) =...
    [02, 12, 04, 01, 07, 10, 11, 06, 08, 05, 03, 15, 13, 00, 14, 09;...
     14, 11, 02, 12, 04, 07, 13, 01, 05, 00, 15, 10, 03, 09, 08, 06;...
     04, 02, 01, 11, 10, 13, 07, 08, 15, 09, 12, 05, 06, 03, 00, 14;...
     11, 08, 12, 07, 01, 14, 02, 13, 06, 15, 00, 09, 10, 04, 05, 03];

substitutionTable(:,:,6) =...
    [12, 01, 10, 15, 09, 02, 06, 08, 00, 13, 03, 04, 14, 07, 05, 11;...
     10, 15, 04, 02, 07, 12, 09, 05, 06, 01, 13, 14, 00, 11, 03, 08;...
     09, 14, 15, 05, 02, 08, 12, 03, 07, 00, 04, 10, 01, 13, 11, 06;...
     04, 03, 02, 12, 09, 05, 15, 10, 11, 14, 01, 07, 06, 00, 08, 13];

substitutionTable(:,:,7) =...
    [04, 11, 02, 14, 15, 00, 08, 13, 03, 12, 09, 07, 05, 10, 06, 01;...
     13, 00, 11, 07, 04, 09, 01, 10, 14, 03, 05, 12, 02, 15, 08, 06;...
     01, 04, 11, 13, 12, 03, 07, 14, 10, 15, 06, 08, 00, 05, 09, 02;...
     06, 11, 13, 08, 01, 04, 10, 07, 09, 05, 00, 15, 14, 02, 03, 12];

substitutionTable(:,:,8) =...
    [13, 02, 08, 04, 06, 15, 11, 01, 10, 09, 03, 14, 05, 00, 12, 07;...
     01, 15, 13, 08, 10, 03, 07, 04, 12, 05, 06, 11, 00, 14, 09, 02;...
     07, 11, 04, 01, 09, 12, 14, 02, 00, 06, 10, 13, 15, 03, 05, 08;...
     02, 01, 14, 07, 04, 10, 08, 13, 15, 12, 09, 00, 03, 05, 06, 11];

outBlock = [];
```

```
for i=1:8
    row_index_1 = num2string(inBlock(i*6-5));               % first bit of the 6-bit block
    row_index_2 = num2string(inBlock(i*6));                 % last bit of the 6-bit block
    row_index = strcat(row_index_1,row_index_2);            % convert the bits into an index
    row = bin2dec(row_index) + 1;                           % matlab indexing starts at 1, not 0
    col = bin2dec(dec2bin(inBlock(i*6-4:i*6-1))') + 1;      % middle four bits as index
    value = dec2bin(substitutionTable(row,col,i),4)-'0';    % apply the sunstitution table
    outBlock = [outBlock value];                            % append the bit-block
end
```

Code 11: addParityBits.m

```
function [ output ] = addParityBits( bin_array )
% addParityBits counts and adds parity bits to a 56-bit key, producing a 64-bit key

if length(bin_array)≠56
    error('Please provide a 56-bit key')
end

bin_matrix = reshape(bin_array,[7,8])';
bin_matrix = [bin_matrix zeros(8,1)];

for i=1:8
    sum=0;
    for j=1:7
        sum = sum + bin_matrix(i,j);
    end
    bin_matrix(i,8)= 1-mod(sum, 2);
end

output = reshape(bin_matrix,1,64);

end
```