CRYPTOGRAPHY LAB

Semester IV

Practical No. 3

Name : Saloni Vishwakarma

Batch-Roll no : C1-13

Analysis of Prac 3 Code:- The Bold region indicates the code explanation /*

* Data Encryption Standard

* An approach to DES algorithm

*

* By: Daniel Huertas Gonzalez

* * Version: 0.1

*

* Based on the document FIPS PUB 46-3

*/

#include <stdio.h>

#include <stdlib.h>

#include <stdint.h> /***The "stdint.h" header file provides a way to declare these integer types with guaranteed sizes, such as int8_t (8-bit integer), int16_t (16-bit integer), int32_t (32-bit integer), and int64_t (64-bit integer)** */

#define LB32_MASK 0x00000001

#define LB64_MASK 0x0000000000000001 ***/In this case, LB32_MASK and LB64_MASK are defined as macros that represent bit masks used for bitwise operations. Similarly, 0x0000000000000001 represents a 64-bit binary number with only the least significant bit set to 1.These masks can be used to perform bitwise operations on specific bits of a binary number. This will perform a bitwise AND operation between num and LB32_MASK, resulting in a binary number where all the bits except the least significant bit are set to 0/***

#define L64_MASK 0x00000000ffffffff

#define H64_MASK 0xffffffff00000000 */**# defines L64_MASK as a 64-bit mask with the lower 32 bits set to 1 and the upper 32 bits set to 0. This means that when L64_MASK is used in a bitwise AND operation with a 64-bit value, only the lower 32 bits of that value will be retained, and the upper 32 bits will be set to 0. # defines H64_MASK as a 64-bit mask with the lower 32 bits set to 0 and the upper 32 bits set to 1**

/*

/* Initial Permutation Table */ {**The initial permutation table is a fixed table that specifies a permutation of the 64 bits in the input message block. The purpose of the initial permutation is to provide diffusion, which means that**

even small changes in the input message block will result in large changes in the output ciphertext.}

static char 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

};

/* Inverse Initial Permutation Table */{**The inverse initial permutation table is simply the inverse of the initial permutation table. The purpose of the inverse initial permutation is to undo the effects of the initial permutation so that the original message can be recovered.**}

static char PI[] = {

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

};

/*Expansion table */

static char E[] = {

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

};
```



```
/* Post S-Box permutation */

static char P[] = {
```

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

};

/* The S-Box tables */ {**The reason for having 8 S-boxes in DES is to introduce confusion in the input data. The output of all 8 S-boxes is then concatenated to produce a 32-bit output, which is then fed to the permutation tables.**

The permutation tables, on the other hand, are used to introduce diffusion in the input data. The PC2 table is used to permute the 56-bit key into 48 bits, which are used as input to the S-boxes in each encryption round.

The use of multiple S-boxes and permutation tables increases the complexity of the DES algorithm and makes it more resistant to attacks.}

static char S[8][64] = {{

```
/* 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,

7

```
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

}};
```

CRYPTOGRAPHY LAB

Semester IV

Practical No. 3

/* Permuted Choice 1 Table */

static char PC1[] = {

8

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

};

/* Permuted Choice 2 Table */

static char PC2[] = {

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

};

/* Iteration Shift Array */

static char iteration_shift[] = {

/* 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 */

1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1

};

/*

* The DES function

* input: 64 bit message

* key: 64 bit key for encryption/decryption
```

```
* mode: 'e' = encryption; 'd' = decryption

*/

uint64_t des(uint64_t input, uint64_t key, char mode) {

int i, j;

/* 8 bits */

char row, column;
```



```
/* 28 bits */

uint32_t C = 0;

uint32_t D = 0;

/* 32 bits */

uint32_t L = 0;

uint32_t R = 0;

uint32_t s_output = 0;

uint32_t f_function_res = 0;

uint32_t temp = 0;

/* 48 bits */
```

```
uint64_t sub_key[16] = {0};


uint64_t s_input = 0;
/* 56 bits */
uint64_t permuted_choice_1 = 0;
uint64_t permuted_choice_2 = 0;
/* 64 bits */
uint64_t init_perm_res = 0;
```



```
uint64_t inv_init_perm_res = 0;
uint64_t pre_output = 0;
/* initial permutation */
for (i = 0; i < 64; i++) {
init_perm_res <<= 1;
init_perm_res |= (input >> (64-IP[i])) & LB64_MASK;
}
L = (uint32_t) (init_perm_res >> 32) & L64_MASK;
```

R = (uint32_t) init_perm_res & L64_MASK;

{

The IP table defines the new position of each bit in the 64-bit input block. During each iteration of the loop, the "init_perm_res" variable is shifted to the left by one bit and then a new bit is added to its least significant bit position using bitwise OR operation. The expression (input >> (64-IP[i])) & LB64_MASK shifts the input data right by a certain number of bits (determined by the IP table value) and then applies a logical AND with a bit mask (LB64_MASK) to extract the least significant bit.

The result of this loop is a 64-bit value stored in the "init_perm_res" variable that represents the input block after the initial permutation step.

The second part of the code extracts the left (L) and right (R) halves of the initial permuted block. The 64-bit "init_perm_res" value is first shifted right by 32 bits to obtain the upper 32 bits (which represent L) and then masked with L64_MASK to discard any bits beyond the 32nd position.}

/* initial key schedule calculation */

for (i = 0; i < 56; i++) {

permuted_choice_1 <<= 1;

permuted_choice_1 |= (key >> (64-PC1[i])) & LB64_MASK;

}

C = (uint32_t) ((permuted_choice_1 >> 28) & 0x000000000fffffff); D = (uint32_t) (permuted_choice_1 & 0x000000000fffffff);

{

The first for loop performs the PC1 permutation by iterating through each of the 56 bits in the original 64-bit key. The extracted bit is obtained by shifting the key right by 64 - PC1[i] bits and masking it with the lower 6 bits of all 1s using the "&" operator with the LB64_MASK constant.

After the for loop completes, the 56-bit permuted key is divided into two 28-bit halves, C and D. The right half, D, is obtained by masking "permuted_choice_1" with the lower 28 bits of all 1s, which effectively clears

the upper 28 bits of "permuted_choice_1" and leaves only the lower 28 bits corresponding to D.}

/* Calculation of the 16 keys */

for (i = 0; i< 16; i++) {

/* key schedule */

// shifting Ci and Di

for (j = 0; j < iteration_shift[i]; j++) {

C = 0x0fffffff & (C << 1) | 0x00000001 & (C >> 27);

D = 0x0fffffff & (D << 1) | 0x00000001 & (D >> 27);

}

permuted_choice_2 = 0;

permuted_choice_2 = (((uint64_t) C) << 28) | (uint64_t) D ;

sub_key[i] = 0;

for (j = 0; j < 48; j++) {

sub_key[i] <<= 1;

sub_key[i] |= (permuted_choice_2 >> (56-PC2[j])) & LB64_MASK;

} } {**The first two lines define a 64-bit integer variable permuted_choice_2 and assign it a value generated by combining two 28-bit values C and D. It then**

uses a bitwise OR operation to combine the shifted value of sub_key[i] with the next bit of the permuted_choice_2 value, which is extracted using a bitwise right shift and a bit mask.

The bit mask LB64_MASK is defined elsewhere in the code and is used to ensure that only the least significant 6 bits of the 8-bit PC2 value are used to extract the corresponding bit from the permuted_choice_2 value. The PC2 array contains a fixed permutation table used to select a subset of the bits from the permuted_choice_2 value and rearrange them into the sub-key.

This code generates a 48-bit sub-key by selecting and rearranging a subset of the bits from the original 64-bit key.

}

for (i = 0; i < 16; i++) {

/* f(R,k) function */

s_input = 0;

for (j = 0; j< 48; j++) {

```
s_input <<= 1;

s_input |= (uint64_t) ((R >> (32-E[j])) & LB32_MASK);

}

/*

* Encryption/Decryption
```



```
* XORing expanded Ri with Ki

*/

if (mode == 'd') {

// decryption

s_input = s_input ^ sub_key[15-i];

} else {


// encryption

s_input = s_input ^ sub_key[i];

}
```

CRYPTOGRAPHY LAB

Semester IV

Practical No. 3

{**The code is performing the XOR operation between the "s_input" and a sub-key selected from a set of 16 sub-keys, based on the value of the variable "i". If the code is in decryption mode (i.e., the loop is currently at the 15th iteration), it selects the sub-key from the end of the sub-key array (15-i), whereas, in encryption mode, it selects the sub-key from the start of the sub-key array (i).**

The XOR operation is a bitwise operation that results in a binary 1 in each bit position where the corresponding bits of the two operands are different.

}

16

```
/* S-Box Tables */

for (j = 0; j < 8; j++) {

// 00 00 RCCC CR00 00 00 00 00 00 s_input

// 00 00 1000 0100 00 00 00 00 00 row mask

// 00 00 0111 1000 00 00 00 00 00 column mask

row = (char) ((s_input & (0x0000840000000000 >> 6*j)) >> 42-6*j); row =
(row >> 4) | row & 0x01;
column = (char) ((s_input & (0x0000780000000000 >> 6*j)) >> 43-6*j);
s_output <<= 4;
s_output |= (uint32_t) (S[j][16*row + column] & 0x0f);

}

f_function_res = 0;

for (j = 0; j < 32; j++) {

f_function_res <<= 1;
```

```
f_function_res |= (s_output >> (32 - P[j])) & LB32_MASK;

}

temp = R;

R = L ^ f_function_res;
```



```
L = temp;

}

pre_output = (((uint64_t) R) << 32) | (uint64_t) L;

/* inverse initial permutation */

for (i = 0; i < 64; i++) {



inv_init_perm_res <<= 1;

inv_init_perm_res |= (pre_output >> (64-PI[i])) & LB64_MASK; }
return inv_init_perm_res;

}

int main(int argc, const char * argv[]) {

int i;
```

uint64_t input = 0x9474B8E8C73BCA7D;

uint64_t key = 0x0000000000000000;

uint64_t result = input;

{The code takes an input message, represented by a 64-bit unsigned integer variable "input", and a 64-bit key, represented by the variable "key", and performs encryption using the DES algorithm. The code extracts the appropriate 6-bit input from each half of the message, computes the corresponding row and column values using masks and bit shifts, and then looks up the corresponding output from the S-box tables.

The permutation operation is performed using the "P" table, which reorders the bits in the output of the S-box substitution. The code then combines the results from each round, and performs a final inverse initial permutation using the "PI" table to produce the encrypted output.

The main() function initializes the input and key variables, and then assigns the value of "input" to "result".}

/*

* TESTING IMPLEMENTATION OF DES

* Ronald L. Rivest

* X0: 9474B8E8C73BCA7D

* X16: 1B1A2DDB4C642438

*

* OUTPUT:

* E: 8da744e0c94e5e17

* D: 0cdb25e3ba3c6d79

19

* E: 4784c4ba5006081f

* D: 1cf1fc126f2ef842

* E: e4be250042098d13

* D: 7bfc5dc6adb5797c

* E: 1ab3b4d82082fb28

* D: c1576a14de707097

* E: 739b68cd2e26782a

* D: 2a59f0c464506edb

* E: a5c39d4251f0a81e

* D: 7239ac9a6107ddb1

* E: 070cac8590241233

* D: 78f87b6e3dfecf61

```
* E: 95ec2578c2c433f0



* D: 1b1a2ddb4c642438 <-- X16

*/

for (i = 0; i < 16; i++) {

if (i%2 == 0) {
```



```
result = des(result, result, 'e');

printf ("E: %016llx\n", result);

} else {

result = des(result, result, 'd');

printf ("D: %016llx\n", result);

}

}

//result = des(input, key, 'e');

//printf ("E: %016llx\n", result);

//result = des(result, key, 'd');
```

```
//printf ("D: %016llx\n", result);

exit(0);

}
```