

Data Encryption Standard

- Implement DES with Rust and Linear Analysis -

Ji Yong-Hyeon

Department of Information Security, Cryptology, and Mathematics

College of Science and Technology
Kookmin University

March 7, 2024

Acknowledgements

Contents

- 1 Data Encryption Standard 1**
 - 1.1 Key Schedule 1
 - 1.2 The S-Boxes of DES 2

Chapter 1

Data Encryption Standard

- Symmetric Block Cipher.
- A.k.a Data Encryption Algorithm.
- Adopted by NIST in 1977.
- Advanced Encryption Standard (AES) in 2001.

Table 1.1: Parameters of the Block Cipher DES

Input	Output	Master Key	Sub-key	Round Key	No. of rounds
64-bit	64-bit	64-bit	56-bit	48-bit	16 rounds

1.1 Key Schedule

1.2 The S-Boxes of DES

- The Data Encryption Standard (DES) is a 64-bit block cipher with 16 rounds and a 56-bit key.
- There are eight S-Boxes used in DES. They are usually denoted by S_1, \dots, S_8 .
- Let $\{0, 1\}^n$ denote the set of sequences of bits (0's and 1's) of length n . We can equivalently think of $\{0, 1\}^n$ as the integers $\{0, 1, \dots, 2^n - 1\}$.
- Each S-Box of DES is a function $S_i : \{0, 1\}^6 \rightarrow \{0, 1\}^4$.
- So we can think of the domain of each function S_i to be the set of integers $\{0, 1, \dots, 63\}$ and the range to be $\{0, 1, \dots, 15\}$.
- The natural way to specify any function

$$S : \{0, \dots, 63\} \rightarrow \{0, \dots, 15\}$$

would just be as a list of 64 values, where the i -th value of the list for $0 \leq i \leq 63$ is the value of $S(i)$. Moreover, each value in the list would be an element of $\{0, \dots, 15\}$.

- Even though the S-Boxes of DES are specified using a list of 64 values, with each value being in $\{0, \dots, 15\}$, the list has to be interpreted differently than the natural way to determine the values of the given S-Box.

https://en.wikipedia.org/wiki/DES_supplementary_material

- Let $x = (b_5b_4b_3b_2b_1b_0) \in \{0, 1\}^6$ be a given 6-bit input.
- We form a 2-bit value $r = (b_5b_0) \in \{0, 1\}^2$ using the most significant bit b_5 and the least significant bit b_0 of x .
- The value of r is in $\{0, 1, 2, 3\}$ and it determines the row of the table.
- We also form a 4-bit value $c = (b_4b_3b_2b_1) \in \{0, 1\}^4$ using the inner four bits of x .
- The value of c is in $\{0, 1, \dots, 15\}$ and it determines the column of the table.
- Here we are using zero based counting. So for example $r = 0$ refers to the first row and $c = 15$ refers to the sixteenth column.

Example 1.1. If $x = (27)_{10} = (011011)_2$, the outer two bits are $r = (01)_2 = (1)_{10}$, so we look in the second row. The inner four bits are $c = (1101)_2 = (13)_{10}$ which gives the fourteenth column.

$$x = 011011 \rightarrow \begin{cases} 01 & \rightarrow \text{second row} \\ 1101 & \rightarrow \text{fourteenth column} \end{cases}$$

Therefore the value of $S_5(x)$ is $(9)_{10} = (1001)_2 \in \{0, 1\}^4$.

- Let $x = (b_5b_4b_3b_2b_1b_0) \in \{0, 1\}^6$. To access the most significant bit b_5 , we can use:

```
1 msb = x >> 5
```

To access the least significant bit b_0 , we can use:

```
1 lsb = x & 1
```

So the two-bit value $(b_5b_0) \in \{0, 1\}^2$ formed by the most and least significant bits is:

```
1 row = (msb << 1) | lsb
```

This gives us the row of the table we need to look at.

- To access the inner four bits of x we start by right-shifting one position to knock off the least significant bit b_0 . What's left is $(b_5b_4b_3b_2b_1)$.

To get rid of b_5 but keep the rest of the bits we AND this value with $(1111)_2$ which in hex is $(f)_{16}$. This is coded as:

```
1 col = (x >> 1) & 0xf
```

This gives us the column of the table we need to look at.

- To find the value of $S(x)$ we need to find the list index associated to x .

Now that we know the row and column values determined by x we can get the corresponding index of the table by:

$$\text{index} = \text{row} \times 16 + \text{col}$$

Note that we are multiplying by 16 because each row of the table has 16 values.

We can code this list index function as follows:

```

1 // # x is 6-bits
2 // # the index returned is in {0,...,63}
3 // def index(x):
4 //   msb = x >> 5 # most significant bit (the leftmost bit)
5 //   lsb = x & 1   # least significant bit (the rightmost bit)
6 //   row = (msb << 1) | lsb # outer 2-bits of x
7 //   col = (x >> 1) & 0xf   # inner 4-bits of x
8 //   return row * 16 + col # calculate the list index
9
10 fn index(x: u8) -> usize {
11   // Ensure x is 6 bits
12   let x = x & 0x3F; // Apply mask to limit to 6 bits if not
13                       // already
14   // Extract bits similarly to the Python version, utilizing
15   // Rust's type safety and bit operations
16   let msb: u8 = x >> 5; // Most significant bit
17   let lsb: u8 = x & 1;   // Least significant bit
18   let row: u8 = (msb << 1) | lsb; // Combine msb and lsb to
19                                   // form the row
20   let col: u8 = (x >> 1) & 0xF; // Extract the 4 middle
21                                   // bits for the column
22   // Calculate the final index, converting the row and col
23   // into usize for the return type
24   (row as usize) * 16 + (col as usize)
25 }
```


So the S-Box S_5 of DES can be implemented in the following way:

```

1 // # x is 6-bits
2 // # The index returned is in [0,...,63]
3 // def index(x):
4 //   msb = x >> 5 # most significant bit (the leftmost bit)
5 //   lsb = x & 1   # least significant bit (the rightmost bit)
6 //   row = (msb << 1) | lsb # outer 2-bits of x
7 //   col = (x >> 1) & 0xf   # inner 4-bits of x
8 //   return row * 16 + col # calculate the list index
9
10 // s5 = [2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9,
11 //       14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6,
12 //       4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14,
13 //       11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3]
14
15 // # x is 6-bits,
16 // # The return value is 4-bits
17 // def S5(x):
18 //   return s5[index(x)]
19
20 // # Example usage
21 // print(S5(27)) # should get 9
22
23 const S5_TABLE: [u8; 64] = [
24     2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9,
25     14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6,
26     4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14,
27     11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3,
28 ];
29
30 fn index(x: u8) -> usize {
31     let x = x & 0x3F; // Ensure x is 6 bits
32     let msb = x >> 5; // Most significant bit
33     let lsb = x & 1;   // Least significant bit
34     let row = (msb << 1) | lsb; // Outer 2-bits of x
35     let col = (x >> 1) & 0xF;   // Inner 4-bits of x
36     (row as usize) * 16 + (col as usize) // Calculate the list
37     index
38 }
39
40 fn s5(x: u8) -> u8 {
41     // Call index(x) to get the index, then use it to retrieve the
42     // value from S5_TABLE
43     S5_TABLE[index(x)]
44 }
45
46 fn main() {
47     println!("{}", s5(27)); // Should output 9
48 }

```