**Practical Session 3**

***Simplified DES (sDES)***

DES is a complex algorithm, and operates on 64-bit blocks, with a key of bits. This makes it almost impossible to work through the algorithm by hand to get a feeling of how it works. To help understand the intricacies and inner working of

DES, a simplified version has been developed by Edward Schaefer [1]. Encryption and decryption of this simplified cipher can be performed by hand, giving a valuable insight into the workings and design of DES.

**Initial permutation.**

This starts off by permuting the bits of the plaintext.



That is, bit 0 ends up in the third place, bit ends up in the zeroth place, bit 2 ends up in the second place, and so on (for all discussion of sDES, lists and arrays will be indexed starting at zero). This permutation can be expressed more simply as

(1,5,2,0,3,7,4,6)

Corresponding to this initial permutation is its inverse, which simply puts the bits back where they started from:



Again, this can be simply expressed as

(3,0,2,4,6,1,7,5)

To see how this works, consider bit 1 of the plaintext. According to the initial permutation, it gets placed in position 5. Now look at the inverse permutation. Bit 5 is placed in position 1 - exactly where it started from. Figure 1 shows this.

0 1 2 3 4 5 6 7

0 1 2 3 4 5 6 7

0 1 2 3 4 5 6 7

P

P-1

Figure 1. The permutation is sDES

**Subkeys**

Simplified DES uses a 10-bit key, of which subkeys of 8 bits each are used in each round. The subkeys are obtained as follows:

1. The ten bits of the key are first permuted according to the permutation:

(2,4,1 6,3 9,0 8,7 5)

1. The result is then split into two halves of five bits each.
2. For round one, each half is cyclically shifted one bit to the left:



1. Subkey one is obtained by taking these bits out of the result of the last operation:

(5,2,6,3,7,4,9,8)

1. For round two, the result of step 3 is shifted two bits to the left:



1. Then subkey two is obtained by taking the same bits out as were used for subkey one.

If the key is  then the two subkeys are





**The Feistel step**

At the heart of sDES is a Feistel-type function, as shown in Figure 2. Note that the left and right halves are not switched around, as they are in the Feistel structure reviewed at our sessions.

**The mixing function**

The next part of sDES is the function f(Ki,Ri) which mixes the bits in structure shown in Figure 2. Inputs are an 8-bit subkey, and a 4-bit block, which is first expanded into 8-bits.



The subkey is XOR-ed with this expansion, and result broken into two halves of 4 bits each. These halves are turned into 2 bits, each by means of two S-boxes S0 and S1. These are then put together to make 4 bits, and permuted according to



The S-boxes are defined by the tables:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 |
| 0 | 1 | 0 | 3 | 2 |
| 1 | 3 | 2 | 1 | 0 |
| 2 | 0 | 2 | 1 | 3 |
| 3 | 3 | 1 | 3 | 2 |

S-box S0

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 |
| 0 | 0 | 1 | 2 | 3 |
| 1 | 2 | 0 | 1 | 3 |
| 2 | 3 | 0 | 1 | 0 |
| 3 | 2 | 1 | 0 | 3 |

S-box S1

Li

Ri

*f*(Ki,Ri)

Li+1

Ri+1

Ki

+

Figure 2. The Feistel-type function in sDES

To apply an S-box, say S0, suppose the 4 input bits are . The outer two bits give the row number; the inner two bits the column number. So for example 1011 has row 11, or 3, and column 01, or 1. At row 3 and column 1 is the value 1, or 01 binary. The entire mixing function is shown in Figure 3.

4 bits

8 bits

8 bits

4 bits

4 bits

2 bits

2 bits

4 bits

4 bits

8 bit subkey

**+**

Expansion

Permutation

S0 S0

S11S0

Figure 3. The mixing function of sDES

**The rounds.**

Simplified DES has two rounds. The first round involves switching the two halves of the Feistel output; the second round keeps them as they are. So a complete encryption using sDES is:

1. Initial permutation
2. Feistel operation using subkey K1
3. Switch left and right halves
4. Feistel operation using subkey K2
5. Inverse permutation

**An example**

The plaintext will be 01010101 and key 0000011111. The subkeys can be obtained immediately as

K1 = 01101011

K2 = 10101010

Below are all the steps for the encryption.

**Step 1.** Initial permutation



**Step 2.** The left and right halves of the last step are 1100 and 1100 respectively. The right half has to be mixed with the subkey K1. This is done using the function shown in Figure 3.

First the 4 bits 1100 are expanded to 01101001, and added to the subkey K1 with XOR:



The first four bits are used as input to S0, and the second four bits used as input to S1:





Then the outputs are concatenated, 0101, and permuted, 1100. This output is now XOR-ed with the left half,



and concatenated with the right half 1100 to produce 00001100 as the output of this step.

**Step 3.** Interchange the left and right halves: 11000000.

**Step 4.** The left half is now 1100 and right half is 0000, with subkey K2 = 10101010. Expand the right half



and XOR with the subkey:



Put the two halves of this through the S-boxes





and concatenate, 1000 and permute: 0001. Again this is XOR-ed with the left half



and concatenated with the right half 0000 to produce 11010000 as the output of this step.

**Step 5.** Now the final inverse permutation:



This last string is the ciphertext.

**Decryption**

Given that XOR is self-inverse, the decryption of sDES is the same as the encryption, except that the keys are used in reverse order: K2 first, and K1 second:

1. Initial permutation
2. Feistel operation using subkey K2
3. Switch left and right halves
4. Feistel operation using subkey K1
5. Inverse permutation

This means that essentially the same program can be used for both encryption and decryption.

**Test Vectors**

This can be used as test vector [2] to test your own sDES implementation.

|  |  |  |
| --- | --- | --- |
| **Key** | **Plaintext** | **Ciphertext** |
| 0000011111 | 01010101 | 11000100 |
| 0010010111 | 00110110 | 10100101 |
| 0000000000 | 00000000 | 11110000 |
| 1111111111 | 11111111 | 00001111 |

**Activity**

1. Using the provided description and pseudo-code, in discuss how this algorithm should be implemented, what would be the best programming language to do it.
2. Discuss what is the process you need to follow to decrypt the data. Describe it and create the pseudo code.
3. Upload a file with all these findings individually.

**Implementation**

1. Make your individual submission on Alphagrader your implementation in the programming language of your choice. The testing cases are the ones presented on the table above.

**References**

[1] Edward Schaefer, “*A Simplified Data Encryption Standard algorithm*” Cryptologia, 20(1):77-84, 1996.

[2] Schneier, B., “Applied Cryptography: Protocols, Algorithms, and Source Code in C”, 2nd Edition, 1996.