Homework 2 - Information Security

Catherine Baker

January 2024

Contents

1 DES Non-Linear S-Boxes

Show that $S_1(x_1) \bigoplus S_1(x_2) \neq S_1(x_1 \bigoplus x_2)$

To show this for each given x_1 and x_2, we will first derive the right-hand side of the equation which will always simplify to a single substitution operation of a single value. We will then show that the two sides of the equation are not equal for each x_1 and x_2 because of the non-linearity of the substitution box in DES.

1.1 $x_1 = 000000, x_2 = 000001$

$$S_1(x_1) \bigoplus S_1(x_2) \neq S_1(x_1 \bigoplus x_2)$$

$$\Rightarrow S_1(x_1 \bigoplus x_2) = S_1(000000 \bigoplus 000001)$$

Further simplifying the right-hand side:

$$\Rightarrow S_1(000001) = S_1(x_2)$$

Then our equation becomes:

$$S_1(x_1) \bigoplus S_1(x_2) \neq S_1(x_2)$$

Because it is non-linear, the output of $S_1(x_2)$ cannot be represented by a simple XOR operation and therefore the left- and right-hand sides of the above equation cannot be equal.

1.2 $x_1 = 111111, x_2 = 100000$

$$S_1(x_1 \bigoplus x_2) = S_1(111111 \bigoplus 100000)$$

$$\Rightarrow S_1(011111)$$

Then our equation becomes:

$$S_1(x_1) \bigoplus S_1(x_2) \neq S_1(011111)$$

As in Section 1.1, because it is non-linear, the output of $S_1(011111)$ cannot be represented by a simple XOR operation and therefore the left- and right-hand sides of the above equation cannot be equal.

1.3 $x_1 = 101010, x_2 = 010101$

$$S_1(x_1 \bigoplus x_2) = S_1(101010 \bigoplus 010101)$$

$$\Rightarrow S_1(111111)$$

Then our equation becomes:

$$S_1(x_1) \bigoplus S_1(x_2) \neq S_1(011111)$$

And again, as above, because it is non-linear, the output of $S_1(011111)$ cannot be represented by a simple linear XOR operation of the inputs, and therefore the left- and right-hand sides of the above equation cannot be equal.

2 Inverse IP Operations

The initial permutation IP of the first 64 bits of data in DES is meant to re-order the data slightly as a sort of priming to the search algorithm (possibly to order the data in 8-bit busses to make it more efficient to fetch). The final permutation or IP^{-1} is meant to perform the same permutation but in reverse order, putting the rearranged bits back in their original locations. To show this process through an example, we will run the first five bits of data from the vector $x = [x_1, x_2, ..., x_{63}, x_{64}]$, through the initial permutation:

$$IP(x_1, x_2, ..., x_5) = x_2, x_4, x_1, x_5, x_3$$

Then performing the final permutation:

$$IP^{-1}(x_2, x_4, x_1, x_5, x_3) = x_1, x_2, x_3, x_4, x_5$$

the vector is re-sorted and the the x's are in the correct order. And since the input of $IP^{-1}()$ equals the output of $IP()$:

$$x_2, x_4, x_1, x_5, x_3 = IP(x_1, x_2, ..., x_5)$$

We can further show:

$$IP^{-1}(IP(x_1, x_2, ..., x_5)) = x_1, x_2, x_3, x_4, x_5$$

which holds for the entire set of x, and therefore:

$$IP^{-1}(IP(x)) = x$$

3 DES Output: Zeroes

The first step in DES is the initial permutation. Since our plaintext is all zeroes it's permutation results in all zeroes as well. We then split the plaintext into left and right halves of 32 bits each, which both consist of zeroes.

Next we use the 64-bit key of zeroes to generate 16 48-bit keys through permutation and shifting which results in a round 1 key K_1 of all zeroes. We then expand the 32-bit right half so we can XOR it with our key. This is done through an expansion table but would results in a right half of all zeroes since we started with all zeroes. We then XOR the 48-bit plaintext with the 48-bit K_1 which are both all zeroes and therefore result in 48 bits of zeroes.

We then go into our byte substitution which is done in 6-bit sections trimming our text down to 4 bits after the substitution is finished. The output here will depend on our substitution table's output for a 0 input, but assuming the substitution table on Slide 41 of the in-class presentation, results in values of 14 (1110). All 8 6-bit inputs will be returned identically as S_1 4-bit ouputs for 0 (1110 1110 1110 1110 1110 1110 1110 1110). Then we permute these bits again according to the permutation table P from the same presentation: 0101 1001 1111 1111 1001 0111 1111 1101.

The next step XORs the right and left halves. The left half is entirely zeroes up to this point, so XOR-ing it with the right half will produce the right half as the result. After this XOR, we swap the left and right halves.

So the result after this first round of DES is a 32-bit left half: 0101 1001 1111 1111 1001 0111 1111 1101. We also have a new right half of all zeroes. After going through DES again, the right half will still change a lot, especially since the left half is now non-zero.

4 DES Output: Ones

Because the plaintext is all ones, the initial permutation results in an identical plaintext. Likewise, our 64-bit key is all ones and its round 1 key K_1 (derived through permutations and shifts) will be all ones too. The plaintext is split into two 32-bit halves of all ones.

The right half is then expanded to 48 bits of all ones and XOR'd with K_1. This will result in a plaintext of all zeroes since $1 \bigoplus 1 = 0$ for all the bits in the plaintext. This new all zero right half gets fed through the substitution boxes (with 0 mapping to 14 as before) and that result is permuted as in Section 3: 0101 1001 1111 1111 1001 0111 1111 1101.

This result is then XOR'd with the left half which is currently all ones. This will produce an output that is equivalent to switching all the values of the right side from 0 to 1 or 1 to 0 (the mirror image of the plaintext): 1010 0110 0000 0000 0110 1000 0000 0010. Then we swap the left and right halves.

So the new right half is all ones. The new left half is: 1010 0110 0000 0000 0110 1000 0000 0010.

5 DES Diffusion

We are given that the 64-bit input text has a 1 at position 57 and a 0 at all other positions. We are also given that this plaintext has yet to undergo the initial permutation. Let us walk through the DES algorithm given this context.

First we would undergo the initial permutation which, according to the Initial DES Permutation table on Slide 30 of the DES in-class powerpoint, would displace the 1 at position 57 to the first position of the right half. We then split the plaintext into two 32-bit halves.

We first expand the right half through a defined expansion table (which results in a duplicated 1 value in the second and last positions: 010000 000000 000000 000000 000000 000000 000000 000001). We then XOR the now 48-bit half with the 48-bit K_1 which results in a copy of the right half since the key is all zeroes. We then perform substitutions: 0011 1110 1110 1110 1110 1110 1110 0000, and permutations: 0101 0001 0111 1101 0001 0111 1101 1111.

This gets XOR'd with the left half (all zeroes) resulting in a copy of the right half: 0101 0001 0111 1101 0001 0111 1101 1111.

5.1 S-Boxes with All-Zero Plaintext

Compared to DES with all zeroes, this round had **two** 6-bit sections altered (the 2 containing the shuffled and permuted ones). In the substitution process most sections turned from 000000 to 1110. The first changed from 010000 to 0011, and the last from 000001 to 0000.

5.2 Output Bits of S-Boxes

Because the S-box maps a 6-bit input to a 4-bit output, any change in one of the six values will result in a different 4-bit output (the answer is **4 bits**). This increases the confusion of our algorithm.

5.3 Output After Round 1

As shown in Section 5, the output is: 0101 0001 0111 1101 0001 0111 1101 1111

5.4 Changed Output Bits

All zero plaintext after round 1: 0101 1001 1111 1111 1001 0111 1111 1101 Our plaintext after round 1: 0101 0001 0111 1101 0001 0111 1101 1111

6 bits have been changed in total, either from 0 to 1 or 1 to 0.

6 DES Keys

An exhaustive key search attack is equivalent to a brute-force attack where the attacker tries every possible key until the right one is found. Given the nature of DES and a 64-bit key, the attacker would only need to find the 56-bit key that is used cryptographically. The key has a length of 56 so the worst-case scenario for this attack is having to test 2^{56} keys. The average-case scenario occurs when the attacker finds the key halfway through the search with $\frac{2^{56}}{2} = 2^{55}$ keys to search through.

7 PRESENT

We are given a plaintext of 0000 0000 0000 0000 and a starting key of BBBB 5555 5555 EEEE FFFF. We are also given a Table 7.1 shown and filled out in Section 7.1. We looked for and found a paper, PRESENT: Ultraweight Block Cipher, on the algorithm for PRESENT to complete this problem.

7.1 State After Round 1

To first find the round key we took the leftmost 64-bits of the original key: BBBB 5555 5555 EEEE.

Plaintext	0000 0000 0000 0000
RoundKey	BBBB 5555 5555 EEEE
State After KeyAdd	BBBB 5555 5555 EEEE
State After S-Layer	8888 0000 0000 1111
State After P-Layer	8001 8001 8001 8001

7.2 Round 2 Key

To find the second round key we first convert our hex key to binary: 1011101110111011 0101010101010101 0101010101010101 1110111011101110 1111111111111111 and rotated 61 bits to the left. We will show this and all other operation results on Table 7.2 by converting everything back to hex.

Key	BBBB 5555 5555 EEEE FFFF
Key State After Rotation	DFFF F777 6AAA AAAA BDDD
Key State After S-Box	7FFF F777 6AAA AAAA BDDD
Key State After CounterAdd	7FFF F577 6AAA AAAA BDDD
Round Key for Round 2	7FFF F577 6AAA AAAA

8 LSFR Implementation

See code lsfr.py included in zip file. Code contains comments defining each line and the process. To run this code please navigate to the command line in terminal in the folder containing the file and execute 'python lfsr.py' to see the generated pseudo-random binary output. It is currently set to only produce 10 random numbers but that amount can be edited on line 32 of the code.