Halil İbrahim Kaplan

707221004

HW2

To apply differential cryptanalysis, we need differential characteristics for the cipher.

To find differential characteristics, we need to compute DDT tables for the S-boxes.

With the code below I computed DDT table for each S-boxes.

```python
def print_ddt(sbox, inputsize):

    # Defines two dimentional array for DDT
    ddt_values = [[0 for x in range(pow(2, inputsize))] for y in range(pow(2, inputsize))]
    for dx in range(pow(2, inputsize)):

        for x in range(pow(2, inputsize)):
            # Calculates y and dy according to x and dx
            x1 = dx ^ x
            y = sbox[x]
            y1 = sbox[x1]
            dy = y ^ y1

            # Sets values of the DDT
            ddt_values[dx][dy] = ddt_values[dx][dy] + 1

    for element in ddt_values:
        print(element)
```

For S-box 1:

```
[16, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 4, 0, 0, 0, 4, 0, 2, 2, 0, 0, 2, 2]
[0, 0, 2, 0, 2, 0, 2, 2, 0, 0, 0, 2, 6, 0, 0, 0]
[0, 0, 2, 2, 2, 0, 2, 0, 4, 0, 0, 0, 2, 0, 0, 2]
[0, 0, 2, 2, 0, 0, 6, 2, 0, 0, 4, 0, 0, 0, 0, 0]
[0, 0, 4, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 4, 0]
[0, 4, 0, 0, 2, 0, 2, 0, 0, 0, 4, 0, 2, 0, 2, 0]
[0, 0, 2, 0, 2, 4, 0, 0, 0, 0, 2, 0, 2, 0, 4, 0]
[0, 0, 0, 4, 0, 2, 0, 2, 4, 0, 0, 0, 0, 2, 0, 2]
[0, 0, 0, 2, 0, 2, 0, 4, 0, 0, 0, 2, 4, 2, 0, 0]
[0, 4, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4]
[0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 2, 6, 0, 0, 2, 2]
[0, 0, 2, 0, 0, 6, 0, 0, 0, 0, 0, 2, 0, 2, 2, 2]
[0, 4, 0, 0, 0, 2, 2, 0, 0, 0, 2, 2, 0, 2, 0, 2]
[0, 0, 2, 2, 0, 0, 2, 2, 0, 8, 0, 0, 0, 0, 0, 0]
[0, 4, 0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 0, 8, 0, 0]
```

For S-box 2:

```
[16, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 2, 2, 2, 0, 0, 2, 0, 0, 0, 2, 0, 0, 2, 2, 2]
[0, 2, 2, 2, 0, 0, 0, 2, 0, 2, 0, 0, 4, 0, 2, 0]
[0, 2, 2, 0, 2, 0, 2, 0, 4, 0, 0, 0, 0, 0, 2, 2]
[0, 0, 0, 2, 0, 2, 6, 2, 0, 0, 2, 0, 0, 2, 0, 0]
[0, 0, 0, 0, 2, 4, 0, 2, 2, 0, 0, 2, 4, 0, 0, 0]
[0, 2, 0, 2, 6, 0, 2, 0, 0, 0, 4, 0, 0, 0, 0, 0]
[0, 0, 2, 0, 2, 2, 0, 2, 2, 2, 0, 2, 0, 0, 2, 0]
[0, 0, 0, 4, 0, 2, 0, 2, 4, 0, 0, 0, 0, 2, 0, 2]
[0, 0, 2, 0, 0, 0, 0, 2, 0, 2, 2, 2, 0, 2, 4, 0]
[0, 2, 0, 0, 2, 0, 4, 0, 0, 2, 0, 0, 2, 0, 0, 4]
[0, 0, 0, 0, 0, 2, 0, 2, 0, 2, 0, 2, 2, 2, 2, 2]
[0, 0, 4, 0, 0, 4, 0, 0, 0, 0, 2, 2, 0, 0, 2, 2]
[0, 2, 0, 0, 2, 0, 0, 0, 2, 2, 0, 2, 0, 4, 0, 2]
[0, 2, 2, 2, 0, 0, 0, 2, 0, 4, 0, 2, 2, 0, 0, 0]
[0, 2, 0, 2, 0, 0, 0, 0, 2, 0, 4, 2, 2, 2, 0, 0]
```

Then I construct differential for Cipher 1.

Note: We know key xor does not affect differential since we add same key to the both value. Therefore, I omit key xor part in the below table.

		Differential				Probability
	INPUT	0	0	0001	0	
Round 1	S-BOX	0	0	0100	0	4/16
	PERM.	0	0010	0	0	
Round 2	S-BOX	0	1100	0	0	6/16
	PERM.	0100	0100	0	0	
Round3	S-BOX	0110	0110	0	0	6/16, 6/16
	PERM.	0	1100	1100	0	

Total probability is: $\frac{4}{16} * \left(\frac{6}{16}\right)^3 = \frac{864}{65536} = \frac{27}{2048} \approx 0.013$

We have characteristic for differential cryptanalysis of cipher 1. So, we can start implementing the SPN.

I implement SPN uses array of size 4 as an input, s-box, key xor. Ex: [12,4,5,0].

In permutations, I transferred 4 integer state to the 16-bit state and does permutation. After permutation, state again becomes inter of 4 inputs.

Here my round function:

```
def cipher1round(girdi, anahtar,sbox,perm):
    # Define empty arrays
    sboxcikti = [0] * 4; permgirdi = [0] * 4
    permara = [0] * 16 ; permcikti = [0] * 4
    keyxorcikti = [0] * 4
    # S-box
    for i in range(0, 4):
        sboxcikti[i] = sbox[girdi[i]]
    # Permutation
    permgirdi = integer_to_bit_array(sboxcikti, 4)
    for i in range(0, 16):
        permara[i] = permgirdi[perm[i]]
    permcikti = bit_array_to_integer(permara, 4)
    # Key xor
    for i in range(0, 4):
        keyxorcikti[i] = permcikti[i] ^ anahtar[i]

    return keyxorcikti
```

Here my key schedule functions:

```
def keyScheduleFunc(Mkey):
    # Define empty arrays
    key1 = [0] * 16
    key2 = [0] * 16

    key1 = integer_to_bit_array(Mkey, 4)
    for i in range(0, 15):
        key2[i] = key1[i + 1]
    key2[15] = key1[15] ^ key1[13] ^ key1[11] ^ key1[10]
    return bit_array_to_integer(key2, 4)
```

```python
def keySchedules(Mkey, round):
    # Define empty arrays
    key1 = [0] * 4; key2 = [0] * 4
    key3 = [0] * 4; key4 = [0] * 4
    key5 = [0] * 4; key6 = [0] * 4
    # Computes round keys
    key1 = keyScheduleFunc(Mkey); key2 = keyScheduleFunc(key1)
    key3 = keyScheduleFunc(key2); key4 = keyScheduleFunc(key3)
    key5 = keyScheduleFunc(key4); key6 = keyScheduleFunc(key5)
    #Returns corresponding round key
    if round == 0:
        return Mkey
    elif round == 1:
        return key1
    elif round == 2:
        return key2
    elif round == 3:
        return key3
    elif round == 4:
        return key4
    elif round == 5:
        return key5
    elif round == 6:
        return key6
```

And here my encryption function:

```python
def cipher1(girdi, masterkey):
    # Define empty arrays
    round1 = [0] * 4; round2 = [0] * 4
    round3 = [0] * 4; round4 = [0] * 4
    round41 = [0] * 4; cikti = [0] * 4
    key0 = [0] * 4

    # First tree rounds
    key0 = keySchedules(masterkey, 0)
    keywhitening = xor(girdi,key0 ,4)
    round1 = cipher1round(keywhitening,keySchedules(masterkey,1),sbox1,perm1)
    round2 = cipher1round(round1, keySchedules(masterkey, 2),sbox1,perm1)
    round3 = cipher1round(round2, keySchedules(masterkey, 3),sbox1,perm1)
    round4 = xor(round3,keySchedules(masterkey,4),4)
    # Last round S-box
    for i in range(0, 4):
        round41[i] = sbox1[round4[i]]
    cikti = xor(round41,keySchedules(masterkey,5),4)
    return cikti
```

I have more functions like converts integer arrays to bit array etc. I omit them for simplicity of the report.

1. ATTACK OF BLOCK CIPHER 1

I generate random plaintexts. Then with xoring the differential to the plaintexts, I obtain differential plaintexts and write random plaintexts and differential plaintexts to two different text files.

Text files look like below (as you can see initial differential is [0,0,0001,0]):

```
6 11 6 10
4 0 8 0
13 0 4 7
0 9 6 12
4 13 7 5
5 11 6 7
0 7 1 13
9 0 2 9
3 9 8 7
0 11 3 7
10 5 8 9
13 0 5 1
0 3 9 12
0 12 12 14
```

```
6 11 7 10
4 0 9 0
13 0 5 7
0 9 7 12
4 13 6 5
5 11 7 7
0 7 0 13
9 0 3 9
3 9 9 7
0 11 2 7
10 5 9 9
13 0 4 1
0 3 8 12
0 12 13 14
```

Then I encrypted those plaintexts and similarly write them into two text files.

```
13 9 13 0
13 3 14 3
14 9 0 4
12 11 12 8
13 15 6 14
11 9 13 5
12 12 14 2
9 4 0 9
5 10 14 5
12 11 15 14
10 3 8 1
```

```
13 9 6 15
7 12 4 9
11 3 14 2
3 1 12 8
8 14 0 0
11 9 6 11
12 2 14 12
12 12 6 9
3 10 6 6
15 0 5 10
10 10 3 13
```

In each attack, new pairs are created ,encrypted with new random key and writed to the text files.

Here the code that does what I explained.(Number of pairs and input differential can be changed)

```
count=0
reverse_sbox_output_one=[0,0,0,0]
reverse_sbox_output_two=[0,0,0,0]
reverse_key_xorb=[0,0,0,0]
reverse_key_xora=[0,0,0,0]
number_of_pairs=4096
input_differential=[0,0,1,0]
```

```
plaintexts = open("Plaintexts5000.txt", "w")
d_plaintexts = open("Differential Plaintexts5000.txt", "w")
encrypted_plaintexts = open("Encrypted plaintexts5000.txt", "w")
d_encrypted_plaintexts = open("Differential Encrypted plaintexts5000.txt", "w")

for i in range(number_of_pairs):
    plaintext = random_key()

    # writes Plaintexts to the text file
    for ele in plaintext:
        plaintexts.write(str(ele) + " ")
    plaintexts.write("\n")

    # writes plaintext which satisfy differential
    d_plaintext=xor(plaintext,input_differential,4)
    for ele in d_plaintext:
        d_plaintexts.write(str(ele)+ " ")
    d_plaintexts.write("\n")

    # writes Ciphertexts to the text file
    for ele in cipher1(plaintext,anahtar):
        encrypted_plaintexts.write(str(ele) + " ")
    encrypted_plaintexts.write("\n")

    # writes corresponding differential ciphertext
    for ele in cipher1(d_plaintext,anahtar):
        d_encrypted_plaintexts.write(str(ele)+ " ")
    d_encrypted_plaintexts.write("\n")
```

Then for the attack. I read ciphertexts from the text file and and reverse key xor and s box for key candidate and look if it fits to the characteristics. If it fits to the characteristic, then I increment the count for the key candidate. I do this for all possible key candidates. Most counted key will be our key in the last round. From probability of the characteristic, For 4096 pairs I expect 53 count. 53 is not exact number ıt can be more or less depending on the key and random plaintexts.

Here my code does what I explained above.

```
            # Makes XOR between ciphertext and key candidate
            key_xora = xor(ciphertextlist_one, [0, key1, key2, 0],4)

            key_xorb = xor(ciphertextlist_two, [0, key1, key2, 0],4)

            # Puts result to inverse S-box

            reverse_sbox_out1 = s_box_inverse(key_xora,reverse_sbox1)
            reverse_sbox_out2 = s_box_inverse(key_xorb,reverse_sbox1)
            # Calculates differential
            diff = xor(reverse_sbox_out1, reverse_sbox_out2,4)
            # Checks if pair is right pair
            if (diff[0] == 0) and (diff[1] == 12) and (diff[2] == 12) and (diff[3] == 0):
                count += 1

        # Find max count and keys
        if count > max_count:
            max_count = count
            k = key1
            kk = key2
        print("key1:{} key2:{} Count : {}".format(key1, key2, count ))
        ciphertexts_one.close()
        ciphertexts_two.close()

print("----------------------------------------\n Last key : {} \n".format(keySchedules(anahtar,5)))
print("Max count: {} at key1 : {}  key2 : {} , Prob : {} \n".format(max_count, k, kk , max_count/number_of_pairs))
```

And here my results.

```
key1:15 key2:3 Count : 14
key1:15 key2:4 Count : 0
key1:15 key2:5 Count : 12
key1:15 key2:6 Count : 12
key1:15 key2:7 Count : 0
key1:15 key2:8 Count : 9
key1:15 key2:9 Count : 0
key1:15 key2:10 Count : 0
key1:15 key2:11 Count : 16
key1:15 key2:12 Count : 1
key1:15 key2:13 Count : 9
key1:15 key2:14 Count : 23
key1:15 key2:15 Count : 0
----------------------------------------
 Last key : [3, 1, 14, 7]


Max count: 48 at key1 : 1  key2 : 14 , Prob : 0.01171875
```

Here another result for different key.

```
key1:15 key2:10 Count : 3
key1:15 key2:11 Count : 0
key1:15 key2:12 Count : 7
key1:15 key2:13 Count : 2
key1:15 key2:14 Count : 1
key1:15 key2:15 Count : 7
------------------------------------------
 Last key : [15, 4, 2, 6]

Max count: 54 at key1 : 4  key2 : 2 , Prob : 0.01318359375
```

So we found [---, key1 , key2 , ---]. Other keys can be found by brute force. With Reversing the key schedule, master key can be found.

2. ATTACK OF BLOCK CIPHER 2

New characteristic for cipher 2:

I used DDT table for S-box 2 and permutation.

		Differential				Probability
	INPUT	0	0	0110	0	
Round 1	S-BOX	0	0	0100	0	6/16
	PERM.	0	0010	0	0	
Round 2	S-BOX	0	1100	0	0	4/16
	PERM.	0100	0100	0	0	
Round3	S-BOX	0110	0110	0	0	6/16,6/16
	PERM.	0	1100	1100	0	

Total probability is: $\frac{4}{16} * \left(\frac{6}{16}\right)^3 = \frac{53}{4096} \approx 0.013$

I implemented cipher2(just changed s-box) made the same process with the attack (with appropriate changes of attack code) of cipher1 then I obtained [---, ---, key1, key2]

I give 2 examples of attack with different keys

```
key1:15 key2:5 Count : 0
key1:15 key2:6 Count : 1
key1:15 key2:7 Count : 3
key1:15 key2:8 Count : 16
key1:15 key2:9 Count : 0
key1:15 key2:10 Count : 14
key1:15 key2:11 Count : 4
key1:15 key2:12 Count : 14
key1:15 key2:13 Count : 0
key1:15 key2:14 Count : 13
key1:15 key2:15 Count : 1
------------------------------------------
 Last key : [0, 8, 13, 5]

Max count: 94 at key1 : 8  key2 : 13 , Prob : 0.02294921875
```

```
key1:15 key2:1 Count : 8
key1:15 key2:2 Count : 3
key1:15 key2:3 Count : 15
key1:15 key2:4 Count : 2
key1:15 key2:5 Count : 23
key1:15 key2:6 Count : 6
key1:15 key2:7 Count : 9
key1:15 key2:8 Count : 5
key1:15 key2:9 Count : 4
key1:15 key2:10 Count : 6
key1:15 key2:11 Count : 6
key1:15 key2:12 Count : 1
key1:15 key2:13 Count : 1
key1:15 key2:14 Count : 1
key1:15 key2:15 Count : 4
------------------------------------------
 Last key : [0, 12, 5, 1]

Max count: 78 at key1 : 12  key2 : 5 , Prob : 0.01904296875
```

3. ATTACK OF BLOCK CIPHER 3

We need new characteristic.

I used DDT table for S-box 2 and no permutation.

		Differential				Probability
	INPUT	0	0	1100	1100	
Round 1	S-BOX	0	0	0010	0010	4/16, 4/16
	PERM.	0	0	0010	0010	
Round 2	S-BOX	0	0	1100	1100	4/16, 4/16
	PERM.	0	0	1100	1100	
Round3	S-BOX	0	0	0010	0010	4/16, 4/16
	PERM.	0	0	0010	0010	

Total probability is: $\left(\frac{1}{4}\right)^6 = \frac{1}{4096} \approx 0.002$

The probability is quite low therefore I will use 10.000 pair for the attack.

Other part of the attack is the same.

Here 2 of my results:

```
key1:15 key2:2 Count : 0
key1:15 key2:3 Count : 1
key1:15 key2:4 Count : 1
key1:15 key2:5 Count : 1
key1:15 key2:6 Count : 0
key1:15 key2:7 Count : 1
key1:15 key2:8 Count : 1
key1:15 key2:9 Count : 0
key1:15 key2:10 Count : 0
key1:15 key2:11 Count : 2
key1:15 key2:12 Count : 0
key1:15 key2:13 Count : 1
key1:15 key2:14 Count : 0
key1:15 key2:15 Count : 1
------------------------------------------
 Last key : [0, 4, 5, 1]

Max count: 17 at key1 : 5  key2 : 1 , Prob : 0.0017
```

```
key1:15 key2:0 Count : 0
key1:15 key2:1 Count : 2
key1:15 key2:2 Count : 1
key1:15 key2:3 Count : 0
key1:15 key2:4 Count : 2
key1:15 key2:5 Count : 3
key1:15 key2:6 Count : 0
key1:15 key2:7 Count : 0
key1:15 key2:8 Count : 0
key1:15 key2:9 Count : 0
key1:15 key2:10 Count : 0
key1:15 key2:11 Count : 1
key1:15 key2:12 Count : 0
key1:15 key2:13 Count : 0
key1:15 key2:14 Count : 0
key1:15 key2:15 Count : 1
------------------------------------------
 Last key : [12, 14, 13, 11]

Max count: 14 at key1 : 13  key2 : 11 , Prob : 0.0014
```

So I get key [---, ---, key1, key2]
Other part of the key can be found by reversing key schedule and brute force.

4. COMPARISON

For the first two cipher, I think there is no much difference. They have same permutation and key schedule. Only S-boxes are different. In the first S-box higher value on the DDT is 8 . In the second S-box, higher value is 6. So with this information we can say that S-box 2 is better then S-box 1. But the position of 8 is not usable in my characteristic so I didn’t use the 8. Other than that attack was very similar.

In the third cipher we don’t have any permutation. With no permutation, characteristic can consist of columns of S-boxes. Therefore, number of active s box cannot be reduced for obtaining 8 bit of key and this situation has negative effect to the probability.

5. CODE

I made my implementation with python.

With running main, you can test result yourself.

```
C:\Users\Halil\AppData\Local\Programs\Python
1 - DDT
2 - Attack on Cipher 1
3 - Attack on Cipher 2
4 - Attack on Cipher 3
5 - Quit
Go to :
```