Yash Maniya
B20CS033

Cryptograph HW2

Q1. Padding Oracle Attack against AES :

- I have used the **pycryptodome** python library and have written code in a jupyter notebook (ipynb).
- For padding, following will be used in the pad_message()

  p = 16 - plaintext_length % 16

  Done implementing padding scheme, encrypt, decrypt functions as well as padding_oracle. (easy and not much thinking process required)

- Plaintext used is "B20CS033YashRajeshHiralSurat" with length = 28
- First step of the attack is to find the pad length.
- To find pad_length, we iterate over each byte of the given cipher and increment it as well as ask the oracle until it returns False. At this point we know that it is the very first pad block.
- Then after knowing the pad length by above step,
  - we increment all the pad value by 1,
  - keep on changing the first non-padded byte from the right, until I get the answer as true
  - whenever I get true, the following relation of plaintext would hold :

    plain_text[i] = IV[i] ^ X[i-16] ^ padded_length

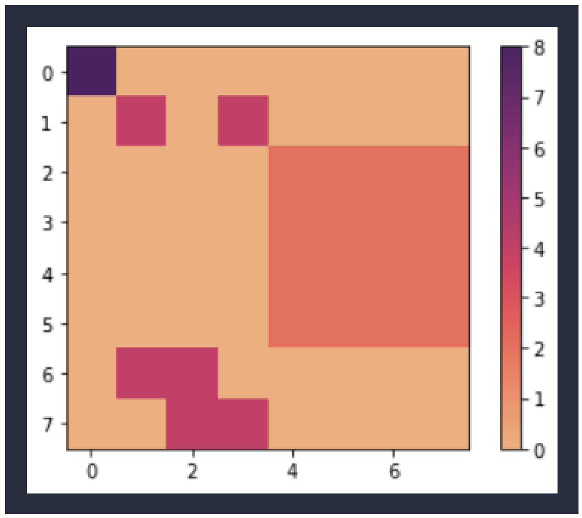
    Where X is IV if i<16, else it is cipher_text[i-16]

```
Plaintext retrieved from the attack :    B20CS033YashRajeshHiralSurat
Original Plaintext :                     B20CS033YashRajeshHiralSurat

Plaintext retrieved from Attack is same as the original one!!
```

Q2. Attack against SPN :

- For constructing DDT, a table (2d list) is created to keep the count of delta x and delta y value pairs. Consider, we selected a value for x now we xor it with every other possible value of x. This pair every time is passed on to an s-box and the obtained outputs are XORed to compute the delta y value. The count of these pairs is logged into the table.

```
Question 2 (a) - DDT :

   0  1  2  3  4  5  6  7
0  8  0  0  0  0  0  0  0
1  0  4  0  4  0  0  0  0
2  0  0  0  0  2  2  2  2
3  0  0  0  0  2  2  2  2
4  0  0  0  0  2  2  2  2
5  0  0  0  0  2  2  2  2
6  0  4  4  0  0  0  0  0
7  0  0  4  4  0  0  0  0
```



- The subkey mixing operation is now carried out for the first two rounds, after which they are passed on to substitution boxes, and finally the output is rearranged. The output bits are not rearranged in the third round, and the final round's series of steps ends immediately after the subkey mixing operation itself. So the last round would give the Ciphertext itself.

```
Question 2 (b) :

Round 1 : Row A : 001111
Round 2 : Row B : 101100
Round 3 : Row C : 110010
Round 4 : Row D : 111001
Round 5 : Row E : 100101
Round 6 : Row F : 110001
Round 7 : Row G : 001001
Round 8 : Row H : 101101
Round 9 : Row J : 010011

Encryption of the Plaintext is : 010011
```

- In order to determine the potential values of δH, all of the cases in which the P XOR P' value is 1 should be taken into account. If this is true for a particular pair, we determine c1 and c2 to be the H binary strings that are produced when P and P' are encrypted. This value's xor is kept in a set. We were able to determine 6 possible values for δH after combining every possible combination of plaintext. Here, the set's size corresponds to the number of possible values.

```
Question 2 (c) - Possible δH values :

Value  : Frequency

000001 : 16
010100 : 8
010000 : 8
000101 : 16
010101 : 8
010001 : 8
```

- On examining, the difference between p1 and p2 is 1, and calculation yields a differential of H of 010011. Now, starting with the cipher bits, we compute the last 3 bits of H in an inverse manner to determine the final 3 bits of K4. We compute the H bits using all possible combinations of the last 3 bits of K4. We're hoping to find the combination with the highest frequency. So the combination with the highest frequency is supposed to be K4.

```
Question 2 (d) - Last 3 bits of K4 :

Value  : Frequency

100000 : 1
100001 : 1
010111 : 2
000011 : 1
010010 : 1
000010 : 1
110101 : 2
110000 : 1
```

```
Hence we suppose K4 must be 010111
So, last 3 bits of K4 are : 111
```