Assignment 2

Name - Tummala Madhav
Roll No - 16CS01041

- Block ciphers process data in chunks.
- For an encryption process, we need data and key.
- Changing the data based on the key material is called Key Mixing, which is often just a XOR operation between both.
- This alone isn't sufficient to ensure security, as the key can be found easily by XORing the plain-text and cipher-text (known-plain-text attack).
- Permutation operations are inserted to add diffusion, to distribute the information throughout the cipher text, making it immune to freqency analysis attacks.
- Substitution operations add confusion to cipher. This makes it more resistant to linear and diffrential cryptanalysis. Substitution functions have to be reversible for the decryption to work and hence they by themselves do not provide any security, but in combination with key mixing and permuatation operations, they make it really diffcult for attackers.

In the design of a block cipher, each round has a key mixing (with sub-keys), a substitution and a permuation operation and this is reapeated for several rounds to improve the security.
The arrangement of rounds can be simple as in AES or follow a feistel structure like in DES.

We will take a simple SPN (substitution permuation network) and analyse the attacks that can be done on it. The algorithm will be more similar to AES, it will be a more compromised version for the purpose of testing.

Design of the Algorithm

For the purpose of testing, the crytpo algorithm we will be using will have a 4 bit plain text, 2 keys each 4 bit (total 8 bit), 2 rounds (key-mixing+substitution) to produce 4 bit cipher text.

```
(PlainText) (Key1)
    |          |
    |__________|
  (xor)
    |
    |
 (S-Box)     (Key2)
    |           |
    |___________|
  (xor)
    |
    |
 (S-Box)
    |
    |
(CiperText)
```

We will add some more rounds and see how to attacks fare against the now more secure algorithm at the end.

Linear Attacks

Lets say we had only one round in the above design instead of two.
We then woud have easily found out the key with a know-cipher-text attack.
We coud have passed the cipher text in reverse through the s-box, xor the value with the plain text (we know a set of plain-text and corresponding cipher-text pairs in a know-cipher-text attack) to get the key.

- Ths is not possible with this design, since after passing up through the second s-box we don't have the output from the first s-box to xor and find the key2, and for getting this ouput we need key1.
- We could obviously do brute force here, but the whole point of linear attack is to exploit the linearity of s-box and make a good guess of key1 and thus making a good guess of key2.
- So, for exploting linearity of an s-box we need to select a feature which cannot be affected by xor-operation (we don't want it to be disturbed by key-mixing operation).
- One such feature is parity. We take all possible inputs (4-bit -> 16) and run it through the s-box.

For each input, we calculate the input parity based on an input mask(4-bit) and output parity (output of s-box) based on an output mask(4-bit) and see if the parities are equal. We maintain the count of successful matches for each pair of input/ouput masks.

```
 (input)
    |____(inputMask)___(inputParity)
    |
(s-box)
    |
(output)
    |
    |____(outputMask)___(outputParity)
```

If the s-box has perfect non-linearity each entry should have 50% i.e is 8 matches.
The S-box we used for experimentation is not so linear (a more realistic setting)

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

Observations

- We see that with this s-box for a certain pair like (11,11), we get 14/16 matches.
- This means no matter the input, if we take the 0,1,3 bits for parity, there is a very good probability that output will also have the same parity calculated with the same bits.
- We can use this to make a better guess at selecting keys instead of trying all possible keys.

Attack

1. Take a known plain-text, cipher-text pair
2. Guess Key1
3. Calculate output of first s-box
4. Test how good this input is to second s-box by comparing its parity with that of cipher text
5. If better than current max score (better guess) use the input to guess Key2. (passing cipher text backwards)
6. Check this key pair with other known plain-text, cipher-text pairs, to see if it is correct.
7. If the key pair succeeds, correct, else continue.

Complexity

- For brute force attack (256 possible key pairs,16 plain-text,cipher text pairs, two rounds(key-mixing+s-box) leads to `256*16*2` units of computations.
- For our linear attack, initially it has to test against the parity feature which will take `16*16` units.
- But then it will check only with highly probable gueses of key1, leading to effectively lets say 2 candidates to be checked against 16 plain-text,cipher-text pairs over 2 rounds.
- This leads to a complexity of `16*16 + 16*2*2` units

The C-Code file `linear_cryptanalysis.c` attached does this analysis and prints the units of computations actually done for guessing the key.

Differential Attacks

Here we follow a similar scheme with the same design, but instead of trying to exploit based on a linear characteristic, we try to find a differential characteristic.

In linear characteristic, a feature was extracted from an input and was compared with same feature of the output.
In differential characteristic, a feature will be extracted based on two inputs and will be compared by extracting the feature from the two outputs.
So it depends on how the difference between two inputs is maintained as they are passed through the s-box.

- The interesting thing is that XOR-keymixing doesn't affect the differentials between two values.
- Lets say our differential feature is XOR of two input values
- We analyse the possible input pairs (16*16) and see how many have the same pair of (inputXOR,outputXOR)

```
(input1)      (input2)
    |____(xor)____|______(inputXOR)
    |              |
(s-box)         (s-box)
    |              |
(output1)     (output2)
    |____(xor)____|_____(outputXOR)
```

So, we have an SBOX again for our experiment.

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

Observations

- After running through all the possible input pairs, using the above s-box, we have 6 pairs having (inputXOR, outputXOR) as (4,7)
- We can remove the part of second s-box since it is reversible and doesn't add anything to security and bring the reversed cipher text till that point
- This means that if we choose a two plain-text such that their input differential (inputXOR) is 4 and outputXOR is 7, then the inputs to s-box in between can only be one of the 6 pairs
- We will also store the information about the pairs that have this XOR pair, which will be used later

Attack

We first need to find the plain-text,cipher-text pair which has the (inputXOR,outputXOR) as (4,7).
For this we need to be able to do a choosen-cipher-text attack.

1. Guess a random plain-text P0, generate P1 as P0 xor 4(our input differential) to get good input pair

2. Calculate the cipher texts of P0 and P1 to get C0 and C1, and find C0 xor C1
3. If it is equal to our choice of output differential (7), then is a good pair and we can stop, else we have to continue to till we get a good pair

Now that we have found a plain-text input pair satisfying our criterion, we need to find the key pair.

4. Inputs to s-boxes are of the form `P0 xor Key1` and `P1 xor Key1`
5. From the design we can see that the pair (4,7) is only possible with 6 input pairs at s-box as we calculated earlier.
6. Using each of the 6 input paris and xoring them with (P0,P1), we can get Key1 value. Similarly from back from cipher text we can get Key2 value.
7. We need to verify this Key Pair with other know plain-text, cipher-text pairs to see if they are correct. If they are not correct, we test with other 5 s-box input pairs.

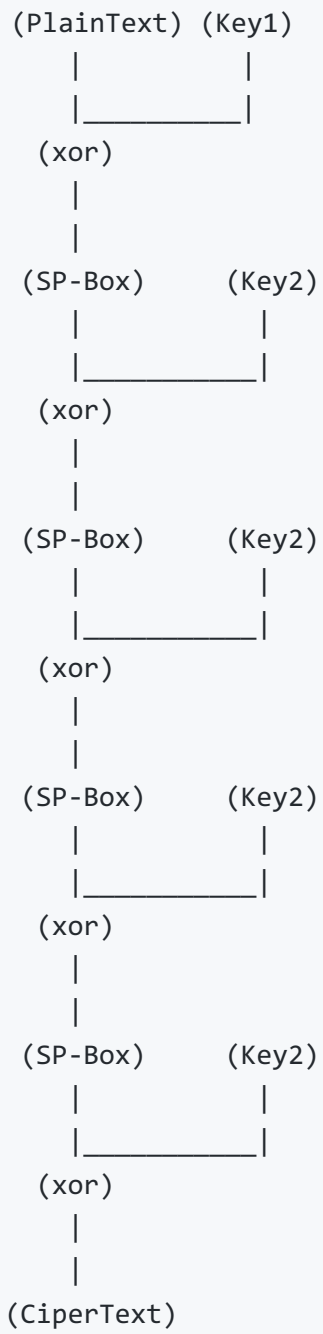
Complexity

- We can choose a (inputXOR,outputXOR) with less input pairs (less than 6 which was used above) to make the Key finding part faster
- But this will make the plain-text pair finding part slower, since we will have less input pairs satisfying the criterion
- This is tradeoff, but for the first part needs us to perform choosen-plain-text attack, we need decrease those operations, hence the choice
- Brute force is same as general case of `256*16*2` units
- In this case, we can say that we found out key pair ir `6` units after some variable no of choosen plain-text attacks

The C-code file `diffrential_cryptanalysis.c` attached might be helpful to get the exact no of computations needed to break the design.

Breaking stronger design like AES (SPN)

We can expand on differential analysis to attack on a much stronger block cipher having more rounds and permutation blocks.
This design for example has 4 rounds each performing substitution and permutaion operations which we view as a single function SP-Box.
The input is 8 bit, key is 32 bit (4*8), S-box is 4-bit.



We can use the same S-box like the one in the previous one and add a new permuation box.

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

P-Box just maps the 8 bits input to new positions to produce 8 bit output. (rearranging)

Input	Output
0	1
1	6
2	0
3	7
4	2
5	3
6	5
7	4

- To Perform brute force here, we need to check all possibilites of 32 bit keys.
- We don't actually need to try out the last key, so we need 2^24 which is still a huge number.

Attack

1. Similar to the previous differential analysis attack, we need to find the input, output differential characteristics.
2. This will now take 256 units (8*8)
3. We now have to create a path. We still only give importance to highest valued i/o pair, but now due to multiple rounds(3 in this case) we need to set the output of first round to be the input differential of second round and then soo on..

```
        val1                    val2                    val3
input1 -------> output1/input2 -------> ouput2/input3 ------> output2
       SP-Box 1                SP-BOX 2                SP-BOX 3
```

4. We want to max cost path (val1+val2+val3) to get increase the chance of getting a good pair in the next step
5. We need to have a good amount of know plain-text, cipher-text pairs, and find a good pair of plain-texts from these which satisfies the above path.
6. Now, we only have some possiblities for Keys, we can search in the just like before and find the right one.

C-Code for doing this and comparing it with brute force is given in the file `differential_spn.c`
It doesn't work all the time properly. Because we are searching for a good pair amond the existing know cipher-text data and not doing a choosen-cipher-text attack, we may not find a good pair sometimes.