TC Assignment 3
Submission Date - 21 February 2025

For this assignment consider the following toy cipher - TC[1]. TC is an iterated cipher with r rounds that operates on 16-bit blocks. For this assignment consider **r=5**.

Both the message and the ciphertext blocks will be 16 bits long, as will the $r+1$ round keys, which are independent and chosen uniformly at random. Encryption with TC is outlined in Algorithm 1 and illustrated in Figure 1.

It can be shown that each bit of the ciphertext depends on each bit of the plaintext after four rounds of encryption, and TC appears to have many of the features of a real-life block cipher. Note that in TC, there is no permutation $P[\cdot]$ in the rth round; in addition to having no cryptographic importance, it would serve as a distraction to our analysis.

Algorithm 1 The Toy Cipher-TC's Encryption Algorithm

1: **S-Box:**

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$S[x]$	6	4	c	5	0	7	2	e	1	f	3	d	8	a	9	b

2: **Permutation:**

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P[i]$	0	4	8	12	1	5	9	13	2	6	10	14	3	7	11	15

3: Set $u_0 = m$.
4: Set $r = 5$.
5: **for** $i := 1$ to $r-1$ **do**
6: XOR the round key k_{i-1} with u_{i-1}: $a_i = u_{i-1} \oplus k_{i-1}$.
7: Divide a_i into four nibbles: $a_i = A_0 \| A_1 \| A_2 \| A_3$.
8: Compute $S[A_0] \| S[A_1] \| S[A_2] \| S[A_3]$ where $S[\cdot]$ is defined above.
9: Write $S[A_0] \| S[A_1] \| S[A_2] \| S[A_3]$ as $y_{15} \dots y_0$.
10: Permute bit y_i to position j according to the permutation $j = P[i]$.
11: Set $u_i = y_{15} \| y_{11} \| \dots \| y_4 \| y_0$.
12: **end for**
13: **Last round:**
14: XOR the round key k_{r-1} with u_{r-1}: $a_r = u_{r-1} \oplus k_{r-1}$.
15: Divide a_r into four nibbles: $a_r = A_0 \| A_1 \| A_2 \| A_3$.
16: Compute $S[A_0] \| S[A_1] \| S[A_2] \| S[A_3]$ where $S[\cdot]$ is defined above.
17: Write $S[A_0] \| S[A_1] \| S[A_2] \| S[A_3]$ as $y = y_{15} \dots y_0$.
18: Output $y \oplus k_r$ as ciphertext.

The Assignment Parts

Part 1: Implement the toy cipher TC. Choose the round keys $k_i, 0 \le i \le 5$ randomly.

I have provided my python code at the end of this document. You may use it to verify your code. **Run the code in any online compiler and create your test cases.** If you find any mistake in my code, let me know.

Part 2: Construct the difference distribution table (DDT) of the S-box S of the toy cipher.

Part 3: Construct a differential trail $\Delta(x) \xrightarrow{4-rounds} \Delta(y)$ for the first 4 rounds of the toy cipher, with a high probability.

Part 4: Use the obtained differential distinguisher to recover the last round key k_r.

[1]The cipher is taken from the book - "The Block Cipher Companion"

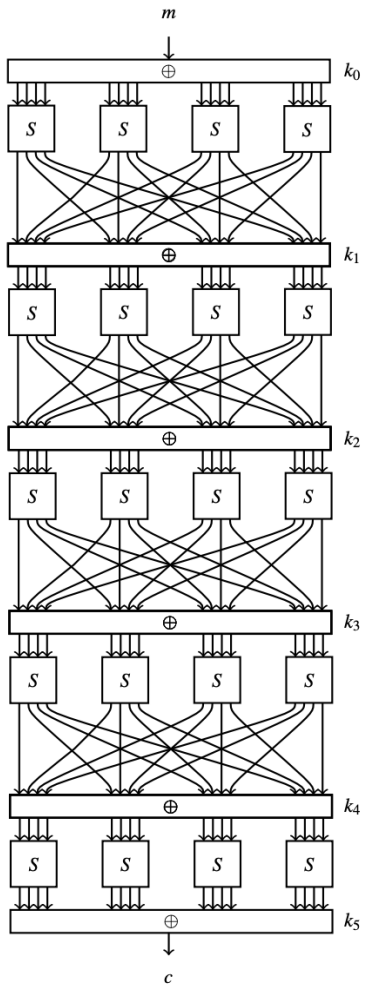

Figure 1: The toy cipher TC

Description of the assignment

The goal of this assignment is to mount a differential attack on **r=5** rounds of this toy cipher and recover the last round key k_r.

- First note that all components are linear with respect to the exclusive-or operation except the S-box.
- If we are trying to mount a differential attack, then the first thing we need to do is to find a characteristic which predicts the difference between two partially encrypted messages after $r-1=4$ rounds, i.e., the difference after the XORing of k_4, since XOR does not affect the difference propogation..
- If such a characteristic can be identified that holds with a sufficiently high probability, then, in principle, we should be able to find the round key k_r.

Description of *Part 3* - construction of the differential

To build characteristics over several rounds, one normally tries to find one-round characteristics (of high probability) that can be joined together. So, with TC, we will need to identify input differences to four S-boxes for which high-probability output differences exist. Then, since $P[\cdot]$ is linear, one can easily compute the output difference from one round. Let us denote by

$$(\alpha_1, \alpha_2, \alpha_3, \alpha_4) \xrightarrow{S} (\beta_1, \beta_2, \beta_3, \beta_4)$$

the input and output differences to four S-boxes, where α_i is the difference between the values A_i input to the relevant S-box and β_i is the difference between the output values $S[A_i]$. Since the action of $P[\cdot]$ is fixed, we can predict with certainty how the difference will evolve across the permutation

$$(\beta_1, \beta_2, \beta_3, \beta_4) \xrightarrow{P} (\gamma_1, \gamma_2, \gamma_3, \gamma_4)$$

where $\beta_1 \| \beta_2 \| \beta_3 \| \beta_4$ is the 16-bit input difference to $P[\cdot]$ and $\gamma_1 \| \gamma_2 \| \gamma_3 \| \gamma_4$ is the 16-bit output difference from $P[\cdot]$. As a result, over the full round R,

$$(\alpha_1, \alpha_2, \alpha_3, \alpha_4) \xrightarrow{R} (\gamma_1, \gamma_2, \gamma_3, \gamma_4),$$

which denotes a nibble-based, one-round characteristic for TC.

You know that an input difference of 0 gives an output difference of 0 with probability 1. Clearly, equal inputs to the S-box $S[\cdot]$ lead to equal outputs! Certainly, choosing all input differences to all four S-boxes to be 0 would lead to a one-round characteristic of probability 1. But this would be useless for cryptanalysis since we would, in effect, be encrypting the same message twice and getting the same ciphertext as a result.

So we deduce that a nontrivial one-round characteristic must have a maximum of three S-boxes taking a 0 input difference. It would seem to be a good strategy to look for a one-round characteristic for TC that has a 0 input to three S-boxes and an input difference f to one S-box, let's say the fourth.

Why f? You will see in the DDT that the biggest entry in the difference distribution table is for the input-output difference $f \to d$. So expressing this as a characteristic for the whole input block we have the characteristic

$$(0, 0, 0, f) \xrightarrow{S} (0, 0, 0, d)$$

that holds with probability $\frac{10}{16}$. We can immediately account for the action of $P[\cdot]$, which gives

$$(0, 0, 0, d) \xrightarrow{P} (1, 1, 0, 1)$$

and so

$$(0, 0, 0, f) \xrightarrow{R} (1, 1, 0, 1)$$

is a one-round characteristic for TC that holds with probability $\frac{10}{16}$.

So far so good. Now we can extend this characteristic by one additional round, and we must find characteristics across three S-boxes with a starting input difference 1. Similarly find a trail for upto 4 rounds.

> Note: Note that the above approach has been to add additional rounds by looking for the highest probability characteristic for each S-box from the DDT. However, this does not always lead to the highest probability characteristics over several rounds. A specific choice of difference might be optimal over one round but other cipher components (in this case $P[\cdot]$) might mean that much less optimal differences will need to be used later. You will notice that some other input difference, say $\Delta(x) = (0, 0, 2, 0)$, gives a 4-round differential with higher probability. Try to chose those difference which are not only high but also activates as less S-boxes as possible.

Description of *Part 4* - Key Recovery

Let us **assume** that in **Part 3**, you construct a differential of the form $\Delta(x) = (0, 0, 2, 0) \rightarrow (0, 0, 2, 0) = \Delta(y)$ with a probability say, $p = 2^{-12}$. Remember this is just an example, the values might be wrong. Do not follow this trail without verifying it yourself.

Also **assume** that you have an implementation of the 5-round TC with a fixed key $(k_1||k_2||k_3||k_4||k_5)$.

Now the key recovery step will consist of the following steps:

1. **Data collection phase**

   Encrypt 2^{12} pairs of plaintexts $(m, m' = m \oplus \Delta(x))$ and obtain the corresponding ciphertexts (c, c') (because i have assumed the trail probability to be 2^{-12}. The value will depend on your trails probability.).

   Remember these ciphertexts are the encryption of the plaintexts by the whole 5-round cipher TC.

2. **Filtering phase**

   Now, as per the differential, the difference after 4^{th} round is $(\Delta(y) = (0, 0, 2, 0)$. The last round only has $S - box$ and round key xor. So, in the last round the difference $(\Delta(y) = (0, 0, 2, 0))$ will propogate as follows:

   $$\Delta(y) = (0, 0, 2, 0) \xrightarrow{S} (0, 0, z, 0)$$

   where, according to the DDT of the S-box, z can take any values in $\{1, 2, 9, a\}$.

   Since the permutation $P[\cdot]$ is not used in the last round and XOR doesnot affect the propogation of the difference, these four difference must be a part of the ciphertexts.

   So, for each pair, you can check if the 12 bits of the XOR of the ciphertext pairs is equal to 0 or not. If it is not, it must be a wrong pair and can be discarded/filtered out.

   This process of discarding the wrong pairs is called *filtering*. Remember that a good filtering technique is essential t the success of differential attacks.

3. **Key recovery phase**

   Once you are left with a reduced set of ciphertetx pairs, do the following:

   (a) Initialize a set of counters $T_0, \ldots, T_{k-1}$, one for each possible last round key, and set them to zero.

   (b) For each pair of ciphertexts (c, c') left:
   - For each possible choice of the last round key k_r:
     * Compute the intermediate values $v = S^{-1}(c \oplus k_r)$ and $v' = S^{-1}(c' \oplus k_r)$. This allows us to determine $v \oplus v'$. Here S^{-1} is the inverse of the S-box S. You can construct the S^{-1} from the given S-box S.
     * If $v \oplus v' = \Delta(y)$, increment the counter T_{k_r}.

   (c) The counter with the maximum value is most likely the correct key.

Verify with the TC implementation if the recovered key is correct.

Listing 1: Encryption Algorithm for TC

```python
S_BOX = [0x6, 0x4, 0xC, 0x5, 0x0, 0x7, 0x2, 0xE, 0x1, 0xF, 0x3, 0xD, 0x8, 0xA, 0x9, 0xB]
P = [0, 4, 8, 12, 1, 5, 9, 13, 2, 6, 10, 14, 3, 7, 11, 15]

def substitute(nibble):
    return S_BOX[nibble]

def permute(bits):
    permuted = [0] * 16
    for i in range(16):
        permuted[P[i]] = bits[i]
    return permuted

def to_bits(val, size=16):
    return [(val >> i) & 1 for i in range(size - 1, -1, -1)]

def from_bits(bits):
    return sum(b << (15 - i) for i, b in enumerate(bits))

def encrypt(plaintext, round_keys):
    state = plaintext
    for i in range(4):
        state ^= round_keys[i]
        state_nibbles = [(state >> (4 * j)) & 0xF for j in range(4)][::-1]
        state_nibbles = [substitute(n) for n in state_nibbles]
        state_bits = to_bits(sum(n << (4 * j) for j, n in enumerate(state_nibbles)))
        state_bits = permute(state_bits)
        state = from_bits(state_bits)
    state ^= round_keys[4]
    state_nibbles = [(state >> (4 * j)) & 0xF for j in range(4)][::-1]
    state_nibbles = [substitute(n) for n in state_nibbles]
    state = sum(n << (4 * j) for j, n in enumerate(state_nibbles))
    return state ^ round_keys[5]

# Example usage
plaintext = 0x1234
round_keys = [0x1111, 0x2222, 0x3333, 0x4444, 0x5555, 0x6666]  # Example keys
ciphertext = encrypt(plaintext, round_keys)
print(f"Ciphertext: {ciphertext:04X}")
```