CS406 Project Report

G Lohith Reddy(210050054), G Adithya Rao(210050060)

May 2, 2023

1 Introduction

This is a basic implementation of a Substitution-Permutation Network (**SPN**) cipher, which takes as input a **16-bit** input block and has **4 rounds**. Each round consists of substitution, transposition, and key mixing.

The **get_sbox** function applies a substitution (**sbox**) to a 16-bit state and returns the result. The **sbox** and **sbox_inv** variables are used to perform the substitution operation. **sbox** is a dictionary that maps an input nibble (**4 bits**) to an output block. **sbox_inv** is the inverse of **sbox** and maps an output block to an input block.

2 Encryption

The **encrypt** function takes in two arguments **plaintext** and **k** which represent the 16-bit plaintext and the secret key respectively.

The function starts by initializing the **state** variable to the plaintext **pt**. It then generates the round keys by using the **keyGen** function to generate a 128-bit random seed, and then taking the first 80 bits of the **SHA-256** hash of that seed as the **'5'** 16-bit round keys.

- The **state** variable is XORed with the corresponding round subkey.
- The state variable is broken up into blocks and each block is substituted using the **sbox** dictionary.
- The blocks of the resulting state are permuted using the **pbox** dictionary taken.

After the first three rounds, the final round is performed where:

- The state variable is **XORed** with the 4^{th} round subkey.
- The state variable's nibbles are substituted using the **sbox** dictionary.
- The **state** variable is XORed with the **final** round subkey.

Finally, the encrypted ciphertext is returned.

3 Decryption

The **decrypt** function takes in two arguments **ciphertext** and **k** which represent the 16-bit ciphertext and the secret key respectively.

The function starts by initializing the **state** variable to the **ciphertext**. Now we divide the key into 5 parts denoting for **5 subkey mixings**(Since number of rounds are 4) this is the same key used for encryption of plaintext to ciphertext else we will not obtain the desired plaintext. The key of encryption is generated using **keyGen()** function which uses a 128-bit random seed, and then taking the first **80 bits** of the **SHA-256** hash of that seed as the **'5'** 16-bit round keys.

- The **state** variable is XORed with the 5^{th} subkey.
- The state variable is broken up into blocks and each block is substituted using the **sbox inverse** dictionary.(Sbox inverse is the inverse mapping of sbox mappings) For round number 3,2,1 we follow the following procedure:
- We will **XOR** with corresponding subkey value for the corresponding round.
- We then permute the state using the pbox dictionary as mentioned in encryption
- We then apply **get_sbox()** function on the blocks that are obtained after XORing and

Finally we XOR state with **subkey[0]** and return the output value which is the **plaintext**.

4 Linear cryptanalysis

4.1 Linear Approximations used to recover partial subkey values $[K_{5,5}...K_{5,8}]$ and $[K_{5,13}...K_{5,16}]$

To do this we concatenate linear approximations through multiple rounds. Linear approximations are given by the **linear approximation table** which is derived by testing all possible linear relationships between input and output bits for a given sbox.

e.g. Through 3 rounds

$S_{1,2} : X_1 \oplus X_3 \oplus X_4 = Y_2$, probBias = +4 (6,11 in LAT) (*1)
$S_{2,2} : X_2 = Y_2 \oplus Y_4$, probBias = -4 (4,5 in LAT) (*2)
$S_{3,2} : X_2 = Y_2 \oplus Y_4$, probBias = -4 (*3)
$S_{3,4} : X_2 = Y_2 \oplus Y_4$, probBias = -4 (*4)

Let U_i = sbox input in round i and $|U_i|$ = block size (16 bits), and V_i = sbox output in round i and $|V_i| = |U_i|$, then $U_1 = P \oplus K_1$ where P = plaintext block and K_1 is round 1 subkey.

Now thinking about individual bits in the state from round to round:

Using equation (1) we get $V_{1,6}$ = sixth state bit in round 1 (from sbox $S_{1,2}$)

$$\begin{aligned} V_{1,6} &= U_{1,5} \oplus U_{1,7} \oplus U_{1,8} (Y_2 = X_1 \oplus X_3 \oplus X_4) \\ &= (P_5 \oplus K_{1,5}) \oplus (P_7 \oplus K_{1,7}) \oplus (P_8 \oplus K_{1,8}) \quad (*5) \end{aligned}$$

With probability bias = 4 $\Rightarrow$ probability = $(4 + 8)/16 = 0.75$

Using equation (2) in round 2: $V_{2,6} \oplus V_{2,8} = U_{2,6}$ with $Pr = (-4+8)/16 = 0.25$

Combining (5) and (6) $V_{2,6} \oplus V_{2,8} = (P_5 \oplus K_{1,5}) \oplus (P_7 \oplus K_{1,7}) \oplus (P_8 \oplus K_{1,8})$ (6) By **Matsui's piling up lemma**, incorrectly assuming independence) With probability $0.5 + 2(0.75 - 0.5) \cdot (0.25 - 0.5) = \mathbf{3/8}$

Using (3) for round 3: $V_{3,6} \oplus V_{3,8} = U_{3,6}$ with $Pr = 0.25$ and: $V_{3,14} \oplus V_{3,16} = U_{3,14}$ with $Pr = \mathbf{0.25}$ (The permutation decides that we need this approximation)

$$U_{3,6} = V_{2,6} \oplus K_{3,6} \quad \text{and} \quad U_{3,14} = V_{2,8} \oplus K_{3,14}$$
$$V_{3,6} \oplus V_{3,8} \oplus U_{3,6} = 0 = V_{3,14} \oplus V_{3,16} \oplus U_{3,14}$$
$$\Rightarrow V_{3,6} \oplus V_{3,8} \oplus V_{2,6} \oplus K_{3,6} = 0 = V_{3,14} \oplus V_{3,16} \oplus V_{2,8} \oplus K_{3,14}$$
$$\text{with } \Pr = 0.5 + 2(0.25 - 0.5) \times (0.25 - 0.5)$$

$$U_{4,6} = V_{3,6} \oplus K_{4,6} \quad \text{and} \quad U_{4,8} = V_{3,14} \oplus K_{4,8}$$
$$\text{and} \quad U_{4,14} = V_{3,8} \oplus K_{4,14} \quad \text{and} \quad U_{4,16} = V_{3,16}$$
$$\Rightarrow U_{4,6} \oplus U_{4,8} \oplus U_{4,14} \oplus U_{4,14} \oplus U_{4,16} \oplus P_5 \oplus P_7 \oplus P_8 \oplus$$
$$(\text{Sum of } K \text{ bits for all 4 rounds}) = 0$$
$$\text{with } \Pr = 0.5 + [2(0.75 - 0.5) \times (0.25 - 0.5)] \oplus 3 = \mathbf{\frac{15}{32}}$$

Since sum over all $K_{bits} == 1|0$ we also know $U_{4,6} \oplus U_{4,8} \oplus U_{4,14} \oplus U_{4,14} \oplus U_{4,16} \oplus P_5 \oplus P_7 \oplus P_8 = 0$ must hold with a probability of either 15/32 or $1 - (15/32)$.

Thus we have a linear approximation of rounds 1-3 which holds with a bias of magnitude **1/32**.

Using this approximation it is possible to extract bits from subkey K_5. In particular, we partially decrypt the last round of the cipher. Next we'll see how to decrypt entire last round key.

4.2 Linear Approximations used to recover partial subkey values $[K_{5,1}...K_{5,4}]$ and $[K_{5,9}...K_{5,12}]$

We use the following approximations of the S-box:

$S12 : X1 \oplus X3 \oplus X4 = Y2$ with probability 12/16 and bias $+1/4$

$S22 : X2 = Y2 \oplus Y4$ with probability 4/16 and bias $-1/4$

$S32 : X2 = Y1 \oplus Y2 \oplus Y3$ with probability 10/16 and bias $+1/8$

$S34 : X2 = Y1 \oplus Y2 \oplus Y3$ with probability 10/16 and bias $+1/8$

Combining the linear approximations above, we get

$U_{4,2} \oplus U_{4,4} \oplus U_{4,6} \oplus U_{4,8} \oplus U_{4,10} \oplus U_{4,12} \oplus P_5 \oplus P_7 \oplus P_8 \oplus \Sigma K = 0$

where

$\Sigma K = K_{1,5} \oplus K_{1,7} \oplus K_{1,8} \oplus K_{2,6} \oplus K_{3,6} \oplus K_{3,14} \oplus K_{4,4} \oplus K_{4,6} \oplus K_{4,8} \oplus K_{4,10} \oplus K_{4,12} \oplus K_{4,14}$

and ΣK is fixed at either 0 or 1 depending on the key of the cipher. By application of the **Piling-Up Lemma**, the above expression holds with probability.

We start with the linear expression:

$$\frac{1}{2} + 2^3 \left(\frac{3}{4} - \frac{1}{2}\right) \left(\frac{1}{4} - \frac{1}{2}\right) \left(\frac{5}{8} - \frac{1}{2}\right)^2 = \frac{63}{128} \text{ (with a } \textbf{bias} \text{ of } -1/128)$$

To recover the 5th round key, we need at least $1/(-1/128)^2 = 16384$ plaintext-ciphertext pairs.

Since the bias obtained when recovering the partial subkey values $[K_{5,5} \dots K_{5,8}]$ and $[K_{5,13} \dots K_{5,16}]$ is much larger than the bias in the linear expression, we need more plaintext-ciphertext pairs. The linear expression of Equation 3 affects the inputs to S-boxes $S_{4,1}$, $S_{4,2}$, and $S_{4,3}$ in the last round. We have obtained the partial subkey values $[K_{5,5} \dots K_{5,8}]$ and will try all 256 values for the target partial subkey $[K_{5,1} \dots K_{5,4}, K_{5,9} \dots K_{5,12}]$ for each plaintext/ciphertext sample. For each partial subkey value, we increment the count whenever Equation 3 holds true, where we determine the value of $[U_{4,1} \dots U_{4,4}]$, $[U_{4,5} \dots U_{4,8}]$, and $[U_{4,9} \dots U_{4,12}]$ by running the data backwards through the target partial subkey and S-boxes $S_{4,1}$, $S_{4,2}$, and $S_{4,3}$. We simulated attacking our basic cipher by generating **20000** known plaintext/ciphertext values and using linear cryptanalysis. The partial subkey values $[K_{5,1} \dots K_{5,4}]$ and $[K_{5,9} \dots K_{5,12}]$ with the largest bias magnitude are **0x3** and **0xE** which correspond to the target partial subkey value **[0x3, 0xE]**, confirming that the attack has successfully derived the subkey bits.

Thus, we managed to recover the 5th round key = **0x38ea**.

4.3 Obtaining the 4th round key and entire key

Now as we can extract the output from the last second round we treat this as output as we have treated **ciphertext** till now. In this way we can find the key of fourth round similar to what we have done as above. Now we do this for all rounds to find entire key which was the primary goal of the project. The biases may vary for different subkey computations which depends upon the expressions we obtain from **Linear Approximation Table** so we need to find out the equations and biases and find out subkeys for different rounds.

For testing you can use the github account where our code is present here