Linear Cryptanalysis

210050092 & 210050096

S-BOX & P-BOX

S-BOX:

input	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
output	E	4	D	1	2	F	B	8	3	A	6	C	5	9	0	7

P-BOX:

input	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
output	1	5	9	13	2	6	10	14	3	7	11	15	4	8	12	16

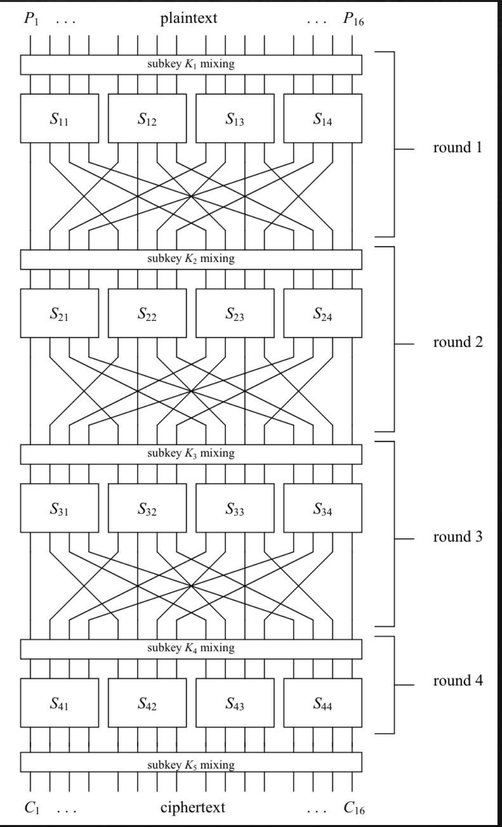
DMUX & MUX Functions

MUX: [A,B,C,D]-> 2^{12}*A + 2^{8}*B + 2^{4}*C + D

DMUX: N -> [A,B,C,D] s.t MUX[A,B,C,D]=N

Encryption

Key mixing is achieved by simple bit-wise exclusive-OR between the key bits associated with a round (referred to as a subkey) and the data block input to a round



Querying Plain texts & Cipher texts

```
inputs=[]
cipher_texts=[]
key=[]
for i in range(no_of_rounds):
    key.append(random.randint(0, 2**16 - 1))
print("Actual Key taken in last round ",demux(key[no_of_rounds-1]))
for i in range(no_of_inputs):
    m=random.randint(0, 2**16 - 1)
    c=encrypt(key,m,no_of_rounds)
    inputs.append(m)
    cipher_texts.append(c)
```

Piling Up Lemma

For n independent, random binary variables, $X_1, X_2, \ldots X_n$,

$$\Pr(X_1 \oplus \ldots \oplus X_n = 0) = 1/2 + 2^{n-1} \prod_{i=1}^{n} \varepsilon_i$$

or, equivalently,

$$\varepsilon_{1,2,..,n} = 2^{n-1} \prod_{i=1}^{n} \varepsilon_i$$

where $\varepsilon_{1,2,..,n}$ represents the bias of $X_1 \oplus \ldots \oplus X_n = 0$.

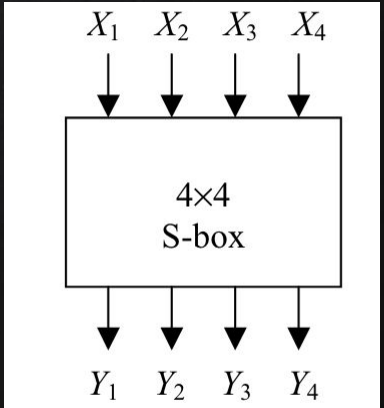
Note that if $p_i = 0$ or 1 for all i, then $\Pr(X_1 \oplus \ldots \oplus X_n = 0) = 0$ or 1. If only one $p_i = 1/2$, then $\Pr(X_1 \oplus \ldots \oplus X_n = 0) = 1/2$.

Linear Approximation Table (Bias Table)

A complete enumeration of all linear approximations of the S-box in our cipher is given in the linear approximation table.Each element in the table represents the number of matches between the linear equation represented in hexadecimal as "Input Sum" and the sum of the output bits represented in hexadecimal as "Output Sum" minus 8

Ex: $A_{5,11}$ represents the bias of the linear equation

$$X2 \oplus X4 \oplus Y1 \oplus Y3 \oplus Y4 = 0$$



Linear Approximation Table (Bias Table)

		Output Sum															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Input Sum	0	+8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	0	0	–2	–2	0	0	–2	+6	+2	+2	0	0	+2	+2	0	0
	2	0	0	–2	–2	0	0	–2	–2	0	0	+2	+2	0	0	–6	+2
	3	0	0	0	0	0	0	0	0	+2	–6	–2	–2	+2	+2	–2	–2
	4	0	+2	0	–2	–2	–4	–2	0	0	–2	0	+2	+2	–4	+2	0
	5	0	–2	–2	0	–2	0	+4	+2	–2	0	–4	+2	0	–2	–2	0
	6	0	+2	–2	+4	+2	0	0	+2	0	–2	+2	+4	–2	0	0	–2
	7	0	–2	0	+2	+2	–4	+2	0	–2	0	+2	0	+4	+2	0	+2
	8	0	0	0	0	0	0	0	0	–2	+2	+2	–2	+2	–2	–2	–6
	9	0	0	–2	–2	0	0	–2	–2	–4	0	–2	+2	0	+4	+2	–2
	A	0	+4	–2	+2	–4	0	+2	–2	+2	+2	0	0	+2	+2	0	0
	B	0	+4	0	–4	+4	0	+4	0	0	0	0	0	0	0	0	0
	C	0	–2	+4	–2	–2	0	+2	0	+2	0	+2	+4	0	+2	0	–2
	D	0	+2	+2	0	–2	+4	0	+2	–4	–2	+2	0	+2	0	0	+2
	E	0	+2	+2	0	–2	–4	0	+2	–2	0	0	–2	–4	+2	–2	0
	F	0	–2	–4	–2	–2	0	+2	0	0	–2	+4	–2	–2	0	+2	0

Attack to obtain the last sub key

- Generate C*(len(bias_table)) paths: Select len(bias_table) paths for each S-box, with a total of C' S-boxes in a round.

- Each path corresponds to a linear equation and a bias

Attack to obtain the last sub key:

- Sort the biases in descending order and iterate through the linear equations corresponding to them

- Then, brute force the respective subkey values and compute the bias for each subkey from the linear equation

Attack to obtain last complete sub key value

- Find the sub key which has bias closest to the chosen linear equation

- Iterate through the linear equations in their decreasing order of biases till you find the complete sub key

Finding the complete key

- ✘ Iterate the above process, replacing the ciphertext with s_inv[cipher + last subkey] at each iteration. Repeat until all keys are obtained.
- ✘ Remember to apply the P-box in all rounds except the last one


thank you