CS553: Cryptography

Assignment 6: Solutions

Rohit Das (11910230)

September 24, 2019

1. Linear Approximation Table

Python code (Python 3): LAT_s-box.py

```
from functools import reduce
from operator import xor
import numpy as np

sbox = {0x0: 0x5, 0x1: 0x4, 0x2: 0xd, 0x3: 0x1, 0x4: 0x2,
        0x5: 0xf, 0x6: 0x6, 0x7: 0x0, 0x8: 0x8, 0x9: 0xc,
        0xa: 0xb, 0xb: 0x9, 0xc: 0x7, 0xd: 0xe, 0xe: 0xa,
        0xf: 0x3}

def lat(sbox):
    lst1 = [format(i,'x') for i,j in sbox.items()]
    print("in\\out|",("{:>3}"*len(lst1[1:])).format(*lst1[1:]))
    print('-'*56)

    lat_dict = {} # to store count for each mask
    for alpha in range(1,16):
        for beta in range(1,16):
            lat_dict[beta] = 0

            for x,sx in sbox.items():
                x_arr = np.array([int(i) for i in # x . alpha
                        list('{:04b}'.format(x & alpha,'b'))])
                sx_arr = np.array([int(i) for i in # sx . beta
                        list('{:04b}'.format(sx & beta,'b'))])
                if reduce(xor,x_arr) == reduce(xor,sx_arr):
                    lat_dict[beta] += 1
```

```
lst2 = [str(int((j/16-0.5)*16)) for i,j in
lat_dict.items()] # eps = p - 1/2
lst2 = list(map(lambda b:b.replace('0','.'),lst2))

# formatting table
frmt = "{:>3}"*len(lst2)
print(str(format(alpha,'x')+'  |').rjust(7),
      frmt.format(*lst2))

lat(sbox)
```

in\out	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
1	.	-2	-2	.	4	-2	2	.	.	-2	-2	-4	.	2	-2
2	2	.	2	-4	2	.	-2	.	-2	.	-2	.	2	-4	-2
3	-2	2	.	4	2	2	.	4	-2	-2	.	.	2	-2	.
4	-2	6	.	.	-2	-2	.	-2	.	.	-2	-2	.	.	-2
5	-2	.	2	.	2	.	6	-2	.	2	.	2	.	-2	.
6	4	2	2	.	.	-2	2	2	-2	.	.	-2	-2	.	4
7	.	.	-4	.	.	4	.	-2	-2	2	-2	-2	-2	-2	2
8	.	2	-2	-2	2	.	.	4	4	2	-2	2	-2	.	.
9	.	.	-4	-2	-2	-2	2	.	.	-4	.	2	2	-2	2
a	-2	-2	.	2	.	-4	-2	.	2	2	.	-2	.	-4	2
b	2	.	-2	2	.	-2	.	.	-2	4	-2	2	4	2	.
c	-2	.	-2	-2	.	-2	.	2	-4	2	4	.	-2	.	-2
d	-2	2	.	-2	4	.	-2	-2	.	.	2	.	2	2	4
e	.	.	.	-2	-2	2	2	2	2	2	2	-4	4	.	.
f	-4	-2	2	-2	-2	.	.	2	-2	.	-4	.	.	2	2

Linear Approximation Table

The LAT above is generated for the S-box given below:

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

From the LAT generated, we can see that the characteristics $4 \xrightarrow{S} 2$ and $5 \xrightarrow{S} 7$ will have the highest probability of producing a linear approximation of the above Sbox with p $= \frac{1}{2} + \frac{6}{16}$.

2. Bi-directional LC of Sypher00C

16-bit key used = (6||f||e||c)

Python code (Python 3): LC_Sypher00C.py

```
from functools import reduce
from operator import xor
import numpy as np

sbox1 = {0x0: 0xf, 0x1: 0xe, 0x2: 0xb, 0x3: 0xc, 0x4: 0x6,
        0x5: 0xd, 0x6: 0x7, 0x7: 0x8, 0x8: 0x0, 0x9: 0x3,
        0xa: 0x9, 0xb: 0xa, 0xc: 0x4, 0xd: 0x2, 0xe: 0x1,
        0xf: 0x5}

keys = list('6fec') # generated using openssl rand -hex 2
mask = int('d',16) # mask d has highest bias

def sbox(p):
    if format(p,'x') not in [format(i,'x') for i in sbox1]:
        exit("Invalid literal")
    return sbox1[p]
```

```
def sbox_inv(c):
    for i,j in sbox1.items():
        if c == int(format(j,'x'),16):
            return int(format(i,'x'),16)


def Sypher00C(p,n):
    for i in range(n):
        p = sbox(p ^ int(keys[i],16))
    return p ^ int(keys[-1],16)


def lin_crypt_k3():
    sum = [0]*16
    print("k3 =",keys[-1])
    frmt = "{:>3}"*len(sum)
    print("Guesses",frmt.format(*[format(i,'x')
                                    for i in range(16)]))


    for p in range(16):
        c = Sypher00C(p,3)
        lst = []
        for k in range(16):
            y_ = sbox_inv(c ^ k)
            # mask & p and then changing to binary list
            p_ = np.array([int(i)
                for i in list('{:04b}'.format(p & mask,'b'))])

            # mask & y' and then changing to binary list
            c_ = np.array([int(i)
```

```
                for i in list('{:04b}'.format(y_ & mask,'b'))])

            # (d.m) xor (d.y')
            xored = reduce(xor,p_) ^ reduce(xor,c_)
            lst.append(xored)

        sum = [i+j for i,j in zip(lst,sum)]
        print((format(p,'x')+'-'+format(c,'x')).ljust(5),
              '|'.rjust(1),frmt.format(*lst))

    print("T0---> ",frmt.format(*[16-i for i in sum]))
    print("T1---> ",frmt.format(*sum))
    candidates = [i for i,j in enumerate(sum) if j == max(sum)
    or j == min(sum)]
    candidates.sort()
    print("Candidate keys:",*[format(i,'x') for i in
          candidates],'\n')

def lin_crypt_k0():
    sum = [0]*16
    print("k0 =",keys[0])
    frmt = "{:>3}"*len(sum)
    print("Guesses",frmt.format(*[format(i,'x')
                                  for i in range(16)]))

    for p in range(16):
        c = Sypher00C(p,3)
        lst = []
```

```
        for k in range(16):
            v_ = sbox(p ^ k)
            # mask & m' and then changing to binary list
            p_ = np.array([int(i)
                for i in list('{:04b}'.format(v_ & mask,'b'))])

            # mask & c and then changing to binary list
            c_ = np.array([int(i)
                for i in list('{:04b}'.format(c & mask,'b'))])

            # (d.v') xor (d.c)
            xored = reduce(xor,p_) ^ reduce(xor,c_)
            lst.append(xored)

        sum = [i+j for i,j in zip(lst,sum)]
        print((format(p,'x')+'-'+format(c,'x')).ljust(5),
              '|'.rjust(1),frmt.format(*lst))

    print("T0---> ",frmt.format(*[16-i for i in sum]))
    print("T1---> ",frmt.format(*sum))
    candidates = [i for i,j in enumerate(sum) if j == max(sum)
    or j == min(sum)]
    candidates.sort()
    print("Candidate keys:",*[format(i,'x') for i in
          candidates])

lin_crypt_k3()
lin_crypt_k0()
```

k_3 \ p-c	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0-d	0	1	0	1	1	0	0	0	1	0	1	1	0	1	0	1
1-b	1	1	0	1	1	0	1	0	1	0	1	0	0	0	0	1
2-e	1	0	1	0	0	0	0	1	1	1	0	1	1	0	1	0
3-1	1	0	1	0	0	1	0	0	0	1	1	1	1	0	1	0
4-c	0	1	0	1	1	0	1	1	1	0	0	0	0	1	0	1
5-7	1	1	1	0	0	1	0	1	0	1	0	1	0	0	1	0
6-2	0	1	0	1	0	0	1	0	1	1	1	0	0	1	0	1
7-3	0	1	0	1	1	1	1	0	0	0	1	0	0	1	0	1
8-9	0	1	1	1	1	0	1	0	1	0	1	0	0	1	0	0
9-4	0	1	1	1	1	0	1	0	1	0	1	0	0	1	0	0
a-a	1	1	1	0	0	1	0	1	0	1	0	1	0	0	1	0
b-8	0	1	0	0	1	0	1	0	1	0	1	0	0	1	1	1
c-f	0	1	0	1	0	0	1	0	1	1	1	0	0	1	0	1
d-0	0	1	0	1	1	0	0	0	1	0	1	1	0	1	0	1
e-6	1	1	0	1	1	0	1	0	1	0	1	0	0	0	0	1
f-5	0	1	0	0	1	0	1	0	1	0	1	0	0	1	1	1
$\sum T_0$	10	2	10	6	6	12	6	12	4	10	4	10	14	6	10	6
$\sum T_1$	6	14	6	10	10	4	10	4	12	6	12	6	2	10	6	10

The table generated for guesses of k_3.

From the table for k_3 we can observe that the probable candidates for k_3 are $\{1,c\}$ as their T_0 and T_1 values are the most imbalanced.

k_0 / p-c	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0-d	0	1	1	1	0	0	1	0	1	0	1	0	0	1	0	1
1-b	0	1	0	0	1	1	1	0	1	0	1	0	0	1	0	1
2-e	0	0	1	0	0	1	1	1	0	1	0	1	1	0	1	0
3-1	1	1	1	0	0	1	0	0	0	1	0	1	1	0	1	0
4-c	1	1	0	1	1	0	0	0	1	0	1	0	0	1	0	1
5-7	1	1	1	0	0	1	0	0	0	1	0	1	1	0	1	0
6-2	0	1	1	1	0	0	1	0	1	0	1	0	0	1	0	1
7-3	0	1	0	0	1	1	1	0	1	0	1	0	0	1	0	1
8-9	0	1	0	1	1	0	1	0	1	0	0	0	1	1	0	1
9-4	0	1	0	1	1	0	1	0	1	0	1	1	0	0	0	1
a-a	1	0	1	0	0	1	0	1	1	1	0	1	1	0	0	0
b-8	0	1	0	1	1	0	1	0	1	1	1	0	0	1	0	0
c-f	0	1	0	1	1	0	1	0	0	0	1	0	0	1	1	1
d-0	0	1	0	1	1	0	1	0	1	1	1	0	0	1	0	0
e-6	0	1	0	1	1	0	1	0	1	0	0	0	1	1	0	1
f-5	0	1	0	1	1	0	1	0	1	0	1	1	0	0	0	1
$\sum T_0$	12	2	10	6	6	10	4	14	4	10	6	10	10	6	12	6
$\sum T_1$	4	14	6	10	10	6	12	2	12	6	10	6	6	10	4	10

The table generated for guesses of k_0.

From the table for k_0 we can observe that the probable candidates for k_0 are $\{1,7\}$ as their T_0 and T_1 values are the most imbalanced. The guess of $k_0 = 6$, which is the exact value, does not have the most imbalanced sum as SNR is low for smaller bits.

To find k_1 and k_2, since we can verify k_0 and k_3 from the above candidates, the cipher will be reduced to Sypher00A. Then we can perform linear cryptanalysis on them using the characteristic $\alpha \xrightarrow{S} \beta = p$. We can reuse d as the mask for both since it has the highest bias as seen in the LAT.

3. State-Meant (AES)

Plaintext to work on: "ROHIT DAS". Padding scheme used: ANSIX9.23

The initial state is derived as follows (padding already in HEX):

R	T	S	00
O		00	00
H	D	00	00
I	A	00	07

$\xRightarrow[\text{with ASCII}]{\text{Replacing}}$

82	84	83	00
79	32	00	00
72	68	00	00
73	65	00	07

$\xRightarrow[\text{of ASCII}]{\text{HEX code}}$

52	54	53	00
4F	20	00	00
48	44	00	00
49	41	00	07

3..1 ShiftRows

After applying ShiftRows, the state is as follows:

52	54	53	00
4F	20	00	00
48	44	00	00
49	41	00	07

$\xRightarrow[\text{by 0,1,2 and 3 places}]{\text{Shifting rows}}$

52	54	53	00
20	00	00	4F
00	00	48	44
07	49	41	00

3..2 SubBytes

After applying SubBytes, the state is as follows:

52	54	53	00
20	00	00	4F
00	00	48	44
07	49	41	00

$\xRightarrow[\text{bytes from S-Box}]{\text{Substituting}}$

00	20	ED	63
B7	63	63	63
63	63	52	1B
C5	3B	83	63