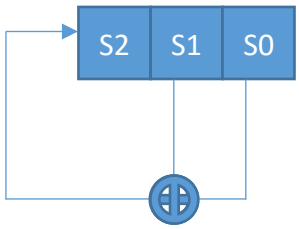
Thomas Mackintosh
ECE 4802
Project 2

1. LFSR
    a. The degree of the key stream generator is 3.
    b. The initialization vector is 0010111.
    c. The feedback coefficients are (0,1,1).



    d. The key is 00101110010111001010111010111. We don't use part of the plaintext as the key because if you know even a very small amount of the plaintext you could possibly get the entire key and message back with little effort.
2. Problem 2.11
    a. The initialization vector is 111111.
    b. The feedback coefficients are (0,0,0,0,1,1).
    c. The program to find the whole sequence and plaintext, 2_11.py, is below:

```
#CT observed J5A0EDJ2B
from collections import deque
#from bitarray import bitarray

init_vec = [1,1,1,1,1,1]
ciphertext = ['J',5,'A',0,'E','D','J',2,'B']
key =[]

cipherbin = []

#GENERATES KEY
key.insert(0,'1')
for j in range(0,44):
    a = int(init_vec[4]) ^ int(init_vec[5])
    init_vec.insert(0,a)
    init_vec.pop()

    key.append(str(init_vec[5]))



#GENERATE BITSTREAM FROM CIPHERTEXT
for j in range(0,9):
    if type(ciphertext[j]) == str:
        cipherbin.append(str(format((ord(ciphertext[j])-ord('A')),'05b')))
    else:
        cipherbin.append(str(format((ord(str(ciphertext[j])) - 22),'05b')))

#print "The cipher " +'{}{}{}{}{}{}{}{}{}'.format(*cipherbin)

#XOR THE CIPHER AND THE KEY

a = "0b"+''.join(key)
b = "0b"+''.join(cipherbin)


```

```
    39.
    40. plainbin = str(bin(int(a.strip(""),2) ^ int(b.strip(""),2)))
    41.
    42.
    43. print "Key       0b"+'{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}\
    44. {}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}'.format(*key)
    45. print "Cipher    0b"+'{}{}{}{}{}{}{}{}{}{}{}{}'.format(*cipherbin)
    46. print "Plaintext " +plainbin
    47.
    48.
    49. u0 = chr(int(("0b"+''.join(plainbin[2:7])).strip(""),2)+ord("A"))
    50. u1 = chr(int(("0b"+''.join(plainbin[7:12])).strip(""),2) + 65)
    51. u2 = chr(int(("0b"+''.join(plainbin[12:17])).strip(""),2)+ord("A"))
    52. u3 = chr(int(("0b"+''.join(plainbin[17:22])).strip(""),2) + 65)
    53. u4 = chr(int(("0b"+''.join(plainbin[22:27])).strip(""),2)+ord("A"))
    54. u5 = chr(int(("0b"+''.join(plainbin[27:32])).strip(""),2)+ord("A"))
    55. u6 = chr(int(("0b"+''.join(plainbin[32:37])).strip(""),2)+ord("A"))
    56. u7 = chr(int(("0b"+''.join(plainbin[37:42])).strip(""),2) + 65)
    57. u8 = chr(int(("0b"+''.join(plainbin[42:47])).strip(""),2)+ord("A"))
    58.
print u0,u1,u2,u3,u4,u5,u6,u7,u8

OUTPUT: Key 0b11111100000100001100010100111101000111001001 0
Cipher      0b01001111111000001101000100000110100111100000001
Plaintext   0b10110011111010001011001110011000000100000100 11
W P I W O M B A T
```

d. Australia.

e. We performed a known-plaintext attack.

3. The implementation of DES S-box S2 and the SAC algorithm is shown below in SAC.py:

```
1  def S2(x): #Implementation of S Box 2, returns value from row
2
3      row0 = [15,1,8,14,6,11,3,4,9,7,2,13,12,0,5,10]
4      row1 = [3,13,4,7,15,2,8,14,12,0,1,10,6,9,11,5]
5      row2 = [0,14,7,11,10,4,13,1,5,8,12,6,9,3,2,15]
6      row3 = [13,8,10,1,3,15,4,2,11,6,7,12,0,5,14,9]
7      s2 = 0
8      x = '0b'+bin(x^2**6)[3:]
9      if x[2] == "0":
10         if x[7] == "0":
11             y = int(x[3:7],2)
12             s2 = row0[y]
13         if x[7] == "1":
14             y = int(x[3:7],2)
15             s2 = row1[y]
16
17     if x[2] == "1":
18         if x[7] == "0":
19             y = int(x[3:7],2)
20             s2 = row2[y]
21         if x[7] == "1":
22             y = int(x[3:7],2)
23             s2 = row3[y]
24
25
26     return s2;
27
28
29 def SAC(x):
30     Sum = 0
31     output = [0,0,0,0,0,0]
32
33     for i in range(0,6):
34         output[i] = S2(x) ^ S2(x^2**i)
35     return output;
36
37 sum0 = 0
38 sum1 = 0
39 sum2 = 0
40 sum3 = 0
41
42 for j in range(2**6):
43     bit0 = 0; bit1 = 0; bit2 = 0; bit3 = 0
```

```
    arr = SAC(j)

    for i in range (0,6):
        bit0 += int(bin(arr[i]^(2**4))[6])
        bit1 += int(bin(arr[i]^(2**4))[5])
        bit2 += int(bin(arr[i]^(2**4))[4])
        bit3 += int(bin(arr[i]^(2**4))[3])
    sum0 += (bit0 % 2)
    sum1 += (bit1 % 2)
    sum2 += (bit2 % 2)
    sum3 += (bit3 % 2)

print sum0, sum1, sum2, sum3
```

This prints the sums of 32, 32, 36, and 32. Since each of these sums is greater than or equal to 2^5 = 32, then the s-box passes the SAC.

4. Below is the code used to attempt to make an exhaustive key search of the provided plaintext and ciphertext. The code failed to find the key within the keyspace.

```
from Crypto.Cipher import DES
import codecs

pt = "48656c6c6f212121"
ct = "d52bd481f21e25a1"

pt = pt.decode("hex")
ct = ct.decode("hex")

iv = "00000000".decode("hex")
key = "0000000000000000".decode("hex")

for x in range(4**8):
    cipher = DES.new(key^x, DES.MODE_ECB)
    msg = cipher.encrypt(pt)
    if msg == ct:
      break
```