Thomas Mackintosh

ECE 4802

Project 2

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

S0

S1

S2

* 1. The key is 0010111001011100101110010111. 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.

1. Problem 2.11
   1. The initialization vector is 111111.
   2. The feedback coefficients are (0,0,0,0,1,1).
   3. The program to find the whole sequence and plaintext, 2\_11.py, is below:
2. #CT observed J5A0EDJ2B
3. **from** collections **import** deque
4. #from bitarray import bitarray
6. init\_vec = [1,1,1,1,1,1]
7. ciphertext = ['J',5,'A',0,'E','D','J',2,'B']
8. key =[]
10. cipherbin = []
12. #GENERATES KEY
13. key.insert(0,'1')
14. **for** j **in** range(0,44):
15. a = int(init\_vec[4]) ^ int(init\_vec[5])
16. init\_vec.insert(0,a)
17. init\_vec.pop()
19. key.append(str(init\_vec[5]))


23. #GENERATE BITSTREAM FROM CIPHERTEXT
24. **for** j **in** range(0,9):
25. **if** type(ciphertext[j]) == str:
26. cipherbin.append(str(format((ord(ciphertext[j])-ord('A')),'05b')))
27. **else**:
28. cipherbin.append(str(format((ord(str(ciphertext[j])) - 22),'05b')))
30. #print "The cipher " +'{}{}{}{}{}{}{}{}{}'.format(\*cipherbin)
32. #XOR THE CIPHER AND THE KEY
34. a = "0b"+''.join(key)
35. b = "0b"+''.join(cipherbin)


39. plainbin = str(bin(int(a.strip(""),2) ^ int(b.strip(""),2)))

42. **print** "Key       0b"+'{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}\
43. {}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}'.format(\*key)
44. **print** "Cipher    0b"+'{}{}{}{}{}{}{}{}{}'.format(\*cipherbin)
45. **print** "Plaintext " +plainbin

48. u0 = chr(int(("0b"+''.join(plainbin[2:7])).strip(""),2)+ord("A"))
49. u1 = chr(int(("0b"+''.join(plainbin[7:12])).strip(""),2) + 65)
50. u2 = chr(int(("0b"+''.join(plainbin[12:17])).strip(""),2)+ord("A"))
51. u3 = chr(int(("0b"+''.join(plainbin[17:22])).strip(""),2) + 65)
52. u4 = chr(int(("0b"+''.join(plainbin[22:27])).strip(""),2)+ord("A"))
53. u5 = chr(int(("0b"+''.join(plainbin[27:32])).strip(""),2)+ord("A"))
54. u6 = chr(int(("0b"+''.join(plainbin[32:37])).strip(""),2)+ord("A"))
55. u7 = chr(int(("0b"+''.join(plainbin[37:42])).strip(""),2) + 65)
56. u8 = chr(int(("0b"+''.join(plainbin[42:47])).strip(""),2)+ord("A"))

**print** u0,u1,u2,u3,u4,u5,u6,u7,u8

OUTPUT: Key 0b111111000001000011000101001111010001110010010

Cipher 0b010011111100000110100010000011010011110000001

Plaintext 0b101100111101000101100111001100000010000010011

W P I W O M B A T

1. Australia.
2. 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
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]
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]

26. **return** s2;

29. **def** SAC(x):
30. Sum = 0
31. output = [0,0,0,0,0,0]
33. **for** i **in** range(0,6):
34. output[i] = S2(x) ^ S2(x^2\*\*i)
35. **return** output;
37. sum0 = 0
38. sum1 = 0
39. sum2 = 0
40. sum3 = 0
42. **for** j **in** range(2\*\*6):
43. bit0 = 0; bit1 = 0; bit2 = 0; bit3 = 0
44. arr = SAC(j)
46. **for** i **in** range (0,6):
47. bit0 += int(bin(arr[i]^(2\*\*4))[6])
48. bit1 += int(bin(arr[i]^(2\*\*4))[5])
49. bit2 += int(bin(arr[i]^(2\*\*4))[4])
50. bit3 += int(bin(arr[i]^(2\*\*4))[3])
51. sum0 += (bit0 % 2)
52. sum1 += (bit1 % 2)
53. sum2 += (bit2 % 2)
54. 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.

1. **from** Crypto.Cipher **import** DES
2. **import** codecs
4. pt = "48656c6c6f212121"
5. ct = "d52bd481f21e25a1"
7. pt = pt.decode("hex")
8. ct = ct.decode("hex")
10. iv = "00000000".decode("hex")
11. key = "0000000000000000".decode("hex")
13. **for** x **in** range(4\*\*8):
14. cipher = DES.new(key^x, DES.MODE\_ECB)
15. msg = cipher.encrypt(pt)
16. **if** msg == ct:

**break**