Başak Amasya – 26628

CS411 – HW3 – SOLUTIONS

1) Please refer to the Q1_student.py for the solution. My a = '11011111' and b = '00011101'.

a) I used BitVector module of Python. I converted given bit strings to BitVector forms. I multiplied a and b with respect to irreducible polynomial ('100011011') in GF(2^8) using gf_multiply_modular function.

C = '01110011'

b) I used gf_MI function of BitVector to calculate a inverse with respect to irreducible polynomial ('100011011') in GF(2^8).

A inverse = '01101011'.

2) Please refer to the hw3_q2.py for the solution.

I used correlation attack approach for the solution. In the end, we have to perform exhaustive search but the more we can narrow down the possible initial states, the better. Firstly, I initialized the connection polynomials and constructed all possible states for each one (all permutations for bit strings of length 14, 17 and 11). Then I generated all possible keystreams for each initial state with corresponding connection polynomials using the LFSR function given in the helper code of hw2. I checked the correlation between those keystreams and output z. z's length is 110, so I assumed that if more than 75 bits are same in same indexes, then these streams are correlated. For LFSR2, there were no correlated streams above the threshold, so I decided that LFSR2 does not show correlation with the output. There were 2 possible keystreams for LFSR1 and 3 for LFSR3, so I tried them with all possible keystreams for LFSR2. If my assumption wouldn't have worked out, I was going to lower down the threshold for LFSR1 and LFSR3 or change my approach. With the combining function F(x1, x2, x3) = x1x2 ⊕ x2x3 ⊕ x3, I obtained the output z when:

Initial state of LFSR1 is [1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1]
Initial state of LFSR2 is [0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0]
Initial state of LFSR3 is [1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0]

It should be noted that checking the correlations for each keystream take time, but this is still much more efficient solution then using just exhaustive search for all possible combinations for 3 of the LFSRs. Corresponding keystreams for each LFSR with these initial states can be found in the hw3_q2.py as well.

3)

a) 79, 85 and 97 are relatively prime to each other. So, the linear complexity can be directly computed as: 79x85 + 79x97 + 85x97 + 79x85x97 = 673978.

Since they are maximum-length LFSR sequences, they should generate the maximum periods. Period of the output sequence can be computed as: (2^79 – 1) x (2^85 – 1) + (2^79 – 1) x (2^97 – 1) + (2^85 – 1) x (2^97 – 1) + (2^79 – 1) x (2^85 – 1) x (2^97 – 1)

b) Nonlinearity degree is maximum order of terms in algebraic form which is x1&x2&x3 in F. That's why, this function has nonlinear order of 3.

Truth Table of F:

When x1 is 0 x2 is 0 and x3 0: F is 0
When x1 is 0 x2 is 0 and x3 1: F is 0
When x1 is 0 x2 is 1 and x3 0: F is 0
When x1 is 0 x2 is 1 and x3 1: F is 1
When x1 is 1 x2 is 0 and x3 0: F is 0
When x1 is 1 x2 is 0 and x3 1: F is 1
When x1 is 1 x2 is 1 and x3 0: F is 1
When x1 is 1 x2 is 1 and x3 1: F is 0

This function gives 5 zeros and 3 ones in 2^3=8 combinations of x1, x2, x3. Number of zeros and ones should be equal in the truth table for the combining function to be balanced. So, we can say that function is unbalanced.

x1 and F $\rightarrow$ 5/8

x2 and F $\rightarrow$ 5/8

x3 and F $\rightarrow$ 5/8

Out of 8 possible values, 5 are same between x1 and F, x2 and F, x3 and F. We can also observe that F gives 1 when 2 of x1, x2 and x3 is 1. In other words, for all x1, x2 and x3 separately, if the input is 0, 75% of the time F gives 0. That's why, we can find statistical dependencies between LFSR sequences and output sequence. They seem correlated.

We know that a good combining function should be balanced, have no statistical dependence, no correlation between LFSR sequences and output sequence and should be highly nonlinear. Since it's not balanced and we observe a correlation with x1, x2 and x3, we can interpret that F is not really a good combining function.

4) ShiftRow and Mixcolumn layers belong to the diffusion layer. Without them the effect of changing one byte in the plaintext will not be diffused to another bytes, it will only affect one byte. Since key length is 128-bit which corresponds to 16 bytes, AES will correspond to 16 substitution ciphers with size 8 (since 1 byte=8 bits). For implementing the byte substitution layer of the AES, we use a lookup table with 256 entries (entries are bytes). Similarly, we can do a chosen plaintext attack to break the byte substitution layer of the AES. Our chosen plaintexts will contain all same values, from 0 to 255. So that we can find all byte value mappings.

5) Ci is corrupted. Since Ci does not arrive, we can use Ci-1 instead. Assuming Ci-1 and Ci+1 are not corrupted, decryption of Ci+2 will not be corrupted. That's why, only 2 blocks decrypt incorrectly if the ciphertext block C_i is corrupted during transmission. Only Pi and Pi+1 will be corrupted. Because they use Ci in their decryption.

$P_i = D_K(\mathbf{C_i}) \oplus C_{i-1}$ and $P_{i+1} = D_K(C_{i+1}) \oplus \mathbf{C_i}$