Student Number: 400445340
Date: Thursday, September 26^{th}, 2024
Macid: modyd1

COMPSCI 4CR3

Assignment 1 — Dev Mody

Question 1:

Recall that a Linear Feedback Shift Register (LFSR) of degree n is given by a recurrence relation $a_n = p_{n-1}a_{n-1} + \cdots + p_1a_1 + p_0a_0$ where $p_{n-1}, \ldots, p_1$ are called the coefficients. An LFSR (mod 2) is determined by a sequence of coefficient bits. For example, the sequence 110101 represents an LFSR of degree 6 in which $p_0 = 1, p_1 = 1, p_2 = 0, p_3 = 1, p_4 = 0, p_5 = 1$, so, the recurrence for this LFSR is

$$a_6 = a_5 + a_3 + a_1 + a_0 \pmod 2.$$

For the initial value of $a_0a_1 \ldots a_5 =$ 011001, this LFSR produces the output

$$a_0a_1a_2a_3 \cdots = 011001000010011001000010011001 00\ldots$$

Alice and Bob want to communicate securely using a stream cipher. They use a stream cipher that is based on an LFSR, i.e., the key bits are generated by an LFSR. They choose an LFSR of degree 20 given by coefficients 11010100010001101010 and initial value 00110110010101101111. The coefficients are the secret key used by both parties, but the initial value is public.

→ We first need to generate the key stream before we encrypt.

The initial seed value of our LFSR is: 00110110010101101111

→ This is the function to generate the key stream:

```python
def lfsr_key_stream_generator (coefficients : list, initial : list, num_bits : int) -> list:
    lfsr = initial[:] # Copies initial seed
    key_stream = [] # Result Key Stream

    # Iterates over the number of bits
    for _ in range(num_bits):
        new_bit = 0 #Value of Feedback bit
        for i in range(len(initial)):
            if coefficients[i] == 1:
                new_bit ^= lfsr[i]

        key_stream.append(lfsr.pop(0)) #Adds to the key stream
        lfsr.append(new_bit) # Sets feedback bit

    return key_stream
```

a) (25 points) Perform the steps of the protocol on the plaintext

00111101011100111000000011100110001001101011 0110100.

The protocol consists of Alice generating the key bits and encrypting the plaintext, and then Bob generating the key bits and decrypting the ciphertext.

Refer to Stream_cipher_lfsr.py

Some of the steps are performed in Python.

The sequence of coefficients is 11010100010001101010

We must first construct a recurrence relation of the form:

$$a_n = p_{n-1}a_{n-1} + \cdots + p_1a_1 + p_0a_0$$ where n is the degree.

Reverse coefficients sequence: 01010110001000101011

$$\therefore\ a_{20} = a_{18} + a_{16} + a_{14} + a_{13} + a_9 + a_5 + a_3 + a_1 + a_0 \pmod 2$$

The length of the plaintext is 50 so we must generate a key stream of the same length.

Therefore, if we pass in our list of coefficients [1,1,0,1,0,1,0,0,0,1,0,0,0,1,1,0,1,0,1,0] our initial seed [0,0,1,1,0,1,1,0,0,1,0,1,0,1,1,0,1,1,1,1] and our desired key length of 50 bits, we obtain the secret key:

00110110010101101111111010110111110111001000001101

→ Now that we have successfully achieved our key stream, we must compute the bit-by-bit XOR of the key with the plaintext to generate the ciphertext. Similarly, we compute the same operation with the ciphertext to achieve the original plaintext.

Function:

```python
# This function either encrypts or decrypts plain/cipher text by performing XORs bit by bit
def xor_cipher (input : list, key_stream : list) -> list:
    result = []
    for i in range(len(input)):
        result.append(input[i] ^ key_stream[i])
    return result
```

Applying the OG plaintext to compute the ciphertext, we obtain:

00001011001001010111111110 1111011100100110010 111001

Applying the operation to compute the original plaintext, we obtain:

0011110101 11 00111000000 0111 0011000100 11 0101101 10100

As the plaintext is identical to the decrypted text, this procedure works.

b) (25 points) Suppose Trudy somehow gains access to the output of the key stream (i.e., the output of the LFSR). Write down the steps Trudy would follow to compute the (secret) coefficients of the LFSR. Be sure to consider how many key bits Trudy would need to successfully perform this computation. Here, you will generate the necessary number of key bits using the LFSR described above.

→ In this case, Trudy's objective is to determine the feedback coefficients of the LFSR used to compute the full key stream. This would then be used to decrypt the public ciphertext into plaintext as seen in a).

→ Let's analyze what Trudy already knows:

1. Key Stream: Trudy has access to some bits of the key stream generated by the LFSR
2. Degree of LFSR: From the knowledge of the initial public seed, Trudy knows the degree based on the number of flip-flops present
3. Ciphertext generated via encrypting plaintext with the key stream

→ Here are the steps Trudy must follow to obtain the Feedback Coefficients:

1. Trudy first needs at least $2n$ consecutive key bits from the key stream, where n is the degree of the LFSR. This is because the LFSR has n unknown coefficients and to solve for them, Trudy must form n independent linear equations. Using Linear Algebra, we know that to ensure these equations are independent, Trudy will need $2n$ key bits. (Ex: For our example in 1a), Trudy needs $2 \cdot 20 = 40$ key bits at least)
2. The next thing to do is to set up a system of linear equations using the key stream.
   By the recurrence relation: $a_n \equiv p_{n-1}a_{n-1} + \cdots + p_1a_1 + p_0a_0 \pmod 2$ where $a_0, a_1, \ldots, a_{n-1}$ are the known key stream bits.
   Trudy can generate n equations for $i \in \mathbb{Z}_n$

$$i=0 \quad a_n = p_{n-1}a_{n-1} + \cdots + p_1a_1 + p_0a_0 \mod 2$$
$$i=1 \quad a_{n+1} = p_{n-1}a_n + \cdots + p_1a_2 + p_0a_1 \mod 2$$
$$\vdots \qquad\qquad \vdots$$
$$i=n-1 \quad a_{2n-1} = p_{n-1}a_{2n-2} + \cdots + p_1a_n + p_0a_{n-1} \mod 2$$

We can make a corresponding matrix equation $A\vec{x} = \vec{b}$ where A represents the outputted key streams, $\vec{x}$ represents the desired feedback coefficients and $\vec{b}$ represents the known output bits.

$A\vec{x} = \vec{b}$ $\Rightarrow$ This is the augmented matrix

$\vec{x} = \begin{bmatrix} P_{19} \\ P_{18} \\ P_{17} \\ \vdots \\ P_1 \\ P_0 \end{bmatrix}$

$A =$

```
[[1 1 1 0 1 1 0 1 0 0 1 1 0 1 1 0 0]
 [1 1 1 1 1 0 1 1 0 1 0 0 1 1 0 1 1 0]
 [1 1 1 1 1 1 0 1 1 0 1 0 1 0 0 1 1 0 1]
 [1 1 1 1 1 1 1 0 1 1 0 1 0 1 0 0 1 1 0]
 [0 1 1 1 1 1 1 1 0 1 1 0 1 0 1 0 0 1 1]
 [1 0 1 1 1 1 1 1 1 0 1 1 0 1 0 1 0 0 1]
 [0 1 0 1 1 1 1 1 1 1 0 1 1 0 1 0 1 0 0]
 [1 0 1 0 1 1 1 1 1 1 1 0 1 1 0 1 0 1 0]
 [1 1 0 1 0 1 1 1 1 1 1 1 0 1 1 0 1 0 1]
 [0 1 1 0 1 0 1 1 1 1 1 1 1 0 1 1 0 1 0]
 [1 0 1 1 0 1 0 1 1 1 1 1 1 1 0 1 1 0 1]
 [1 1 0 1 1 0 1 0 1 1 1 1 1 1 1 0 1 1 0]
 [1 1 1 0 1 1 0 1 0 1 1 1 1 1 1 1 0 1 1]
 [1 1 1 1 0 1 1 0 1 0 1 1 1 1 1 1 1 0 1]
 [0 1 1 1 1 0 1 1 0 1 0 1 1 1 1 1 1 1 0]
 [1 0 1 1 1 1 0 1 1 0 1 0 1 1 1 1 1 1 1]
 [1 1 0 1 1 1 1 0 1 1 0 1 0 1 1 1 1 1 1]
 [1 1 1 0 1 1 1 1 0 1 1 0 1 0 1 1 1 1 1]]
```

$\vec{b} =$

```
[0
 0
 0
 0
 0
 0
 0
 0
 0
 0
 0
 0
 0
 0
 0
 0
 0]
```

3 Solve the systems of equations using Gaussian Elimination mod 2, Matrix Inversion or another algorithms to solve

b) For Gaussian Elimination: Row reduce matrix to RREF using row operations (adding/subtracting rows mod 2 = XOR rows) and then use back substitution to solve for feedback coefficients

→ Please refer to stream_cipher_lfsr.py as I have included a f° for this attack.

↓

The computed feedback coefficients are the same

c) (25 points) Instead of gaining access to the key stream, suppose Trudy gains access to the plaintext. Can Trudy still carry out the attack described in Part (b)? Explain.

Suppose Trudy instead gains access to the entire plaintext instead of gaining access to some portion of the generated keystream. Then Trudy would be able to derive the keystream, given that the ciphertext is public. By understanding cipher = plain $\oplus$ keystream and the property of XOR: $(A \oplus B) \oplus A = B$, we can say that keystream = plain $\oplus$ cipher. Now that we have the entire key, we can compute the feedback coefficients using the same procedure from 2 b). This works because the length of the key would be greater than 2·degree at least for the information provided in the question.

Question 2

(25 points) Recall the Affine Cipher from the lecture: for an alphabet of length m, where each letter is identified with an element of $\mathbb{Z}_m$, the key is a pair $(a, b) \in \mathbb{Z}_m$. A plaintext $x \in \mathbb{Z}_m$ is encrypted using the transform $ax + b \bmod m$, and a ciphertext $y \in \mathbb{Z}_m$ is decrypted using the transform $a^{-1}(y - b) \bmod m$. Is this encryption scheme vulnerable to the frequency analysis attack? Explain.

The Affine Cipher encrypts by multiplying the plaintext by one part of the key followed by an addition of the other part. It is defined as follows:

For an alphabet of length m, where each letter is identified with an element of $\mathbb{Z}_m = \{0, \ldots, m-1\}$, the key is $(a, b) \in \mathbb{Z}_m \times \mathbb{Z}_m$

Plaintext x is encrypted into ciphertext y by: $y \equiv a \cdot x + b \bmod m$

Ciphertext y is decrypted into plaintext x by: $x \equiv a^{-1}(y - b) \bmod m$

Although Affine Ciphers can be attacked exhaustively due to their small key spaces, they also share the same drawbacks of shift & substitution ciphers where the mapping between plaintext & ciphertext characters are fixed. As a result, they can be attacked by Letter Frequency Analysis. This can be done by exploiting patterns to guess which ciphertext letters correspond to common plaintext letters. They're effective when ciphertexts are too long. More specifically, here are some of the steps for such an attack:

1. The attacker must observe the ciphertext y and count the frequency of each letter
2. The attacker must then compare letter frequencies from the ciphertext with known frequencies of letters in the plaintext language $\mathbb{Z}_m$
   b) They could make an assumption that: Most frequent letter in ciphertext → Most frequent letter in English
   $e_K(x_1) = y_1 \equiv a \cdot x_1 + b \bmod m$
   $e_K(x_2) = y_2 \equiv a \cdot x_2 + b \bmod m$
3. The attacker can set up equations for these mappings. let $k = (a, b)$
4. The attacker can find out the values of a and b by solving the system of linear equations
5. Given ciphertext y, they can decrypt it with determined a & b by $d_K(y) = x \equiv a^{-1}(y - b) \bmod m$