Aria Javani

9725303

1.5 :

first we simplify both numbers by finding their corresponding smallest positive integer and then multiply them.

1. $15.29 \; mod \; 13 = 6$
2. $2.29 \; mod \; 13 = 6$
3. $2.3 \; mod \; 13 = 6$
4. $-11.3 \; mod \; 13 = 6$

1.8 :

$\mathbb{Z}_{11} \rightarrow 5.9 \; mod \; 11 = 1 \rightarrow (5)_{11}^{-1} = 9$

$\mathbb{Z}_{12} \rightarrow 5.5 \; mod \; 12 = 1 \rightarrow (5)_{12}^{-1} = 5$

$\mathbb{Z}_{13} \rightarrow 5.8 \; mod \; 13 = 1 \rightarrow (5)_{13}^{-1} = 8$

1.10 :

$\varphi(4) = 2 : 1,3$

$\varphi(5) = 4 : 1,2,3,4$

$\varphi(9) = 6 : 1,2,4,5,7,8$

$\varphi(26) = 12 : 1,3,5,7,9,11,15,17,19,21,23,25$

1.13 :

since plaintexts are encoded by affine cipher we have two equation for our plaintext, encodedtext pairs so we use these equations two find out key values (a,b).

$$\begin{cases} y_1 = ax_1 + b \\ y_2 = ax_2 + b \end{cases} \rightarrow \begin{cases} -y_1 + ax_1 = -b \\ y_2 - ax_2 = b \end{cases} \rightarrow y_2 - y_1 = a(x_2 - x_1) \rightarrow$$

$$a = \frac{y_2 - y_1}{x_2 - x_1} mod\ 26$$

$$b = y_1 - ax_1 mod\ 26$$

2.1 :

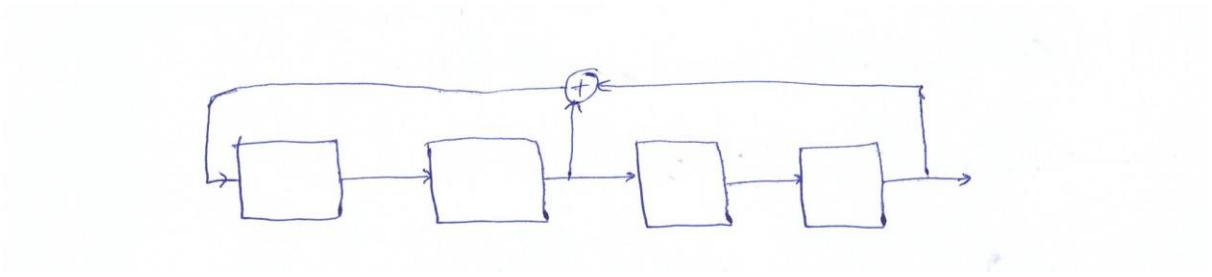
primitive LFSR : these kind of LFSRs produce the maximum possible sequence and all of the possible streams are available from every starting point.

irreducible LFSR : these polynomials cannot be factored and length of the produced sequence is independent from starting point in other words all possible rotations have the same length.

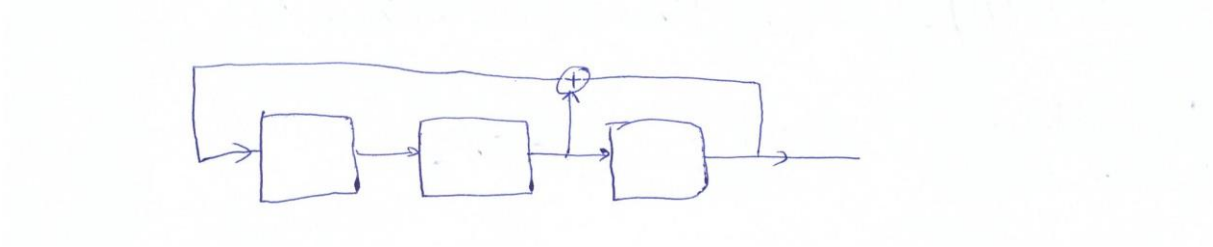
reducible LFSR : these polynomials can be factored and defferent starting points can lead to different length sequences.

2.2 :

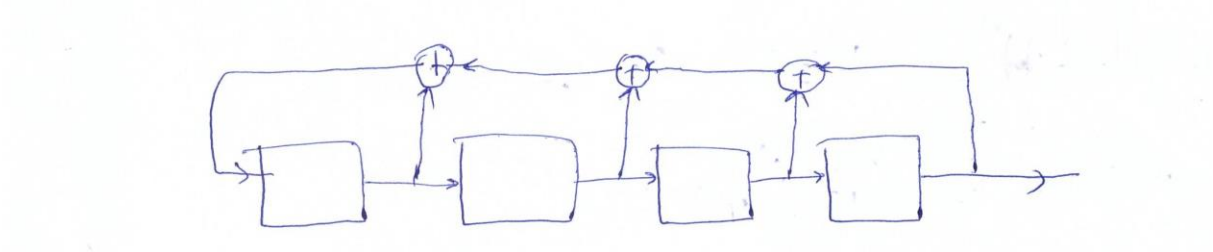
$x^4 + x^2 + 1$



$x^3 + x^2 + 1$



$x^4 + x^3 + x^2 + x + 1$



2.3 :

$x^4 + x^2 + 1$: reducible since it can have different length with different starting point.

$x^3 + x^2 + 1$: primitive since it has maximum possible length.

$x^4 + x^3 + x^2 + x + 1$: irreducible since it has same period with every starting point.

2.4 :

$x^4 + x^2 + 1$: 6, 6, 3

$x^3 + x^2 + 1$: 7

$x^4 + x^3 + x^2 + x + 1$: 5

3.1 : period length = 31

3.2 :

output in each step is the rightmost bit

starting from 11011

```
Message: 01101 at step 1
Message: 00110 at step 2
```

```
Message: 00011 at step 3
Message: 10001 at step 4
Message: 11000 at step 5
Message: 11100 at step 6
Message: 11110 at step 7
Message: 11111 at step 8
Message: 01111 at step 9
Message: 00111 at step 10
Message: 10011 at step 11
Message: 11001 at step 12
Message: 01100 at step 13
Message: 10110 at step 14
Message: 01011 at step 15
Message: 00101 at step 16
Message: 10010 at step 17
Message: 01001 at step 18
Message: 00100 at step 19
Message: 00010 at step 20
Message: 00001 at step 21
Message: 10000 at step 22
Message: 01000 at step 23
Message: 10100 at step 24
Message: 01010 at step 25
Message: 10101 at step 26
Message: 11010 at step 27
Message: 11101 at step 28
Message: 01110 at step 29
Message: 10111 at step 30
```

4.1 :

512 consecutive pairs of plaintext and encoded text bits

4.2 :

after receiving the mentioned bits we has to regenerate key, each bit is sum of its corresponding pair : $s_i = x_i + y_i$

in order to completely characterize the LFSR we need to find all of its coefficients. since it has a degree of 256 we need 256 linearly dependent equations.

to build these equation we use the

$s_{i+m} = \sum_{j=0}^{m-1} p_j . s_{i+j}$

equation for m=256 and i=0,1,…,256

then all we have to do is to find every p_j by solving these equations.

4.3 :

the key is represented by the 256 feedback coefficients. since the output of LFSR is directly XORed with the plaintext and it's easily reversible, it would be easy to find.

5.1 :

ciphered text : j5a0edj2b

sample prefix text : WPI

palin text : WPIWOMBAT

5.2 :

initialization vector : 1,1,1,1,1,1

5.3 :

[1,1,0,0,0,0]

(0,1,6)

5.4 :

The common wombat lives mainly in wet, partly forested areas on the coast, and on the ranges and western slopes.

5.5 :

Known-plaintext Attack

** the python program is attached as 5_1_LFSR_decoding.py

6.6 :

$$
\begin{aligned}
&\text{for} \quad i = 1 \text{ to } N \text{ do} \\
&\qquad t_1 \leftarrow s_{66} + s_{93} \\
&\qquad t_2 \leftarrow s_{162} + s_{177} \\
&\qquad t_3 \leftarrow s_{243} + s_{288} \\
&\qquad z_i \leftarrow t_1 + t_2 + t_3 \\
&\qquad t_1 \leftarrow t_1 + s_{91} \cdot s_{92} + s_{171} \\
&\qquad t_2 \leftarrow t_2 + s_{175} \cdot s_{176} + s_{264} \\
&\qquad t_3 \leftarrow t_3 + s_{286} \cdot s_{287} + s_{69} \\
&\qquad (s_1, s_2, \ldots, s_{93}) \leftarrow (t_3, s_1, \ldots, s_{92}) \\
&\qquad (s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (t_1, s_{94}, \ldots, s_{176}) \\
&\qquad (s_{178}, s_{179}, \ldots, s_{288}) \leftarrow (t_2, s_{178}, \ldots, s_{287})
\end{aligned}
$$

$$
\begin{aligned}
&(s_1, s_2, \ldots, s_{93}) \leftarrow (K_3, \ldots, K_{80}, 0, \ldots, 0) \\
&(s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (IV_1, \ldots, IV_{80}, 0, \ldots, 0) \\
&(s_{178}, s_{179}, \ldots, s_{288}) \leftarrow (0, \ldots, 0, 1, 1, 1)
\end{aligned}
$$

We initially run through four 288 bit shifts before we take an output (this is defined as the warm up phase).

Python file named 6_trivium.py is attached to folder.

7.1 :

Aria Javani → Mdum Vmhmzu

7.2 :

offset = 9725303 mod 26 = 3

answer : Pbqqhpp bpbmiig qcnhp dc djcph wjc msh dcc vbpg dc vh iccyoer acs od.

7.3 :

a. key=AJ

encoded text : Sdclebs dsdaulh cxmns co chxsn wqo jrn txo kuby co ke uoxkrnp fxr rt.

b. key=ARIAJAVANI

encoded text : Slkcnsn ufcacty lohef bo kpobe rhb irv box bpsl bo sm lxofiao ffz ic.

c. entropy of a : 4.20/4.70 (22 out of 26 charcters) , entropy of b : 4.11/4.70 (21 out of 26 characters)

as we can see the first key has a better result since it used more alphabet characters so in this case shorter key caused improvement of the entropy.

7.4 :

the diagram shows the number of matches for every offset.

derived key : ISFAHANUNIVOFTECHNOLOGY

decoded text : Container-based Virtualization provides a different level of

abstraction in terms of virtualization and isolation when

compared with hypervisors. In particular, it can be considered as a lightweight

alternative to hypervisor-based virtualization.

Hypervisors abstract hardware, which results in overhead in

terms of virtualizing hardware and virtual device drivers. A full

operating system (e.g., Linux) is typically run on top of this virtualized hardware

in each virtual machine instance. In

contrast, containers implement isolation of processes at the

operating system level, thus avoiding such overhead. These

containers run on top of the same shared operating system

kernel of the underlying host machine, and one or more

processes can be run within each container.

7.5 :

a. encoded(binary) file has been attached.

b. since the name isn’t long enough algorithm must use the same key over and over again.

c. encoded(binary) files has been attached. as the second key is so shorter than the plaintext the was used many times so by XORing it and comparing it, it is possible to gather some information and in combination with frequency analyzing it’s possible to decipher the text.

Optional Question :

first we have to find the key so we XOR the sample plaintext and ciphered message. the result is key with 1 nonce so in order to decrypt the second message we just have to add one to each byte of the key and then find its summation with the blake's message.

Blake's message = "DONALDTRUMP"

the decipher code is attached as Alex_Blake_RC4.py