UNIVERSITY OF WATERLOO

Winter 2019

Correlation Attack on a Simplified A5 Pseudorandom Sequence Generator

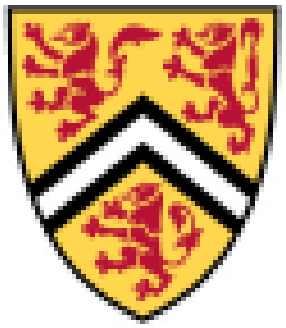
ECE 628 – COMPUTER NETWORK SECURITY

SAEM SETH & MEHROZE JALAL

ECE 628

Computer Network Security

Winter 2019



Project 1

Correlation Attack on a Simplified A5 Pseudorandom Sequence Generator

Submitted By:

Group Members: SaemSheth& Mehroze Jalal

Group Number: ece628-cp-04

Contents

Acknowledgements

We would like to pay gratitude to Prof. Guang Gong and Mr. Raghvendra Singh Rohit for their constant guidance and support during the implementation of this project.

Abstract

The intention behind this report is to outline a demonstration of a correlation attack on a simplified A5 pseudorandom sequence generator. It identifies the various methodologies used to implement a successful attack, the problems faced during its implementation, answers to the project questions provided in the problem statement and a final conclusion with recommendations of the results produced. The A5/1 is a stream cipher used in GSM in many countries around the world and has been shown to be vulnerable to certain attacks because of its relative short key size.In this project, for the attack to be launched on a simplified A5/1, certain bits of the key stream have been pre-identified, using which the key needs to be recovered. The attack has been launched in python using cross-correlation technique. The success of the attack proved A5/1 to be a weak stream cipher, because of its linear feedback shift register configuration and short key size.

Introduction

Stream ciphers are an encryption algorithm which are used to encrypt plain text messages bitwise using an encryption transformation which varies with respect to time. They are useful encryption tools for messages transferred over a wireless channel, which are generally more prone to errors. This prevents the occurrence of error propagation during decryption. Stream ciphers are mainly designed by two methods: (1) using a pseudorandom sequence generator or (2) using a block cipher model with counter mode or cipher feedback mode. Stream ciphers generate a key-stream of bits which is bitwise XORed with the plain text to create the cipher text.

The security of stream ciphers designed using PRSG is based on the randomness of the PRSG and the process undertaken to initialize the key. These stream ciphers if linear, are prone to the linear span attack and if nonlinear, they are prone to algebraic attacks. Some common stream ciphers used in communication systems include the A5/1 in GSM in cellular system, w7, which is

used in optical communication systems, E0 in Bluetooth and WEP (Wire Equivalent Privacy) for protecting wireless broadband networks [11]. This report focuses mainly on the correlation attack on the A5/1 stream cipher used in GSM.

The A5/1 stream cipher was first introduced in 1987 in Europe. It is used as a stream cipher to provide a secure over the air communication in GSM. This cipher, although initially developed to be a strong one and kept secret from the rest of the world was soon leaked and the algorithm was reverse engineered. Every 4.6 millisecond the GSM message is sent in frames of 228 bits each, and the initial key has a length of 64 bits. Below a typical A5/1 stream cipher using three LFSRs is shown.

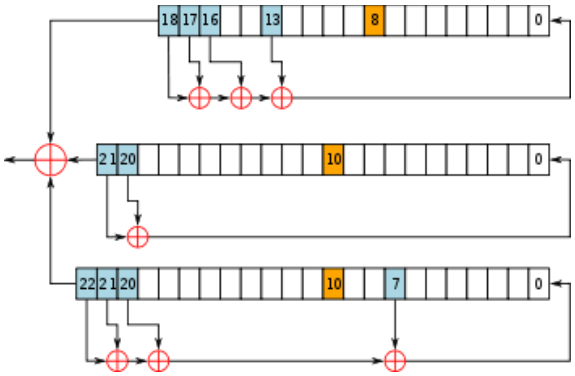


Fig. 1. The A5/1 stream cipher[4]

A number of weaknesses have been discovered in the A5/1 stream cipher as it has poor randomness properties, and hence it is prone to a number of attacks. First of all, the A5/1 stream cipher in GSM can give the same key stream for different initial states, i.e. seeds, fed into the LFSRs, secondly, without the stop-and-go operation the A5/1 has a short period and thirdly the majority function has poor correlation immunity. In this project, the latter weakness will be taken advantage of in order to attack the cipher and obtain its key. Two common attacks on the A5/1 stream cipher are the brute-force attack and the correlation attack. In this project the latter one is implemented.

Correlation attacks can break stream ciphers which make use of a number of linear feedback shift registers to generate a keystream, such as the A5/1 stream cipher in GSM. They take advantage using relatively simple Boolean functions to generate the keystream. Correlation attacks can only be implemented on ciphers if the output of the Boolean function which relates all the LFSRs is correlated to the output of each individual LFSR, which is responsible for the generation of its respective keystream; and if some bits of the keystream are known to the attacker.

This project demonstrates an implementation of a correlation attack on an A5/1 stream cipher, given some bits of the Boolean output function have been retrieved. After ensuring that the output of each LFSR is correlated to the final Boolean output, the algorithm generates each LFSR sequence for any random seeds, inputted by the attacker, and uses these sequences to correlate them with the final Boolean output, and work out the key. The key is then simply used in an XOR operation with the cipher text in order to decipher the plain text.

The project outline and objectives section of this report first defines the given problem and required tasks for this given project. The rest of the report is structured as follows. The methodology section describe the implementation and logic behind the algorithm used with supporting diagrams of the stream cipher to be attacked. Next, the results are discussed followed by answers to questions outlined in the project problem and the conclusion and recommendations that were inferred while working on this project. Finally, the appendix contains different variations of the python codes that were used throughout the project to carry out the correlation attack on the given A5/1 stream cipher.

Project Outline and Objectives

In this project a correlation attack is launched on a simplified A5/1 stream cipher, with the objective to recover the key. Certain bits of the key stream are already known and the specification of the A5/1 stream cipher is as follows. The stream cipher is designed using three LFSRs which generate m-sequences, with periods 2^7-1, 2^8-1, and 2^9-1, respectively. For LFSR$_i$, such that, i= 0, 1, 2 the respective m-sequences generated by these LFSRs are given as,**a** = {$a(t)$}, **b** = {$b(t)$}, **c** = {$c(t)$}. The primitive polynomial for LFSR$_i$ is given as $f_i(x)$where, $f_0(x) = x^7 + x + 1$; $f_1(x) = x^8 + x^4 + x^3 + x^2 + 1$; *and* $f_2(x) = x^9 + x^4 + 1$. The tap positions on the LFSRs are defined at d_0 = 3; d_1 = 4 and d_2 = 5, respectively. The majority function $f(x_0, x_1, x_2) = (y_0, y_1, y_2)$ is defined by Table 1. It can be described asy_i = 1 if x_i is in majority where $(x0; x1; x2) = (a(t + 3), b(t + 4), c(t + 5))$.

(x_0, x_1, x_2)	$f(x_0, x_1, x_2) = (y_0, y_1, y_2)$
000	111
111	
001	110
110	
011	011
100	

101	101
010	

Table 1: Majority function in A5/1

The output sequence is given by **u** = {*u(t)*} which is computed as, $u(t) = y_0(t) + y_1(t) + y_2(t) + x_0(t) + x_1(t) + x_2(t)$; t = 0,1, … where$f(a(t+3), b(t+4), c(t+5)) = (y_0(t), y_1(t), y_2(t)), (x_0(t), x_1(t), x_2(t)) = (a(t+3), b(t+4), c(t+5))$.Hence, the A5/1 behaves as a simple combination generator, as the stop-and-go operation will be removed. The problem statement states that the key initial phase is run 64 times without output and that the attacker has received the following 520 bits of the keys stream, i.e. *u(0); u(1);…; u(519) =*

00111101000011010111011010111101010011011001000000110101010100001
01011001101110111110010001100010000011110011110000010100101110101
00111000011111101010100111101000100011101110110100001011100100101
11110001110100111010100000110001001110010010101111010000110001011
00110011110101101100011111011000101101100100111001110010000110101
00101100110110010010100100010001110100110000011000000110101110100
00001111101111011111101010100001101110101110000101100010010111100
01110110110000010000011011001110100010100110101100110100001111010

The main objectives of this project include computing the Boolean representation for $y_i$for each i = 0, 1, 2, proving that the final output is correlated to each input variable, implementing the correlation attack on the cipher text and finally using the 24 bit key obtained from the attack to decipher the 64 bit cipher text given below:

$c_{1024}, c_{1025}, \ldots, c_{1087}$
=001010110000101111011010001000101001011011100110010010000011111111

Lastly, the above cipher text is required to be decrypted using an attack which is more efficient than the correlation attack.

Methodology

After having completed the necessary readings about stream ciphers, A5/1 and the correlation attack the methodology employed to implement this project begun by solving a similar example problem in the textbook, ‘Communication System Security’ by Lidong Chen and GuangGong

[11].Another example question, designated as part of a home assignment for the given course was also solved, so as to build a strong foundation before attempting the given problem.

A detailed understanding of the project problem was carried out and the LFSR configuration in the given A5/1 stream cipherwas drafted based on the given data and specifications in the problem statement(seeFigure 2).

Using Table 1 and the Karnaugh map, Boolean functions for the output of each LFSR, $y_0(t)$, $y_1(t)$ and $y_2(t)$ were computed in terms of the input variables, $x_0(t)$, $x_1(t)$ and $x_2(t)$. The procedure has been thoroughly explained in the Question/Answers section of this report, under subsection 'Part (a)'. Before having begun the correlation of the given output cipher text to find the key, it was necessary to ensure that all the inputs to the output function were correlated with the output function. This has been confirmed by computing probabilities of the output with respect to each input using table 1 and the $y_i(t)$ equations. The complete procedure has been shown in subsection 'Part (a)' of the 'Question/Answers' section of this report.

The correlation attack on A5/1 has been launched using a code written in python language. Three functions were defined for each LFSRs primitive function and tap positions; a shift function was defined to shift a sequence inputted into the function by 'n' number of bits and; three more functions to cross-correlate $u(t)$ with the output of each LFSR were defined. For any given initial sequence inputted to the three LFSR functions, three m-sequences of the LFSRs defined as, $a(t)$, $b(t)$ and $c(t)$ were generated. These m-sequences were then shifted by 64 bits to take into consideration that the key initial phase is run 64 times without output. Each of the shifted m-sequences was then correlated with the given 520 bits of the output sequence. The correlation algorithm was designed as such to avoid unnecessary, excessive computations. In this algorithm a +1 value was stored in the final sum variable from the correlation of the bits, if the bit of the m-sequence and the corresponding bit of the output sequence were same; and a -1 value was added to the sum variable if the bits were different. Using this logic a final sum of the correlation was computed. The value of τ which resulted in the maximum correlation sum with each sequence was saved. The given m-sequences are then shifted by this value of τ for the key sequence of each LFSR. Using a simple XOR operation on the cipher text, provided in the problem statement, with the key sequence results in the plain text sequence.

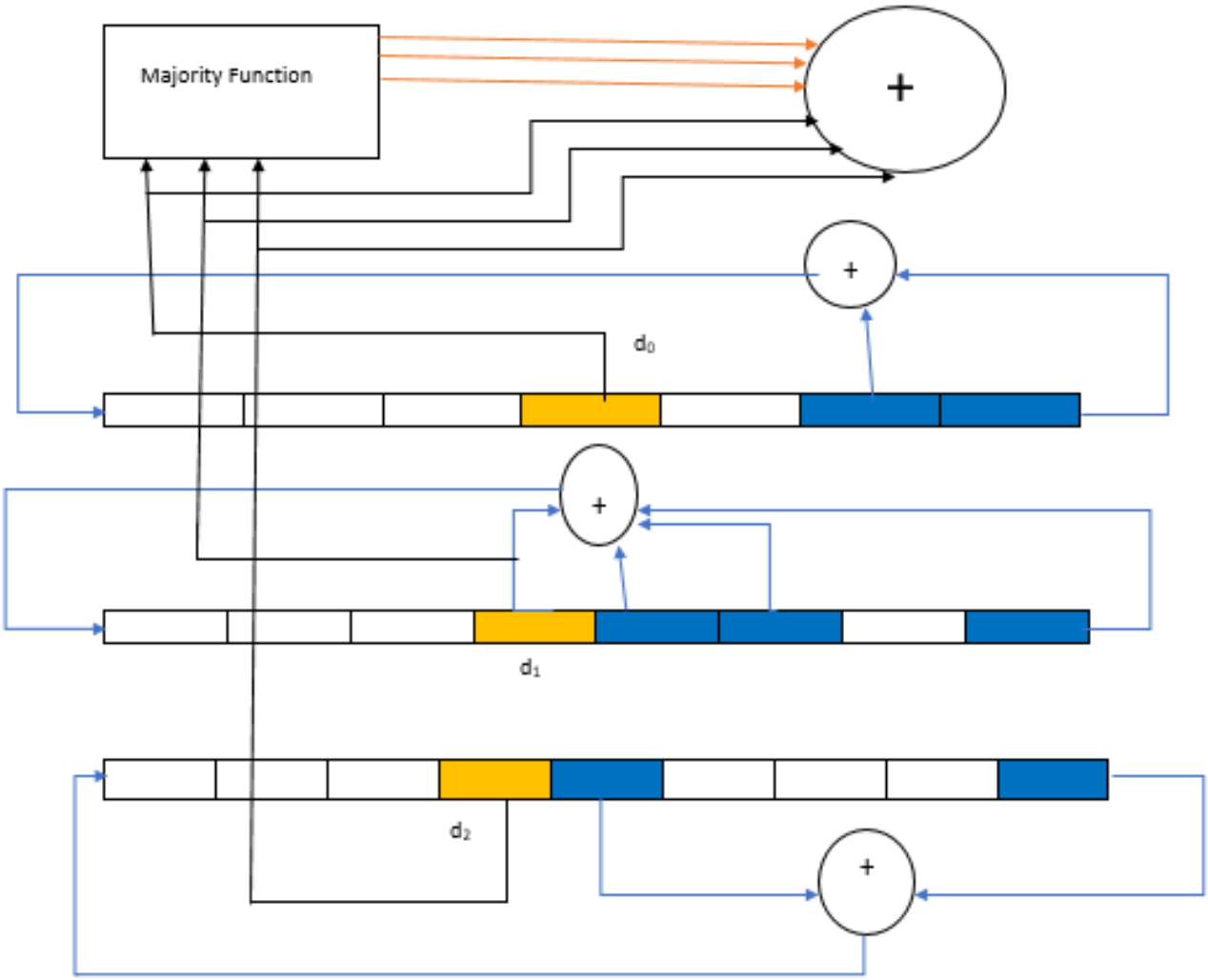


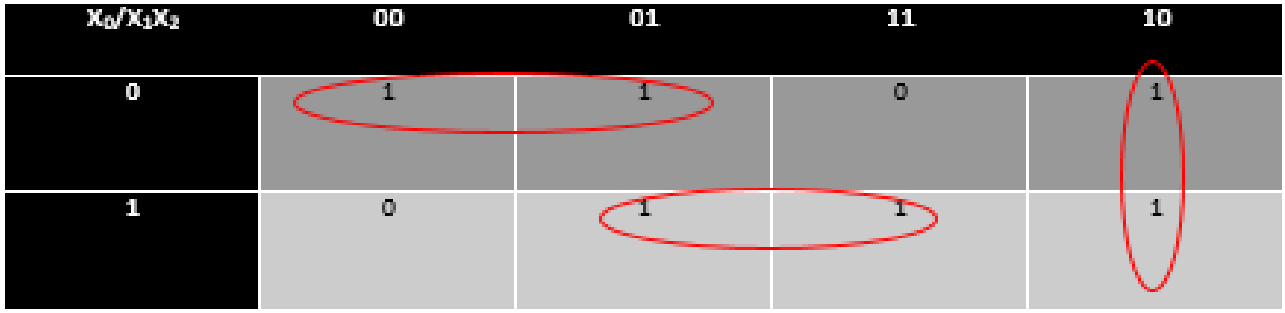
Fig. 2: A5/1 LFSR Configuration

Questions & Answers

Part (a)

Find the Boolean representation for yi for each I = 0, 1, 2, and show that the output is correlated with each input variable.

Boolean Representation of y_0

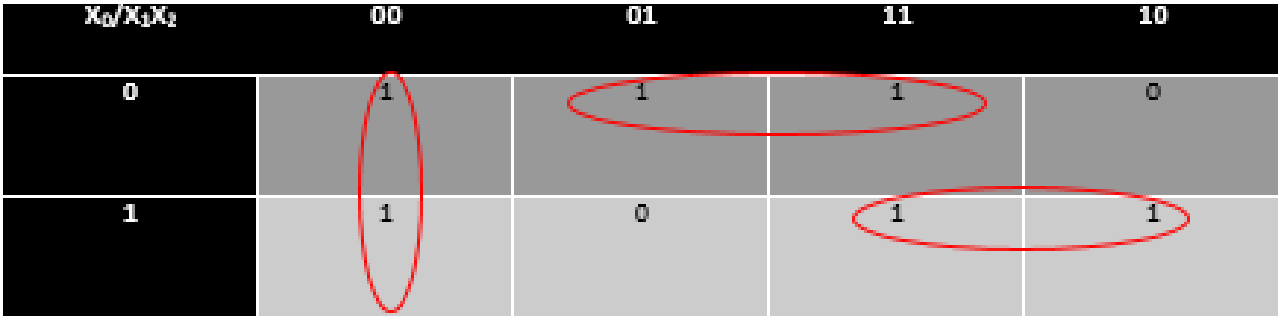


X_0/X_1X_2	00	01	11	10
0	1	1	0	1
1	0	1	1	1

Fig. 3: Karnaugh map for y_0

$y_0 = x'_0x'_1 + x_0x_2 + x_1x'_2$

Boolean Representation of y_1



X_0/X_1X_2	00	01	11	10
0	1	1	1	0
1	1	0	1	1

Fig. 4: Karnaugh map for y_1

$y_1 = x'_1x'_2 + x'_0x_2 + x_0x_1$

Boolean Representation of y_2

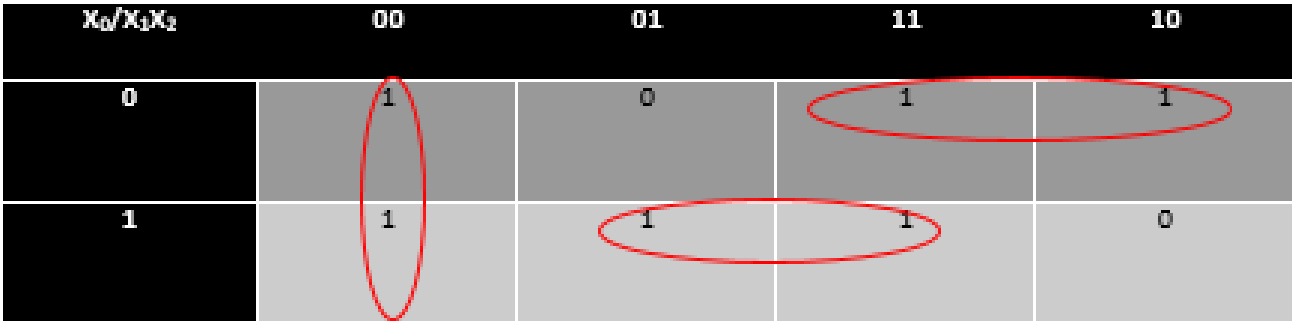


Fig. 5: Karnaugh map for y_2

$$y_2 = x'_1x'_2 + x_0x_2 + x'_0x_1$$

$$u(t) = y_0(t) + y_1(t) + y_2(t) + x_0(t) + x_1(t) + x_2(t);\ t = 0,1, \ldots$$

$x_0(t)$	$x_1(t)$	$x_2(t)$	$y_0(t)$	$y_1(t)$	$y_2(t)$	$u(t)$
0	0	0	1	1	1	1
1	0	0	0	1	1	1
0	1	0	0	0	1	0
1	1	0	1	1	0	0
0	0	1	1	1	0	1
1	0	1	1	0	1	0
0	1	1	0	1	1	0
1	1	1	1	1	1	0

$\Pr(u = 0 \mid x_0 = 0) = 1/2$

$\Pr(u = 1 \mid x_0 = 0) = 1/2$

$\Pr(u = 0 \mid x_1 = 0) = 1/4$

$\Pr(u = 1 \mid x_1 = 0) = 3/4$

$\Pr(u = 0 \mid x_2 = 0) = 1/2$

$\Pr(u = 1 \mid x_2 = 0) = 1/2$

$\Pr(u = 0 \mid x_0 = 1) = 3/4$

$\Pr(u = 1 \mid x_0 = 1) = 1/4$

$\Pr(u = 0 \mid x_1 = 1) = 1$

$\Pr(u = 1 \mid x_1 = 1) = 0$

$\Pr(u = 0 \mid x_2 = 1) = 3/4$

$\Pr(u = 1 \mid x_2 = 1) = 1/4$

The above calculations shows that the output, *u(t)* is correlated to each input, $x_i(t)$, such that, i = 0, 1,2 because the conditional probabilities are not equal. Hence, proven that each LFSR is correlated to the output.

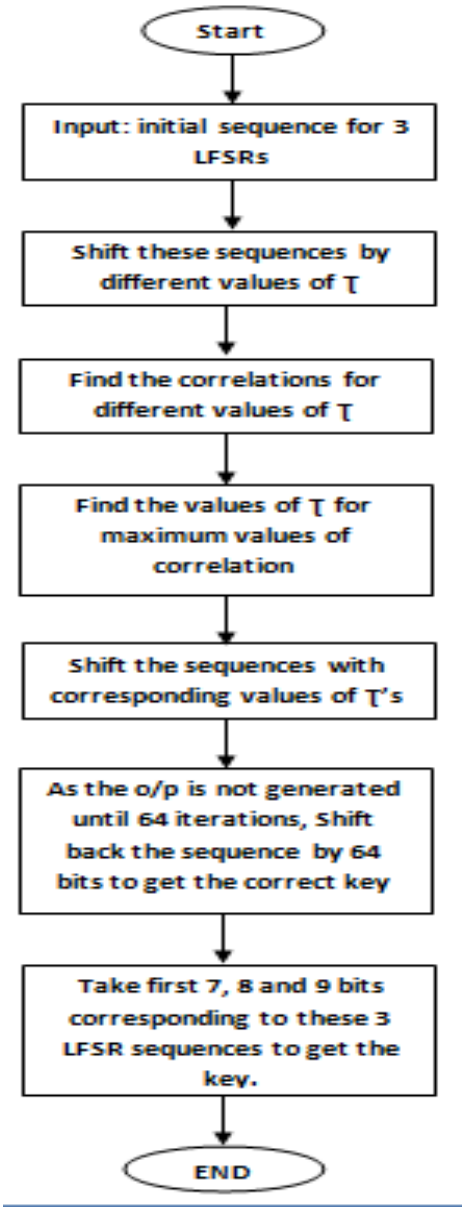
Part (b):

The Flow chart below shows the steps to find the initial state of 3 LFSRs.

Steps:

To find the key:

1. Give any random non zero initial states to 3 LFSRs
2. Shift these LFSRs with their respective polynomials and generate 520 bits.
3. Shift these sequences by different values of Ƭ.
4. Compare the shifted sequence with given bits of stream cipher output and Find the correlation values for different values of Ƭ.
5. Shift these sequences to respective Ƭ values for which the correlation value is maximum.
6. As the stream cipher does not generate the output for first 64 iterations, Shift these sequences 64 bits backwards to get the correct keys.
7. Take first 7, 8 and 9 bits respectively for LFSR 1,2 and 3 to get the key values.



The following flow chart shows the algorithm of correlation function.

Steps:

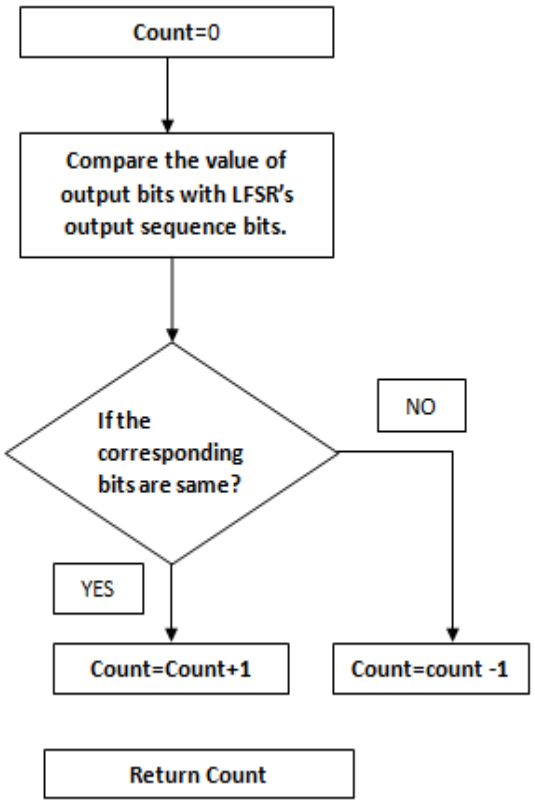
To find correlation:

1. First set the value of count=0

2. Compare the output of given stream cipher with the LFSR sequence.

3. If the corresponding bits are equal than add 1 to counter else subtract 1 from counter.(Same as $\sum (-1)^{a+b+T}$.

4. The final counter value is returned as the correlation value.



The Initial states of 3 LFSRs are:

LFSR 1:[0,1,1,1,1,1,0]

LFSR 2: [1,0,0,1,1,0,1,0]

LFSR 3:[0,1,0,1,1,1,1,0,0]

So our 24 bits key is 0111110 10011010 010111100.

Part (C):

In this part the attacker got the 64 bit cipher text and from that plain text has to be derived.

The flowchart to find the plain text from cipher text is given as below.

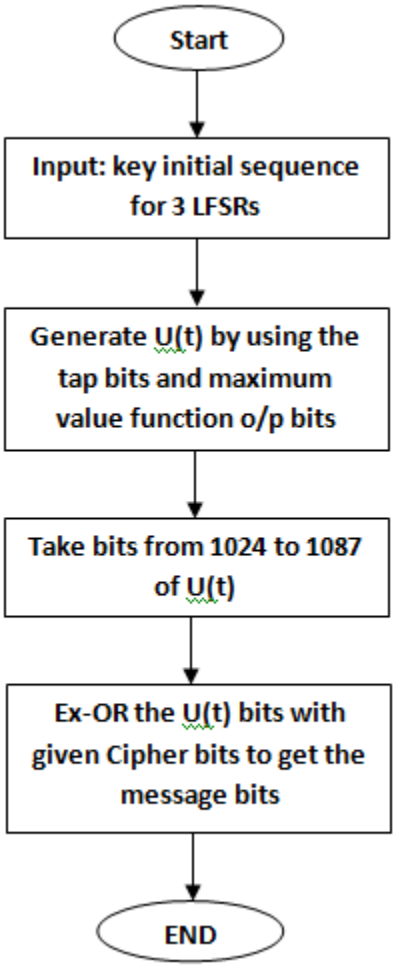
Steps

To find the plain text:

1. Give the keys that are found from part b of the question as the initial states of 3 LFSRs.

2. Generate U(t) from the equations of y_0,y_1,y_2,x_0,x_1 and x_2 that is obtained from part a.

3. Take the bits of U(t) from 1024 to 1087 and ex-or it with the given cipher text from 1024 to 1087 bits to get the plain text.



The plain text obtained from this part is:

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

Conclusion and Recommendations

During the implementation of this attack, a few weaknesses of the A5/1 were observed. One of the major ones being that, A5/1 can result in the same cipher text using more than one key. Which negates the uniqueness of the key with respect to the cipher text. Hence, if an attacker computes anyone of these keys, he or she can work out the plain text as well. Also, the majority function has poor correlation immunity and even if all the inputs are not correlated to the output function, the linearity of the LFSRs in A5/1 make it immune to still being attacked using a hit and trial method where different keys can be tried until the right one is obtained, commonly known as the brute force attack

References

[1] Project Report Format for Final Year Engineering Students. (2015, January 06). Retrieved from https://www.elprocus.com/final-project-report-format-for-electronics-engineering-students/

[2] Abstracts and Executive Summaries. (n.d.). Retrieved from https://ecp.engineering.utoronto.ca/resources/online-handbook/components-of-documents/abstracts-and-executive-summaries/

[3] Canteaut, A. (1970, January 01). A5/1. Retrieved from https://link.springer.com/referenceworkentry/10.1007/978-1-4419-5906-5_332

[4] Correlation attack. (2017, June 03). Retrieved from https://en.wikipedia.org/wiki/Correlation_attack#Explanation

[5] Douglas Wilhelm Harder, M.Math., LEL. (n.d.). Retrieved from https://ece.uwaterloo.ca/~dwharder/Reports/Styles/

[6] LibGuides: Lab Report Writing: Discussion/Conclusion. (n.d.). Retrieved from https://phoenixcollege.libguides.com/c.php?g=225738&p=1496111

[7] Project Report Format for Final Year Engineering Students. (2015, January 06). Retrieved from https://www.elprocus.com/final-project-report-format-for-electronics-engineering-students/

[8] School of Engineering and Informatics (for staff and students). (2013, March 14). Retrieved from http://www.sussex.ac.uk/ei/internal/forstudents/engineeringdesign/studyguides/techreportwriting

[9] Stream Ciphers. (n.d.). Retrieved from https://www.sciencedirect.com/topics/computer-science/stream-ciphers

[10] Villanueva, J. C. (n.d.). An Introduction To Stream Ciphers and Block Ciphers. Retrieved from https://www.jscape.com/blog/stream-cipher-vs-block-cipher

[11] Chen, L., & Gong, G. (2012). Communication System Security (Illustrated ed.). CRC Press.

Appendices

Question

Code

For correlation {part b in our question}:

```
ai=[]
bi=[]
ci=[]
m1=[]
m2=[]
m3=[]
a=[]
b=[]
c1=[]

def LFSR1(x):
    for i in range(0,520):
        x3=x&1
        x = (x>>1)|(((x&1)^((x>>1)&1))<<6)
        ai.append(x3)


def LFSR2(x):
    for i in range(0,520):
        x3=x&1
        x = (x >> 1) | (((x & 1) ^ ((x >> 2) & 1) ^ ((x >> 3) &
1) ^ ((x >> 4) & 1)) << 7)
        bi.append(x3)


def LFSR3(x):
    for i in range(0,520):
        x3=x&1
        x = (x >> 1) | (((x & 1) ^ ((x >> 4) & 1)) << 8)
        ci.append(x3)

def shift(seq, n):
    return seq[n:]+seq[:n]
```

```
def com1():
    c=0
    zi = [0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0,
1,1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1,1,0,
1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1,
0, 0, 0, 0, 1,
          0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1,
1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1,
          1,
          1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0,
1, 0, 1, 1, 1, 0, 1, 0, 1,
          0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0,
1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1,
          1,
          1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1,
1, 0, 0, 1, 0, 0, 1, 0, 1,
          1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0,
1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1,
          0,
          0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0,
1, 1, 0, 0, 0, 1, 0, 1, 1,
          0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1,
0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0,
          1,
          1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0,
0, 0, 0, 1, 1, 0, 1, 0, 1,
          0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0,
1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0,
          0,
          1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0,
1, 0, 1, 1, 1, 0, 1, 0, 0,
          0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1,
1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1,
          0,
          1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0,
0, 1, 0, 1, 1, 1, 1, 0, 0,
          0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0,
0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1,
          0,
          1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0,
0, 0, 1, 1, 1, 1, 0, 1,0]

    for i in range(0,520):
        if(zi[i]==a[i]):
            c=c+1
        else:
```

```
            c=c-1
    return c

def com2():
    c=0
    zi =
[0,0,1,1,1,1,0,1,0,0,0,0,1,1,0,1,0,1,1,1,0,1,1,0,1,0,1,1,1,1,0,1
,0,1,0,0,1,1,0,1,1,0,0,1,0,0,0,0,0,0,1,1,0,1,0,1,0,1,0,1,0,0,0,0
,1,
0,1,0,1,1,0,0,1,1,0,1,1,1,0,1,1,1,1,1,0,0,1,0,0,0,1,1,0,0,0,1,0,
0,0,0,0,1,1,1,1,0,0,1,1,1,1,0,0,0,0,0,1,0,1,0,0,1,0,1,1,1,0,1,0,
1,
0,0,1,1,1,0,0,0,0,1,1,1,1,1,1,0,1,0,1,0,1,0,0,1,1,1,1,0,1,0,0,0,
1,0,0,0,1,1,1,0,1,1,1,0,1,1,0,1,0,0,0,0,1,0,1,1,1,0,0,1,0,0,1,0,
1,
1,1,1,1,0,0,0,1,1,1,0,1,0,0,1,1,1,0,1,0,1,0,0,0,0,0,1,1,0,0,0,1,
0,0,1,1,1,0,0,1,0,0,1,0,1,0,1,1,1,1,0,1,0,0,0,0,1,1,0,0,0,1,0,1,
1,
0,0,1,1,0,0,1,1,1,1,0,1,0,1,1,0,1,1,0,0,0,1,1,1,1,1,0,1,1,0,0,0,
1,0,1,1,0,1,1,0,0,1,0,0,1,1,1,0,0,1,1,1,0,0,1,0,0,0,0,1,1,0,1,0,
1,
0,0,1,0,1,1,0,0,1,1,0,1,1,0,0,1,0,0,1,0,1,0,0,1,0,0,0,1,0,0,0,1,
1,1,0,1,0,0,1,1,0,0,0,0,0,1,1,0,0,0,0,0,0,1,1,0,1,0,1,1,1,0,1,0,
0,
0,0,0,0,1,1,1,1,1,1,0,1,1,1,1,1,0,1,1,1,1,1,1,1,0,1,0,1,0,1,0,0,0,0,1,
1,0,1,1,1,0,1,0,1,1,1,0,0,0,0,1,0,1,1,0,0,0,1,0,0,1,0,1,1,1,1,0,
0,
0,1,1,1,0,1,1,0,1,1,0,0,0,0,0,1,0,0,0,0,0,1,1,0,1,1,0,0,1,1,1,0,
1,0,0,0,1,0,1,0,0,1,1,0,1,0,1,1,0,0,1,1,0,1,0,0,0,0,1,1,1,1,0,1,
0]


    for i in range(0,520):
        if(zi[i]==b[i]):
            c=c+1
        else:
            c=c-1
    return c


def com3():
    c=0
    zi =
[0,0,1,1,1,1,0,1,0,0,0,0,1,1,0,1,0,1,1,1,0,1,1,0,1,0,1,1,1,1,0,1
,0,1,0,0,1,1,0,1,1,0,0,1,0,0,0,0,0,0,1,1,0,1,0,1,0,1,0,1,0,0,0,0
,1,
0,1,0,1,1,0,0,1,1,0,1,1,1,0,1,1,1,1,1,0,0,1,0,0,0,1,1,0,0,0,1,0,
```

```
0,0,0,0,1,1,1,1,0,0,1,1,1,1,0,0,0,0,0,1,0,1,0,0,1,0,1,1,1,0,1,0,
1,
0,0,1,1,1,0,0,0,0,1,1,1,1,1,1,0,1,0,1,0,1,0,0,1,1,1,1,0,1,0,0,0,
1,0,0,0,1,1,1,0,1,1,1,0,1,1,0,1,0,0,0,0,1,0,1,1,1,0,0,1,0,0,1,0,
1,
1,1,1,1,0,0,0,1,1,1,0,1,0,0,1,1,1,0,1,0,1,0,0,0,0,0,1,1,0,0,0,1,
0,0,1,1,1,0,0,1,0,0,1,0,1,0,1,1,1,1,0,1,0,0,0,0,1,1,0,0,0,1,0,1,
1,
0,0,1,1,0,0,1,1,1,1,0,1,0,1,1,0,1,1,0,0,0,1,1,1,1,1,0,1,1,0,0,0,
1,0,1,1,0,1,1,0,0,1,0,0,1,1,1,0,0,1,1,1,0,0,1,0,0,0,0,1,1,0,1,0,
1,
0,0,1,0,1,1,0,0,1,1,0,1,1,0,0,1,0,0,1,0,1,0,0,1,0,0,0,1,0,0,0,1,
1,1,0,1,0,0,1,1,0,0,0,0,0,1,1,0,0,0,0,0,0,1,1,0,1,0,1,1,1,0,1,0,
0,
0,0,0,0,1,1,1,1,1,1,0,1,1,1,1,0,1,1,1,1,1,1,1,0,1,0,1,0,1,0,0,0,0,1,
1,0,1,1,1,0,1,0,1,1,1,0,0,0,0,1,0,1,1,0,0,0,1,0,0,1,0,1,1,1,1,1,0,
0,
0,1,1,1,0,1,1,0,1,1,0,0,0,0,0,1,0,0,0,0,0,1,1,0,1,1,0,0,1,1,1,0,
1,0,0,0,1,0,1,0,0,1,1,0,1,0,1,1,0,0,1,1,0,1,0,0,0,0,1,1,1,1,0,1,
0]
```

```
    for i in range(0,520):
        if(zi[i]==c1[i]):
            c=c+1
        else:
            c=c-1
    return c

i1=int(input("Enter the initial state for LFSR1 in range(1,127)"
))
i2=int(input("Enter the initial state for LFSR2 in range(1,255)"
))
i3=int(input("Enter the initial state for LFSR3 in range(1,511)"
))


LFSR1(i1)
LFSR2(i2)
LFSR3(i3)

for i in range(0,520):
    T1= i % 128
    a=shift(ai,T1)
    z1 = com1()
    m1.append(z1)
v1=max(m1)
```

```
max_index1=m1.index(v1)
k1=shift(ai,max_index1)
k1=shift(k1,128-64)
k1=k1[:7]

for i in range(0,520):
    T2= i % 256
    b=shift(bi,T2)
    z2 = com2()
    m2.append(z2)
v2=max(m2)
max_index2=m2.index(v2)
k2=shift(bi,max_index2)
k2= shift(k2,256-64)
k2=k2[:8]

for i in range(0,520):
    T3= i % 512
    c1=shift(ci,T3)
    z3 = com3()
    m3.append(z3)
v3=max(m3)
max_index3=m3.index(v3)
k3=shift(ci,max_index3)
k3= shift(k3,512-64)
k3=k3[:9]

print(k1)
print(k2)
print(k3)
```

For finding the plain text {part C in our question}:

```
F = []
u=[]
ans=[]
def LFSR(x,y,z):
    for i in range(0,1088):
        t1 = (x >> 2) & 1
        x = (x>>1)|(((x&1)^((x>>1)&1))<<6)
        t2 = (y >> 3) & 1
        y = (y>>1)|(((y&1)^((y>>2)&1)^((y>>3)&1)^((y>>4)&1))<<7)
        t3 = (z >> 4) & 1
```

```
        z = (z >> 1) | (((z & 1) ^ ((z >> 4) & 1)) << 8)
        g=(((~t1) & (~t2)) | ((t2)&(~t3)) | ((t1)&(t3))) %2
        h = ((~t2) & (~t3) | ((t3) & (~t1)) | ((t1) & (t2))) %2
        l = ((~t2) & (~t3) | ((t3) & (t1)) | ((t2) & (~t1))) %2
        f = (t1 ^ t2 ^ t3 ^ g ^ h ^ l)
        F.append(f)

LFSR(62, 89, 122)
c=[0,0,1,0,1,0,1,1,0,0,0,0,1,0,1,1,1,1,0,1,1,0,1,0,0,0,1,0,0,0,1,0,1,0
,0,1,0,1,1,0,1,1,1,0,0,1,1,0,0,1,0,0,1,0,0,0,0,1,1,1,1,1,1,1]

for i in range(1024,1088):
    a=F[i]
    u.append(a)

for i in range(0,64):
    b=u[i]^c[i]
    ans.append(b)
print(ans)
```

Assignment question code: (For correlation part only)

```
xi=[]
y=[]
mi=[]
def LFSR(x):
    for i in range(0,21):
        x3 = x & 1
        x =( (x>>1) | ((x&1) ^ ((x>>1)&1)  )<<2)
        xi.append(x3)


LFSR(1)
print(xi)

def shift(seq, n):
    return xi[n:]+xi[:n]

def com():
    c=0
    zi = [1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1,
0]
    for i in range(0,21):
        if(zi[i]==y[i]):
            c=c+1
        else:
            c=c-1
```

```
    return c

for i in range(0,7):
    T= i % 7
    y=shift(xi,T)
    z=com()
    mi.append(z)
v=max(mi)
max_index=mi.index(v)
print(v)
print(max_index)
k=shift(xi,max_index)
k=k[:3]
print(k)
```

Book/Slides example code (For correlation part only):

```
ai=[]
ci=[]
m1=[]
m3=[]
a=[]
c1=[]
def LFSR1(x):
    #x = 1
    for i in range(0,40):
        x3=x&1
        x = (x>>1)|(((x&1)^((x>>1)&1))<<1)
        ai.append(x3)

def LFSR3(x):
    for i in range(0,40):
        x3=x&1
        x = (x >> 1) | (((x & 1) ^ ((x >> 3) & 1)) << 4)
        ci.append(x3)

def shift(seq, n):
    return seq[n:]+seq[:n]

def com1():
    c=0
    zi =
[0,1,0,0,0,0,1,1,1,0,1,1,1,0,1,0,0,1,0,1,1,0,1,1,0,0,1,1,1,1,0,1,1,0,0
,1,0,1,1,1]

    for i in range(0,40):
```

```
        if(zi[i]==a[i]):
            c=c+1
        else:
            c=c-1
    return c

def com3():
    c=0
    zi =
[0,1,0,0,0,0,1,1,1,0,1,1,1,0,1,0,0,1,0,1,1,0,1,1,0,0,1,1,1,1,0,1,1,0,0
,1,0,1,1,1]
    for i in range(0,40):
        if(zi[i]==c1[i]):
            c=c+1
        else:
            c=c-1
    return c

LFSR1(1)
LFSR3(11)
for i in range(0,40):
    T1= i % 4
    a=shift(ai,T1)
    z1 = com1()
    m1.append(z1)
v1=max(m1)
max_index1=m1.index(v1)
k1=shift(ai,max_index1)
k1=k1[:2]
print(k1)
for i in range(0,40):
    T2= i % 32
    c1=shift(ci,T2)
#    print(y)
    z3 = com3()
    m3.append(z3)
v3=max(m3)
max_index3=m3.index(v3)
k3=shift(ci,max_index3)
k3=k3[:5]
print(k3)
print( ai)
print(ci)
```