CS 406: Cryptography and Network Security
Attacks on Stream Ciphers

Chaitanya Garg 210050039
Omm Agrawal 210050110

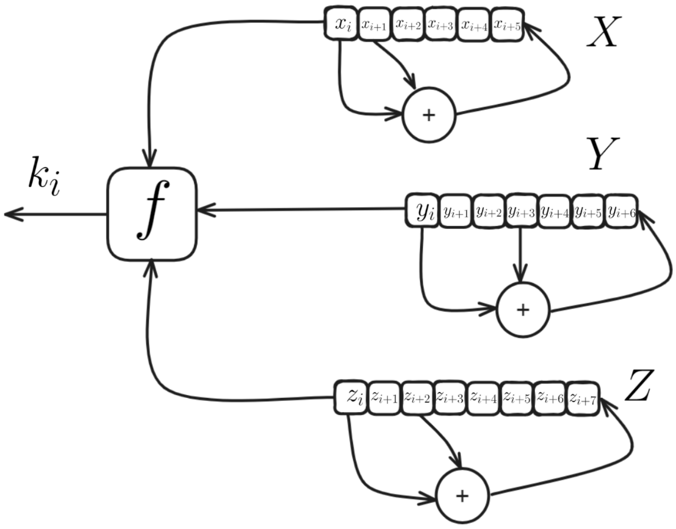
Reference

Both attacks in this project were referred from **Applied Cryptanalysis by Stamp and Low**. Additionally, several images from the book were used in this report.

1 Correlation Attack

1.1 The Cipher Description

Here, we use a keystream generator as in the diagram below consisting of three shift registers X,Y and Z.


X
x_i x_{i+1} x_{i+2} x_{i+3} x_{i+4} x_{i+5}
+
Y
k_i
f
y_i y_{i+1} y_{i+2} y_{i+3} y_{i+4} y_{i+5} y_{i+6}
+
Z
z_i z_{i+1} z_{i+2} z_{i+3} z_{i+4} z_{i+5} z_{i+6} z_{i+7}
+

The feedback functions of the registers are given by :

$$x_{i+6} = x_i \oplus x_{i+1}$$

$$y_{i+7} = y_i \oplus y_{i+3}$$

$$z_{i+8} = z_i \oplus z_{i+2}$$

The function f generates the keystream bits by the following description:

$$k_i = f(x_i, y_i, z_i) = x_i y_i \oplus y_i z_i \oplus z_i$$

1.2 Key of the cipher

The initial fills of the register form the key for the stream cipher.
Note that the feedback functions used here are same as that it the book. Only difference being that in the book, Registers X, Y and Z had only 3, 4 and 5 values respectively in the initial fill. So they had 2^{12} possible values for the key. We have a total 2^{21} values for the key. The Attack remains unaffected by this change.

1.3 Attack details

The adversary, by Kerckhoff's principle, is given the knowledge of the feedback functions and the non linear function f.

Next, it has been given a encoded message (using xor) and with the help of original message, he can get the key streams.

Now the adversary wants to predict the key of the cipher.

1.4 Brute force attack

The most trivial attack would be to find the keystream with the help of the known functions for each value of key and the value of key for which the keystream matches will be decoded key.

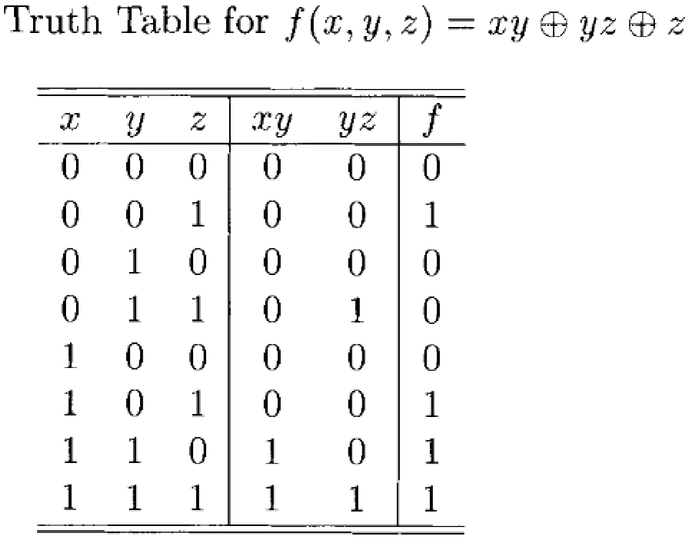
On average, the algorithm has to go through half of all possible keys. Hence the average work required is of the order.

$$w_{\mathrm{bf}} = 2^{\sum_{i=1}^{n} N_i - 1}$$

N_i is the number of initial register values in the ith register.
When key size becomes large, this method becomes infeasible.
Can we do better?

1.5 Better Attack

Let's analyze what the function output is for different values of inputs x, y and z



Truth Table for $f(x, y, z) = xy \oplus yz \oplus z$

x	y	z	xy	yz	f
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	0	0
0	1	1	0	1	0
1	0	0	0	0	0
1	0	1	0	0	1
1	1	0	1	0	1
1	1	1	1	1	1

Important observation : We notice that the f(x,y,z) = x with probability $\frac{3}{4}$ Similarly, f(x,y,z) = z with probability $\frac{3}{4}$. Also note that this probability of $\frac{3}{4}$ holds for each value of x (0 or 1) independently. Similarly for z.

Now, consider the following mechanism by attacker to reveal the initial bits of the register X:
Idea behind this attack : For the correct chosen combination, we know from the boolean table that the probability of k_i matching x_i is $\frac{3}{4}$ assuming that for sufficiently large n, y_i and z_i are sufficiently random.

But if a wrong combination is chosen, then the keystream is no longer correlate with the X generated via feedback function. Thus probability of k_i matching x_i should be $\frac{1}{2}$.
Next, since z also has the same probability effect, we use the same algorithm as described above to predict the initial bits in the Z register.

Finally iterating over initial combinations of Y registers to generate keystream which exactly matches the given keystream.

Algorithm 1 Algorithm to reveal bits of X

```
for All initial bit combinations of X do
    $x_1, x_2, x_3, x_4, x_5, x_6$ are the 6 initial bits
    $n$ = length of ciphertext/keystream
    for $i = 7, 8...n$ do
        $x_i = x_{i-6} \oplus x_{i-5}$ i.e. the feedback equation
    end for
    $x = x_1 x_2 ... x_n$
    $p$ = plain text and $c$ = cipher text
    $k = k_1 k_2 ... k_n = p \oplus c$ is the keystream
    Let t be the number of indices $i$ such that $k_i = x_i$
end for
Select the input bit combination for which t is maximum
```

1.6 Why does it work

Intuitively, as we said earlier, the probability for a character matching in the keystream and the register X (or Z) stream would be $\frac{3}{4}$ for correct initial guess and $\frac{1}{2}$ for incorrect guess. So, if the length of the keystream is n. Expected number of matches in correct guess would be $0.75 \cdot n$ and for incorrect guess it would be $0.5 \cdot n$. So our algorithm should give the correct fills with a sizable advantage.

But now let us be more mathematical and analyse our algorithm in terms of probability of success/failure. Let us analyse the correct fill for register X against only 1 incorrect fill. Our algorithm will give us a correct result iff the number of character matches for correct fill are more than that of incorrect fill.
If correct fill gives zero matches, any number of matches in the incorrect fill will yield bad result (if that is also zero, we can not be sure which one will be output, so we consider that as failure).
For some k matches in the correct fill, k or more matches in the incorrect fill will give us bad result.
By summing up these probabilities, we can get the probability of failure against a single incorrect fill as:

$$P^1[failure] = \left(\frac{1}{4}\right)^n \left({}^nC_0\left(\frac{1}{2}\right)^n + {}^nC_1\left(\frac{1}{2}\right)^n + ... + {}^nC_n\left(\frac{1}{2}\right)^n\right)$$

$$+{}^nC_1\left(\frac{1}{4}\right)^{n-1}\left(\frac{3}{4}\right)\left({}^nC_1\left(\frac{1}{2}\right)^n + ... + {}^nC_n\left(\frac{1}{2}\right)^n\right)$$

$$+{}^nC_2\left(\frac{1}{4}\right)^{n-2}\left(\frac{3}{4}\right)^2\left({}^nC_2\left(\frac{1}{2}\right)^n + ... + {}^nC_n\left(\frac{1}{2}\right)^n\right)$$

$$...$$

$$+{}^nC_n\left(\frac{3}{4}\right)^n\left({}^nC_n\left(\frac{1}{2}\right)^n\right)$$

$$\implies P^1[failure] = \sum_{i=0}^{n} {}^nC_i\left(\frac{3^i}{4^n}\right)\left(\frac{1}{2^n}\right)\left(\sum_{j=i}^{n} {}^nC_j\right)$$

If initial fill of the register has k bits, then we have to take into account failure of correct fill against each incorrect fill. So the total probability of success becomes.

$$P[success] = (1 - P^1[failure])^{2^k - 1}$$

It is not easy to calculate the exact expression here, but for a given k (security parameter), increasing n (length of message) polymonially, the Probability of success does tend to 1.

1.7 Why is it better

The amount of work (i.e number of iterations that were done) for the correlation attack is given by :

$$w_{ca} = \sum_{i=1}^{n} 2^{(N_i - 1)}$$

We can clearly see that for large values of $\sum N_i$,

$$w_{ca} << w_{bf}$$

2 RC4 Attack

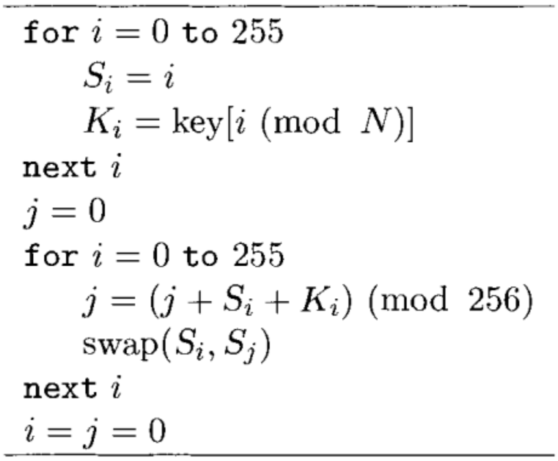
RC4 algorithm was secure (in the past, when security requirements were low) if implemented correctly. But in Wired Equivalence Privacy (WEP) protocol. It was implemented incorrectly and the following attack is on the incorrect implementation of RC4 in WEP.

2.1 RC4 cipher

The RC4 keystream is generated one byte at once. The cipher uses a lookup table S containing a permutation of all bytes from 0 to 255 along with 2 indices. This indices are initialized to 0. But the initial permutation is decided by a key which can be of any length till 256 bytes.

The RC4 initialization is given by :

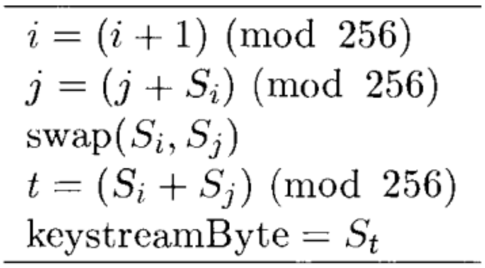
Table 3.9: RC4 Initialization



```
for $i = 0$ to 255
    $S_i = i$
    $K_i = \text{key}[i \pmod{N}]$
next $i$
$j = 0$
for $i = 0$ to 255
    $j = (j + S_i + K_i) \pmod{256}$
    swap$(S_i, S_j)$
next $i$
$i = j = 0$
```

The RC4 key generation algorithm is given by :

Table 3.10: RC4 Keystream Generator



```
$i = (i + 1) \pmod{256}$
$j = (j + S_i) \pmod{256}$
swap$(S_i, S_j)$
$t = (S_i + S_j) \pmod{256}$
keystreamByte $= S_t$
```

2.2 RC4 cipher key in practical applications

If multiple messages are encrypted using exact same key, then the messages are sent in depth which can be a serious threat to stream ciphers. Hence we generally use IV(initialization vector) with stream ciphers.
In RC4 let IV be a 3 byte, which is prepended to private key and thus generate the actual key to be used each time a message is sent.

2.3 Assumptions about Adversary for attack

We assume that he has access to a sufficient number of messages and their encryptions. Also not to mention, he knows the RC4 encryption scheme. IV is known to the adversary but not the private key which is used to construct the final key.

2.4 Fluhrer-Mantin-Shamir(FMS) Attack

Let the IV be of the form

$$IV = (3, 255, V)$$

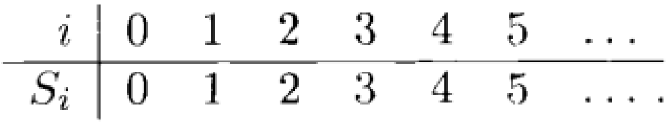
where V is known to the adversary but the actual value doesn't hamper the attack model as such.
Hence the actual key is of the form

$$K = (3, 255, V, K_3, K_4, ..)$$

where the private key bytes start from K_3
Lets the analyze a bit more carefully each step of the RC4 algorithm for this case

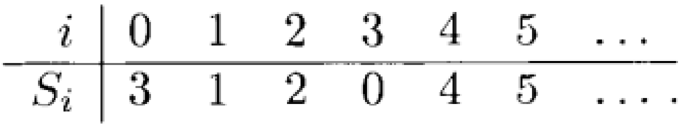
2.4.1 Analyzing RC4 initialization

First the S is the identity permutation :



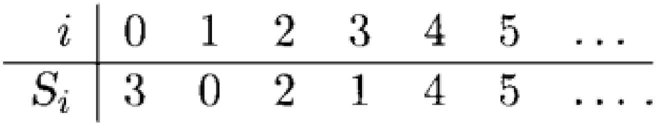
i	0	1	2	3	4	5	...
S_i	0	1	2	3	4	5	....

Now lets go through the first couple of shuffling. At the first shuffle, $i = 0$. Hence $j = 0 + S_0 + K_0 = 3$. Swapping S_0 and S_3 gives :



i	0	1	2	3	4	5	...
S_i	3	1	2	0	4	5	....

Next for $i = 1$, $j = j + S_1 + K_1 = 3 + 1 + 255 = 3$. Swapping S_1 and S_3 gives:



i	0	1	2	3	4	5	...
S_i	3	0	2	1	4	5	....

Next for $i = 2$, $j = j + S_2 + K_2 = 3 + 2 + V = 5 + V$. Swapping S_{5+V} and S_2 gives (assuming 5 + V doesn't overlap with some previous values, this holds with very good probability):
Next for $i = 3$, $j = j + S_3 + K_3 = 5 + V + 1 + K_3 = 6 + V + K - 3$. Swapping S_{6+V+K_3} and S_3 gives (assuming $6 + V + K_3$ doesn't overlap with some previous values, this holds with very good probability):
The relative positioning of $6 + V + K_3$ and $5 + V$ is irrelevant to the attack.

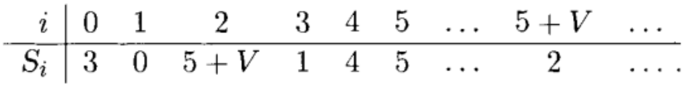
2.4.2 Attack with assumption

Lets now assume that the shuffling stops after these. Let G be the key stream and so G_1 is the first byte. Then according to the key generation algorithm :
$i = 0 + 1 = 1$, and $j = 0 + S_i = S_1 = 0$. Thus $t = S_1 + S_0 = 3$. Thus

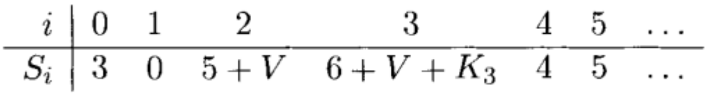
$$G_1 = S_3 = (6 + V + K_3) \mod 256$$

Now G can be obtained by the plaintext and cipher text. Thus

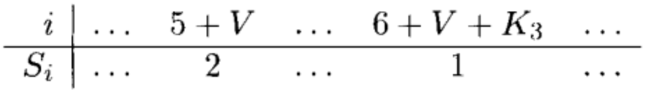
$$K_3 = G_1 - V - 6 \mod 256$$



i	0	1	2	3	4	5	...	$5+V$	...
S_i	3	0	$5+V$	1	4	5	...	2	....



i	0	1	2	3	4	5	...
S_i	3	0	$5+V$	$6+V+K_3$	4	5	...



i	...	$5+V$	...	$6+V+K_3$	...
S_i	...	2	...	1	...

2.4.3 But assumption doesn't hold - (but does it impact?)

We saw that we can predict the key if the initialization stops after the first 4 steps. But in the algorithm it doesnt. But note that as long as S_0, S_1 and S_3 are not altered in the next shuffles, the attack still works. Whats the chance of that happening?

We see that i increases normally till 255 so that wont affect. Assuming j is quite random, the chance of that not falling into the set {0,1,3} for any particular shuffle is $\frac{253}{256}$. Thus chance for j not falling into the set for any subsequent shuffle is

$$\left(\frac{253}{256}\right)^{252} = 0.05 \quad \text{approximately}$$

This implies the above result holds with a chance of 5%.

Lets say that the adversary has n messages where IV is of the given form.
Using above explanation, we expect to solve k_3 for 0.05n cases. Assuming that for remaining 0.95n cases we will get random value while solving RHS, the expected number of times we expect to see something else (not K_3) after simplifying RHS is 0.95n/256.
Hence if we choose a value like $n = 100$, with very high probability we will see that the most frequent output of the RHS is K_3.
This completes the explantion for the first key byte. Similar analysis can be done for other special IVs to get the next bytes with high probabilities.

In general, we find that by using $IV = (\lambda, 255, V)$, the following holds with significant probabilty to create a possible attack

$$K_\lambda = G_1 - \alpha - V - \beta \mod(256)$$

where

$$\alpha = \sum_{i=1}^{\lambda} i$$

$$\beta = \sum_{i=3}^{\lambda-1} K_i$$

3 Evolution of Stream Ciphers

3.1 Legacy of Stream Ciphers

Stream ciphers such as RC4 and LFSR have been central to the history of cryptography due to their simplicity and efficiency. However, over the years, various vulnerabilities and weaknesses have come to light, making them obsolete for modern cryptographic applications where stronger security guarantees are required.

3.2 RC4 and its Evolution

RC4, designed by Ron Rivest in 1987, was once widely used in protocols like WEP in wireless networks. Its popularity was due to it's quick implementation and straightforward design. Unfortunately, significant weaknesses were discovered in RC4 over time, such as biases in its keystream generation and susceptibility to attacks like the Fluhrer-Mantin-Shamir (FMS) attack, which we explored in this project.

Despite these vulnerabilities, RC4 saw some improvements over time. Variants like RC4A and RC4+ were proposed to mitigate certain weaknesses, but they did not adequately address all the security concerns. As a result, RC4 is now considered insecure and should not be used for cryptographic purposes.

3.3 Modern Stream Ciphers

In contrast to older stream ciphers, modern stream ciphers provide stronger security guarantees and are designed to resist a wide range of cryptanalytic attacks. Two prominent modern stream ciphers are:

1. **ChaCha20**: Created by Daniel J. Bernstein, ChaCha20 is an innovative ARX-based stream cipher and hash function. Its strength lies in its clever implementation of the quarter-round function, which dynamically and robustly mixes data, enhancing ChaCha20's resistance to various attacks. This versatility and security have made it widely adopted, especially in TLS protocols, where it offers high performance without compromising security, making it a preferred choice for securing sensitive data transmissions in modern cryptographic environments.

2. **AES in Counter Mode (AES-CTR)**: While AES itself is a block cipher, when used in counter mode, it operates as a stream cipher. AES-CTR combines the strong encryption properties of AES with the flexibility of a stream cipher, making it suitable for many cryptographic applications.

Both ChaCha20 and AES-CTR represent a departure from the design principles of RC4 and LFSR. They incorporate advanced cryptographic techniques and algorithms to provide robust security against modern attacks. While RC4 and LFSR were based on relatively simple designs, ChaCha20 and AES-CTR leverage complex mathematical structures and rigorous security analyses to ensure their resilience.

3.4 Conclusion

In conclusion, while RC4 and LFSR were pioneering stream ciphers, their security vulnerabilities have rendered them obsolete for contemporary cryptographic applications. Modern stream ciphers like ChaCha20 and AES-CTR offer superior security properties and are widely adopted in secure communication protocols and systems.