CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION TO RC4

Security issues are one of the most important aspects of an information system. Data or information will not be useful again if unauthorized persons have stolen the data or information. The security level should be further enhanced. The file is an important data which contains information to be exchanged. The file must have a good security protection system to be in the delivery of the archive is not leaking. Given the widespread use of information technology in all aspects such as education, government, industry, and others, then the security of the data needs to be considered properly. The system used to secure the data is cryptography. Many cryptographic techniques can be applied to the information will be protected; one of which is RC4.

Cryptographic algorithms continue to evolve as the discovery of weaknesses in each of these methods. The cryptographic algorithm consists of modern and classic. In modern algorithms, the key used was twofold symmetrical and asymmetrical. The symmetric key is the key used for encryption and decryption using the same key while the asymmetric uses two different keys in the encryption and decryption process. In this method may be applied method of stream ciphers and block ciphers.

The RC4 algorithm uses the symmetric key-shaped stream cipher. This algorithm is excellent and quick to use on a long plaintext. If the formation of S-Box has been completed, the value of the S-Box can be directly substituted with the data in plaintext. The result of the substitution of a cipher text.

The scheme consists of two parts: a key scheduling algorithm (KSA), and a pseudo-random generator (PRG). To encrypt a message, you run the key through the key scheduler, which produces a scrambled array called the state vector. You then feed the state vector into the PRG, which continuously permutes it while outputting a series of bytes. You then XOR those ‘keystream’ bytes with your plaintext.

1.2 RC4 BASED PROTOCOLS

- WEP (**Wired Equivalent Privacy**)
- WPA (**Wi-Fi Protected Access**)
- Bit Torrent protocol Encryption
- Microsoft office XP
- Microsoft Point-to-Point Encryption
- Transport Layer Security/Secure Sockets Layer
- Secure shell
- Remote Desktop Protocol
- Kerberos
- SASL (**Simple Authentication and Security Layer**)

CHAPTER 2

HISTORY

RC4 was designed by Ron Rivest of RSA security in 1987. While it is officially termed “Rivest Cipher 4”, the RC acronym is alternatively understood to stand for “Ron’s Code”. RC4 is a stream cipher type. RC4 was initially a trade secret, but in September 1994 a description of it was anonymously posted to the Cypherpunks mailing list. It was soon posted on the sci.cryptnewsgroup, where it was broken within days by Bob Jenkins. From there it spread to many sites on the Internet. The leaked code was confirmed to be genuine as its output was found to match that of proprietary software using licensed RC4. Because the algorithm is known, it is no longer a trade secret.

The name RC4 is trademarked, so RC4 is often referred to as ARCFOUR or ARC4 (meaning alleged RC4) to avoid trademark problems. RSA Security has never officially released the algorithm; Rivest has, however, linked to the English Wikipedia article on RC4 in his own course notes in 2008 and confirmed the history of RC4 and its code in a 2014 paper by him.

RC4 became part of some commonly used encryption protocols and standards, such as WEP (**Wired Equivalent Privacy**) in 1997 and WPA (**Wi-Fi Protected Access**) in 2003/2004 for wireless cards; and SSL (**Secure Sockets Layer**) in 1995 and its successor TLS (**Transport Layer Security**) in 1999, until it was prohibited for all versions of TLS by RFC (**Request for Comments**) 7465 in 2015, due to the RC4 attacks weakening or breaking RC4 used in SSL/TLS. The main factors in RC4's success over such a wide range of applications have been its speed and simplicity: efficient implementations in both software and hardware were very easy to develop.

CHAPTER 3

DESIGN

3.1.THEORY

RC4 is a stream cipher type. It processes unit or input data at one time. Unit or data is a byte or even sometimes bits. In this way, the encryption or decryption can be implemented on the length of the variable.

This algorithm does not have to wait a certain amount of data input before it is processed or add extra bytes to encrypt. The example is RC4 as shown in Figure 3.1. Another type is a block cipher that processes at the same time a certain amount of data (typically a 64-bit or 128-bit blocks) For example, Blowfish, DES, Gost, Idea, RC5, Safer, Square, Twofish, RC6, Loki97, etc.

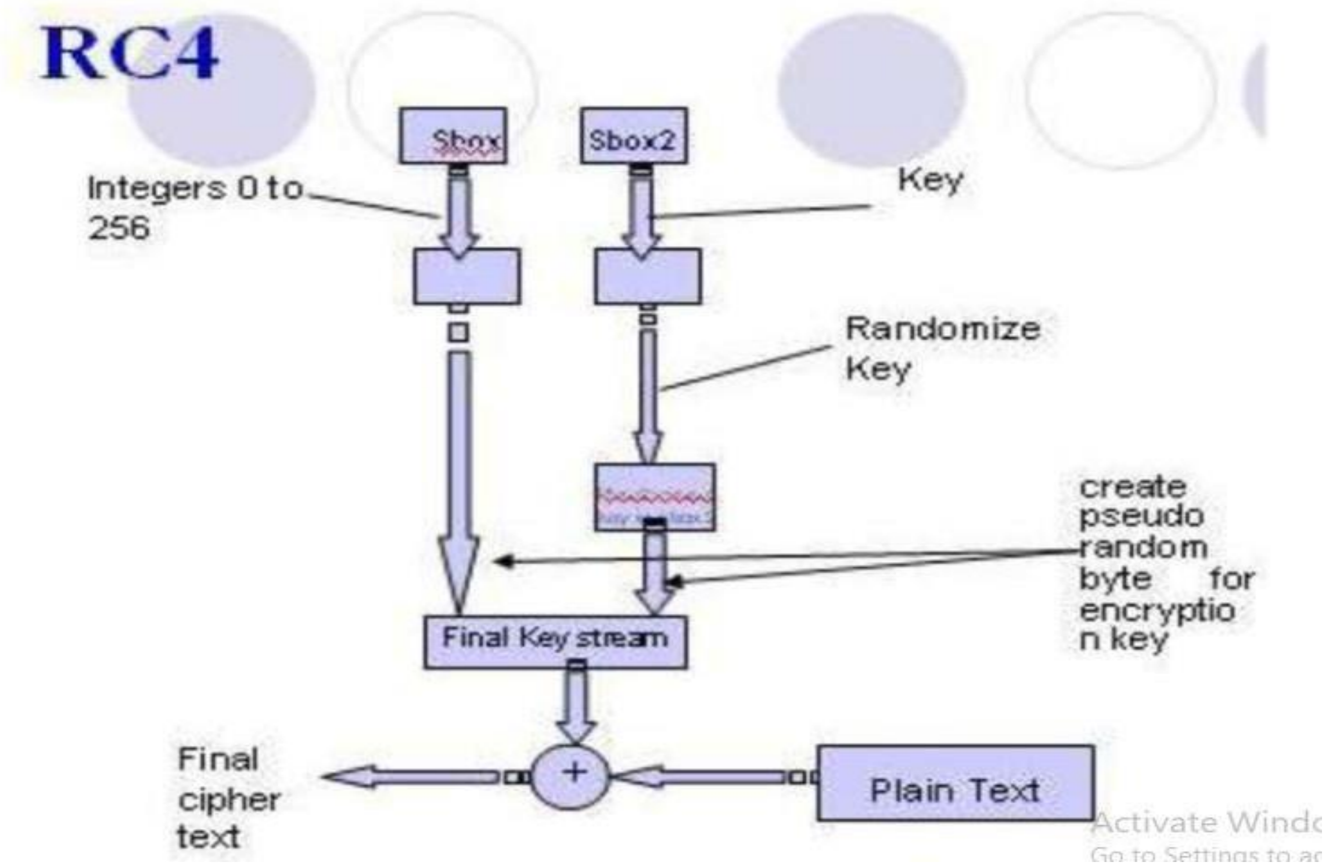


Figure 3.1: RC4 Algorithm

RC4 is a proprietary symmetric encryption stream created by RSA (**Rivest, Shamir, Adleman**) Data Security, Inc. The distribution is initiated from a source code that is believed to be as RC4 and published anonymously in 1994. The published algorithm is very synonymous with the implementation of the RC4 on the official product. RC4 is widely used in multiple applications and is commonly expressed very safe.

RC4 key generation is divided into several stages. Figure 3.2 describes the stages of the RC4. S-Box initialization is to arrange the password occupied to be the byte array. Meanwhile, the permutation is to do the new byte array to as long as the plaintext available. The new key will be encrypted to the plaintext. It generates the cipher text.

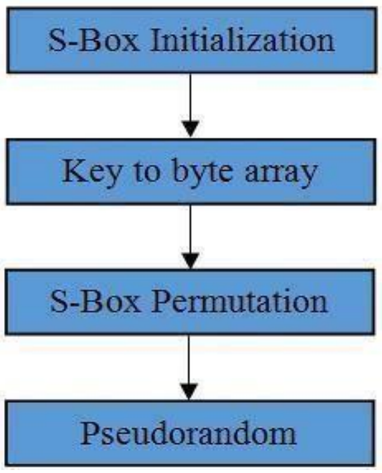


Figure 3.2: RC4 Stages

In RC4 cryptography, this algorithm has an S-Box, S [0], S [1], ... , S [255], which contains a permutation of the numbers 0 to 255 where the permutation is a function key K, with an effective length . Figure 3.3 describes the generation of the S-Box.

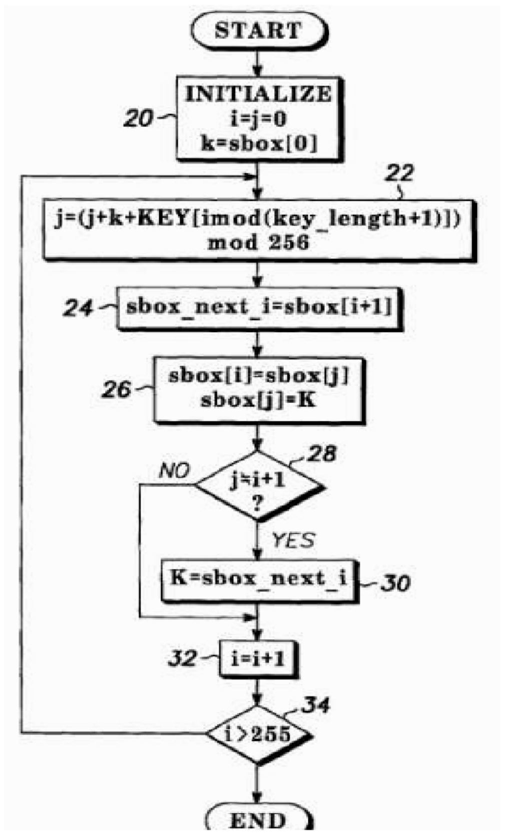


Figure 3.3: S-Box Generator

The composition of the S-Box on the same RC4 may occur. This arrangement resulted in algorithms are vulnerable. It occurs because the value of the same pseudorandom often raised repeatedly, this occurs because the user key is repeated to fill 256 bytes array. Although this method allows the use of variable length can reach 256 bytes, there no one use such length. If the user occupies eight bytes key length, it will be repeated 16 times to fill the byte array.

RC4 encryption is the XOR process between data bytes and pseudorandom byte stream generated from the key; then the attacker will be possible to determine some of the original messages by performing byte XOR process on the two sets of cipher bytes when some unknown plaintext byte. Figure 3.4 describes the Encryption Process.

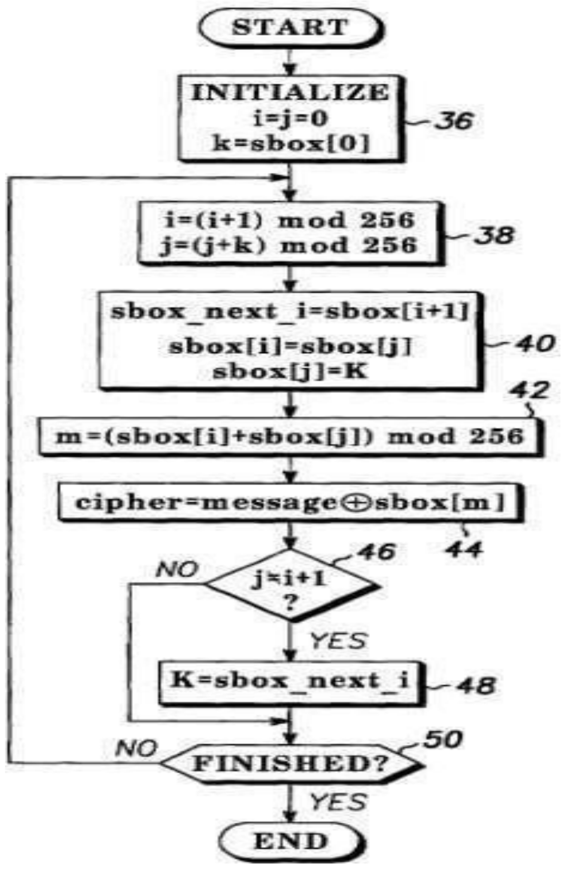


Figure 3.4: Encryption process

For example, "A" successfully intercepted two different messages encrypted using a stream cipher algorithm using the same key. "A" performs XOR to the cipher text process is successfully taken to eliminate the influence of key series. If the "A" managed to find out the plaintext of one of the encrypted messages is then "A" will easily get another message plaintext without knowing the key.

The RC4 algorithm has two phases, key generation, and encryption. Key generation is the first step and the most difficult in this algorithm. The encryption key is used to generate a variable encryption that uses two arrays, state and keys, and the results of merging operations. This merger operation consists of swapping, modulo, and other formulas. Modulo operation is the process that produces the residual value of the shares.

3.2.MATHEMATICAL MODEL

This section describes the usage of the RC4 algorithm. Assume the key is "**THIS IS THE GOOD KEY**". Table 3.1 shows the byte and index of the key provided.

Table 3.1: Table Key

1	2	3	4	5	6	7	8	9	10
84	72	73	32	32	73	83	32	84	72
T	**H**	**I**	**S**		**I**	**S**		**T**	**H**
11	12	13	14	15	16	17	18	19	20
69	32	71	79	79	68	32	75	69	89
E		**G**	**O**	**O**	**D**		**K**	**E**	**Y**

From the key generated above, the S-Box values can be determined by using the previous formula. Table 3.2 shows the S-Box value of the earlier key.

At first the S-Box contains the values from 0 to 255 in ascending order, then the S-Box is scrambled using KSA Algorithm as shown below:

```
Keylength = 20
For i=0
j = (j + S[i] + key [i mod Keylength]) mod 256
  = (0 + 0 + key [0 % 20]) mod 256
  = (84) % 256
  = 84
S[i] = S [0] = 0
S[j] = S [84] = 84
Then swap
S[i] = S [0] = 84
S[j] = S [84] = 0
```

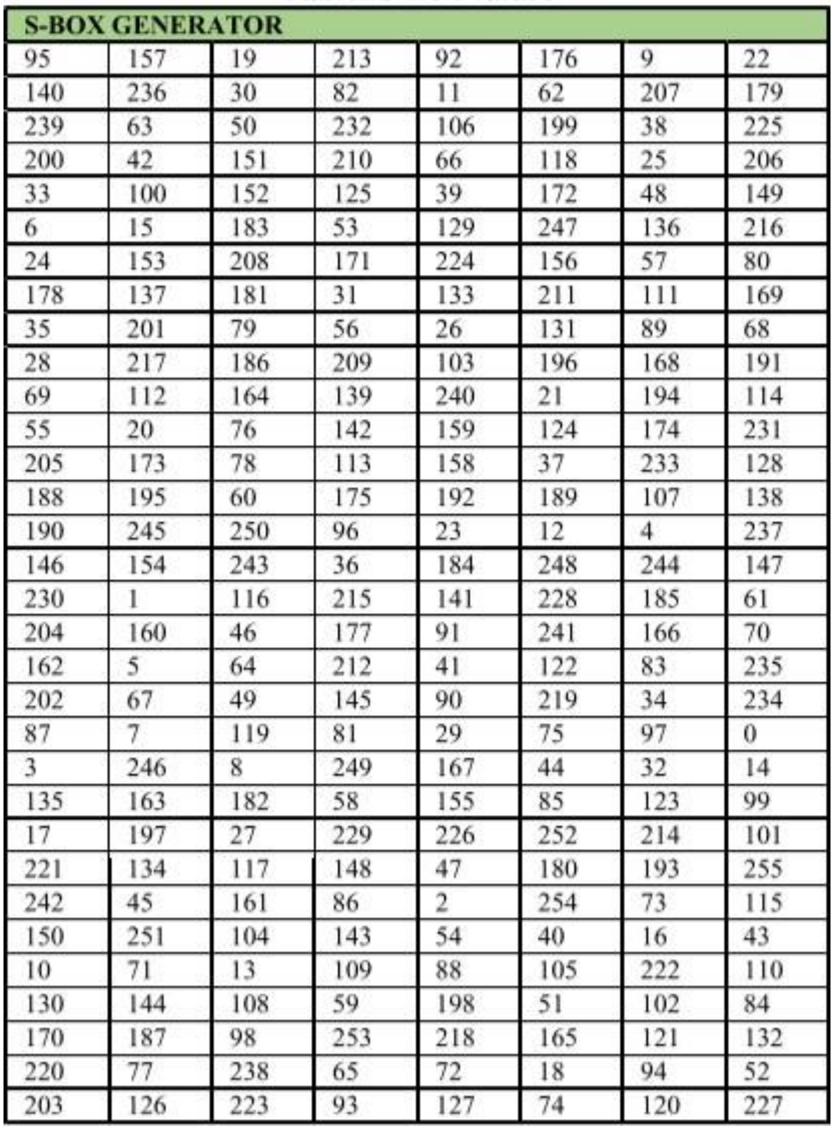
Now the i value increments and the for loop is continued (i=1)

```
For i=1
j = (j + S[i] + key [i mod Keylength]) mod 256
  = (0 + 1 + key [1 % 20]) mod 256
  = (73) % 256
  = 73
```

S[i] = S [1] = 1
S[j] = S [73] = 73
Then swap
S[i] = S [1] = 73
S[j] = S [73] = 1

This continues for 256 times and the scrambled S-Box is as shown below.

Table 3.2: S-Box Values



S-BOX GENERATOR							
95	157	19	213	92	176	9	22
140	236	30	82	11	62	207	179
239	63	50	232	106	199	38	225
200	42	151	210	66	118	25	206
33	100	152	125	39	172	48	149
6	15	183	53	129	247	136	216
24	153	208	171	224	156	57	80
178	137	181	31	133	211	111	169
35	201	79	56	26	131	89	68
28	217	186	209	103	196	168	191
69	112	164	139	240	21	194	114
55	20	76	142	159	124	174	231
205	173	78	113	158	37	233	128
188	195	60	175	192	189	107	138
190	245	250	96	23	12	4	237
146	154	243	36	184	248	244	147
230	1	116	215	141	228	185	61
204	160	46	177	91	241	166	70
162	5	64	212	41	122	83	235
202	67	49	145	90	219	34	234
87	7	119	81	29	75	97	0
3	246	8	249	167	44	32	14
135	163	182	58	155	85	123	99
17	197	27	229	226	252	214	101
221	134	117	148	47	180	193	255
242	45	161	86	2	254	73	115
150	251	104	143	54	40	16	43
10	71	13	109	88	105	222	110
130	144	108	59	198	51	102	84
170	187	98	253	218	165	121	132
220	77	238	65	72	18	94	52
203	126	223	93	127	74	120	227

The key is ready to be used for now. For example, the plaintext is **“NO ONE CAN SAVE FROM DEATH”**.

3.2.1. ENCRYPTION

For example, draw the first character of the plaintext. The char is “N”. The convert it to byte number; it results 78 in decimal format.

Set the first value of i and j to zero (i = 0, j = 0) as well. Finally, perform the calculation to generate the “K” value.

$i = (i + 1) \bmod 256$
$= 1$

$j = (j + S[i]) \bmod 256$
$= 0 + 157$
$= 157$

$S[i] = S[1] = 157$
$S[j] = S[157] = 219$
Then swap
$S[i] = S[1] = 219$
$S[j] = S[157] = 157$

$t = (S[i] + S[j]) \bmod 256$
$= (219 + 157) \bmod 256$
$= 120$

$K = S[t]$
$= S\ [120]$
$= 146$

The “K” value has been determined. It will be used to convert the plaintext to cipher text using XOR operation. The following value is the cipher text byte of the first plaintext character.

$CT = PT \oplus K$
$= 78 \oplus 146$
$= 01001110 \oplus 10010010$
$= 11011100$
$= 220$

Now i and j take the previous values (i = 1 and j = 157). Finally, perform the calculation to generate the "K" value.

i = (i + 1) mod 256
  = 2
j = (j + S[i]) mod 256
= (157 + 19) mod256
= 176

S[i] = S[2] = 19
S[j] = S[176] = 135
Then swap
S[i] = S[2] = 135
S[j] = S[176] = 19

t = (S[i] + S[j]) mod 256
= (135 + 19) mod 256
= 154

K = S[t]
= S [154]
= 49

The "K" value has been determined. It will be used to convert the plaintext to cipher text using XOR operation. The following value is the cipher text byte of the first plaintext character.

CT = PT ⊕ K
  = 79 ⊕ 49
  = 01001111 ⊕ 00110001
  = 01111110
  = 126

And the same continues for all the 26 characters and the result of the Encryption is as follows

CT = PT ⊕ K
  = 32 ⊕ 197
  = 00100000 ⊕ 11000101
  = 11100101
  = 229

$CT = PT \oplus K$
$= 79 \oplus 218$
$= 01001111 \oplus 11011010$
$= 10010101$
$= 149$

$CT = PT \oplus K$
$= 78 \oplus 85$
$= 01001110 \oplus 01010101$
$= 00011011$
$= 27$

$CT = PT \oplus K$
$= 69 \oplus 181$
$= 01000101 \oplus 10110101$
$= 11110000$
$= 240$

$CT = PT \oplus K$
$= 32 \oplus 15$
$= 00100000 \oplus 00001111$
$= 00101111$
$= 47$

$CT = PT \oplus K$
$= 67 \oplus 63$
$= 01000011 \oplus 00111111$
$= 01111100$
$= 124$

$CT = PT \oplus K$
$= 65 \oplus 238$
$= 01000001 \oplus 11101110$
$= 10101111$
$= 175$

$CT = PT \oplus K$
$= 78 \oplus 237$
$= 01001110 \oplus 11101101$
$= 10100011$
$= 163$

$CT = PT \oplus K$
$= 32 \oplus 66$
$= 00100000 \oplus 01000010$
$= 01100010$
$= 98$

CT = PT $\oplus$ K
= 83 $\oplus$ 159
= 01010011 $\oplus$ 10011111
= 11001100
= 204

CT = PT $\oplus$ K
= 65 $\oplus$ 9
= 01000001 $\oplus$ 00001001
= 01001000
= 72

CT = PT $\oplus$ K
= 86$\oplus$ 51
= 01010110$\oplus$ 00110011
= 01100101
= 101

CT = PT$\oplus$ K
= 69 $\oplus$39
= 01000101 $\oplus$ 00100111
= 01100010
= 98

CT = PT $\oplus$ K
= 32 $\oplus$ 212
= 00100000 $\oplus$ 11010100
= 11110100
= 244

CT = PT $\oplus$ K
= 70 $\oplus$ 132
= 01000110 $\oplus$ 10000100
= 11000010
= 194

CT = PT $\oplus$ K
= 82 $\oplus$ 193
= 01010010 $\oplus$ 11000001
= 10010011
= 147

CT = PT$\oplus$ K
= 79 $\oplus$ 62
= 01001111 $\oplus$ 00111110
= 01110001
= 113

CT = PT $\oplus$ K
= 77 $\oplus$ 153
= 01001101 $\oplus$ 10011001
= 11010100
= 212

CT = PT$\oplus$ K
= 32 $\oplus$74
= 00100000 $\oplus$ 01001010
= 01101010
= 106

CT = PT $\oplus$ K
= 68 $\oplus$ 245
= 01000100 $\oplus$ 11110101
= 10110001
= 177

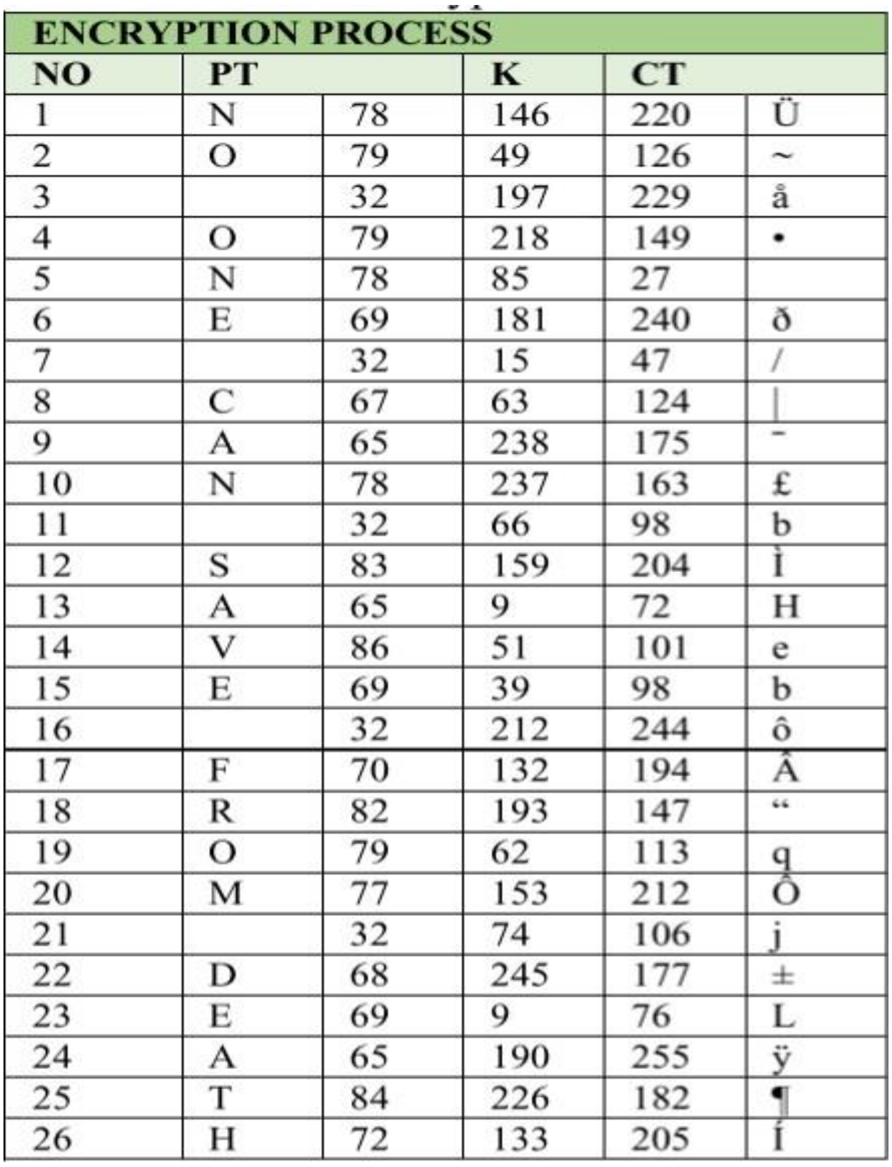
CT = PT $\oplus$ K
= 69 $\oplus$ 9
= 01000101 $\oplus$ 00001001
= 01001100
= 76

CT = PT $\oplus$ K
= 65 $\oplus$ 190
= 01000001 $\oplus$ 10111110
= 11100001
= 255

CT = PT $\oplus$ K
= 84 $\oplus$ 226
= 01010100 $\oplus$ 11100010
= 10110110
= 182

CT = PT $\oplus$ K
= 72 $\oplus$ 133
= 01001000 $\oplus$ 10000101
= 11001101
= 20

Table 3.3: Encryption Result



ENCRYPTION PROCESS					
NO	PT		K	CT	
1	N	78	146	220	Ü
2	O	79	49	126	~
3		32	197	229	å
4	O	79	218	149	•
5	N	78	85	27	
6	E	69	181	240	ð
7		32	15	47	/
8	C	67	63	124	|
9	A	65	238	175	¯
10	N	78	237	163	£
11		32	66	98	b
12	S	83	159	204	Ì
13	A	65	9	72	H
14	V	86	51	101	e
15	E	69	39	98	b
16		32	212	244	ô
17	F	70	132	194	Â
18	R	82	193	147	“
19	O	79	62	113	q
20	M	77	153	212	Ô
21		32	74	106	j
22	D	68	245	177	±
23	E	69	9	76	L
24	A	65	190	255	ÿ
25	T	84	226	182	¶
26	H	72	133	205	Í

Table 3.3 shows the complete result of the cipher text. The cipher text generated is **“Ü~å•ð/|¯£bÌHebôÂ“qÔj±Lÿ¶Í”.**

3.2.2. DECRYPTION

From the cipher text **“Ü~å•ð/|¯£bÌHebôÂ“qÔj±Lÿ¶Í”**, the first character is “Ü”. The convert it to byte number; it results 220 in decimal format.

Set the first value of i and j to zero (i = 0, j = 0). Finally, do the similar calculation to previous calculation to generate the “K” value.

i = (i + 1) mod 256
= 1
j = (j + S[i]) mod 256
= 0 + 157
= 157

S[i] = S[1] = 157
S[j] = S[157] = 219
Then swap
S[i] = S[1] = 219
S[j] = S[157] = 157

t = (S[i] + S[j]) mod 256
= (219 + 157) mod 256
= 120

K = S[t]
=S[120]
= 146

The “K” value has been determined. It will be used to convert the cipher text back to plaintext using XOR operation as well. The following value is the plaintext byte of the first cipher text character.

$PT = CT \oplus K$
$= 220 \oplus 146$
$= 11011100 \oplus 10010010$
$= 01001110$
$= 78$

Now i and j take the previous values (i = 1 and j = 157). Finally, do the similar calculation to previous calculation to generate the “K” value.

i = (i + 1) mod 256
  = 2
j = (j + S[i]) mod 256
= (157 + 19) mod256
= 176

S[i] = S[2] = 19
S[j] = S[176] = 135
Then swap
S[i] = S[2] = 135
S[j] = S[176] = 19

t = (S[i] + S[j]) mod 256
= (135 + 19) mod 256
= 154

K = S[t]
=S[154]
= 49

The “K” value has been determined. It will be used to convert the cipher text back to plaintext using XOR operation as well. The following value is the plaintext byte of the first cipher text character.

PT = CT $\oplus$ K
 = 126 $\oplus$ 49
 = 01111110 $\oplus$ 00110001
 = 01001111
 = 79

And the same continues for all the 26 characters and the result of the Decryption is as follows

PT = CT $\oplus$ K
 = 229 $\oplus$ 197
 = 11100101 $\oplus$ 11000101
 = 00100000
 = 32

PT = CT ⊕ K
= 149 ⊕ 218
= 10010101 ⊕ 11011010
= 01001111
= 79

PT = CT ⊕ K
= 27⊕ 85
= 00011011 ⊕ 01010101
= 01001110
= 78

PT = CT ⊕ K
= 240 ⊕ 181
= 11110000 ⊕ 10110101
= 01000101
= 69

PT = CT⊕K
= 47 ⊕15
= 00101111⊕ 00001111
= 00100000
= 32

PT = CT ⊕ K
= 124 ⊕ 63
= 01111100⊕ 00111111
= 01000011
= 67

PT = CT ⊕ K
= 175 ⊕ 238
= 10101111⊕ 11101110
= 01000001
= 65

PT = CT ⊕ K
= 163⊕ 237
= 10100011⊕ 11101101
= 01001110
= 78

PT = CT ⊕ K
= 98⊕ 66
= 01100010 ⊕ 01000010
= 00100000
= 32

$PT = CT \oplus K$
$= 204 \oplus 159$
$= 11001100 \oplus 10011111$
$= 01010011$
$= 83$

$PT = CT \oplus K$
$= 72 \oplus 9$
$= 01001000 \oplus 00001001$
$= 01000001$
$= 65$

$PT = CT \oplus K$
$= 101 \oplus 51$
$= 01100101 \oplus 00110011$
$= 01010110$
$= 86$

$PT = CT \oplus K$
$= 98 \oplus 39$
$= 01100010 \oplus 00100111$
$= 01000101$
$= 69$

$PT = CT \oplus K$
$= 244 \oplus 212$
$= 11110100 \oplus 11010100$
$= 00100000$
$= 32$

$PT = CT \oplus K$
$= 194 \oplus 132$
$= 11000010 \oplus 10000100$
$= 01000110$
$= 70$

$PT = CT \oplus K$
$= 147 \oplus 193$
$= 10010011 \oplus 11000001$
$= 01010010$
$= 82$

$PT = CT \oplus K$
$= 113 \oplus 62$
$= 01110001 \oplus 00111110$
$= 01001111$
$= 79$

$$\begin{aligned} PT &= CT \oplus K \\ &= 212 \oplus 153 \\ &= 11010100 \oplus 10011001 \\ &= 01001101 \\ &= 77 \end{aligned}$$

$$\begin{aligned} PT &= CT \oplus K \\ &= 106 \oplus 74 \\ &= 01101010 \oplus 01001010 \\ &= 00100000 \\ &= 32 \end{aligned}$$

$$\begin{aligned} PT &= CT \oplus K \\ &= 177 \oplus 245 \\ &= 10110001 \oplus 11110101 \\ &= 01000100 \\ &= 68 \end{aligned}$$

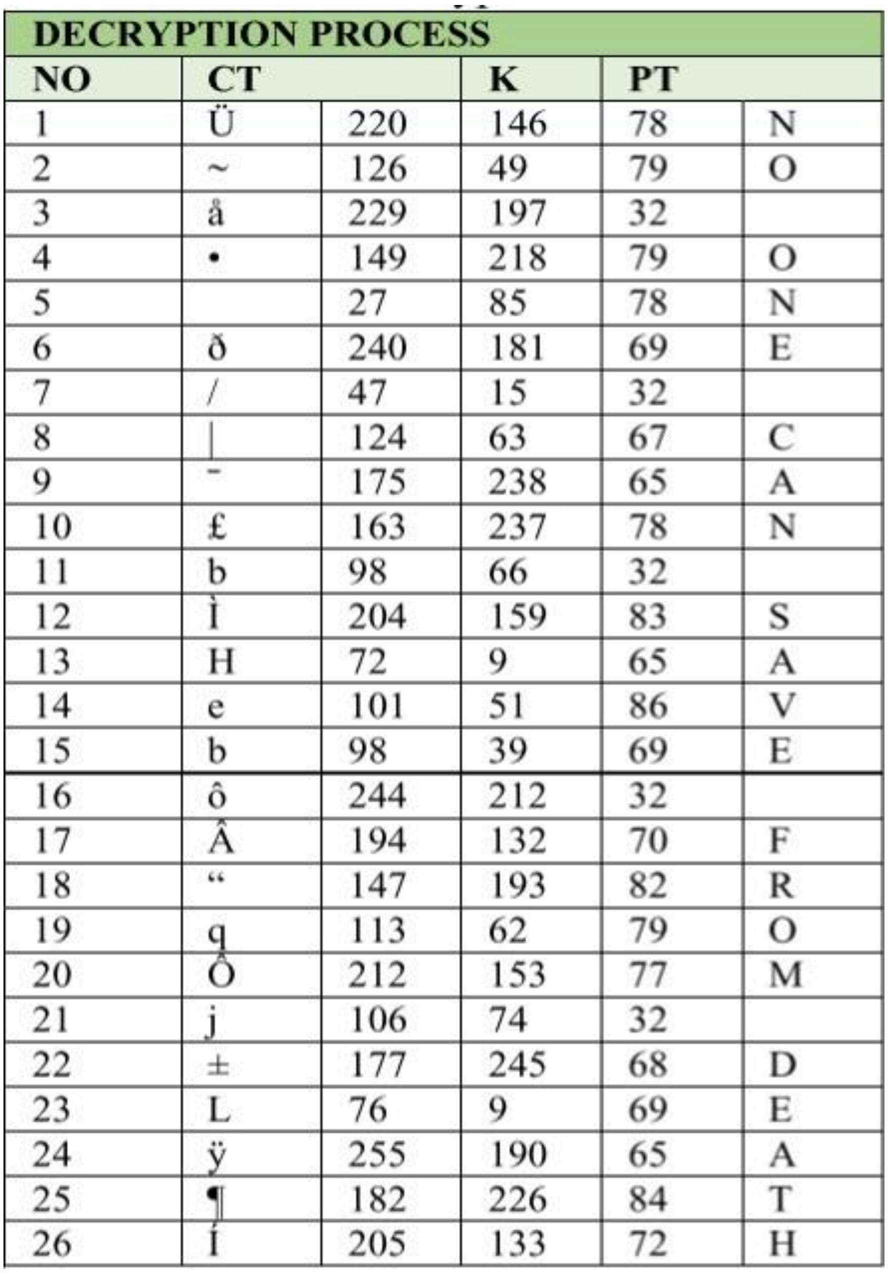
$$\begin{aligned} PT &= CT \oplus K \\ &= 76 \oplus 9 \\ &= 01001100 \oplus 00001001 \\ &= 01000101 \\ &= 69 \end{aligned}$$

$$\begin{aligned} PT &= CT \oplus K \\ &= 255 \oplus 190 \\ &= 11100001 \oplus 10111110 \\ &= 01000001 \\ &= 65 \end{aligned}$$

$$\begin{aligned} PT &= CT \oplus K \\ &= 182 \oplus 226 \\ &= 10110110 \oplus 11100010 \\ &= 01010100 \\ &= 84 \end{aligned}$$

$$\begin{aligned} PT &= CT \oplus K \\ &= 205 \oplus 133 \\ &= 11001101 \oplus 10000101 \\ &= 01001000 \\ &= 72 \end{aligned}$$

Table 3.4: Decryption Result



DECRYPTION PROCESS					
NO	CT		K	PT	
1	Ü	220	146	78	N
2	~	126	49	79	O
3	å	229	197	32	
4	•	149	218	79	O
5		27	85	78	N
6	ð	240	181	69	E
7	/	47	15	32	
8	|	124	63	67	C
9	¯	175	238	65	A
10	£	163	237	78	N
11	b	98	66	32	
12	Ì	204	159	83	S
13	H	72	9	65	A
14	e	101	51	86	V
15	b	98	39	69	E
16	ô	244	212	32	
17	Â	194	132	70	F
18	“	147	193	82	R
19	q	113	62	79	O
20	Ô	212	153	77	M
21	j	106	74	32	
22	±	177	245	68	D
23	L	76	9	69	E
24	ÿ	255	190	65	A
25	¶	182	226	84	T
26	Í	205	133	72	H

Table 3.4 shows the complete result of the plaintext. The plaintext is “**NO ONE CAN SAVE FROM DEATH**”

CHAPTER 4

IMPLEMENTATION

RC4 generate the pseudorandom key stream. Just as a stream cipher, it can be used for encryption by combining the plaintext using XOR while decryption is done in the same way as well. This process is similar to the Vernam cipher except that bit pseudorandom generated. To generate the keystream, the cipher using a secret internal state which consists of two parts:

a. A permutation of all 256-byte ASCII.
b. Two eight-bits-pointer indexes, "i" and "j".

A permutation is initialized with a variable length key, typically between 40 and 256 bits, using the key-scheduling algorithm. Furthermore, the bit stream generated using pseudo-random generation algorithm (PRGA). More specifically, RC4 operates with the following steps

4.1 PERFORM INITIALIZATION OF S

How it works S-box initialization RC4 algorithm that first, S [0], S [1], ..., S [255], with the numbers 0 to 255. First, the variable S will be filled with numbers from 0 to 255 in sequence S [0] = 0, S [1] = 1, S [255] = 255.

Then initialize another array, e.g., array K with a length of 256. The contents of the array K with a key that is repeated until the entire array K [0], K [1], ..., K [255] filled. S-Box initialization process is written as follows:

```
i = 0
j = 0
while GeneratingOutput:
    i = (i + 1) mod 256
    j = (j + S[i]) mod 256
    swap values of S[i] and S[j]
    K = S[(S[i] + S[j]) mod 256]
    output K
endwhile
```

4.2 KEYSTREAM

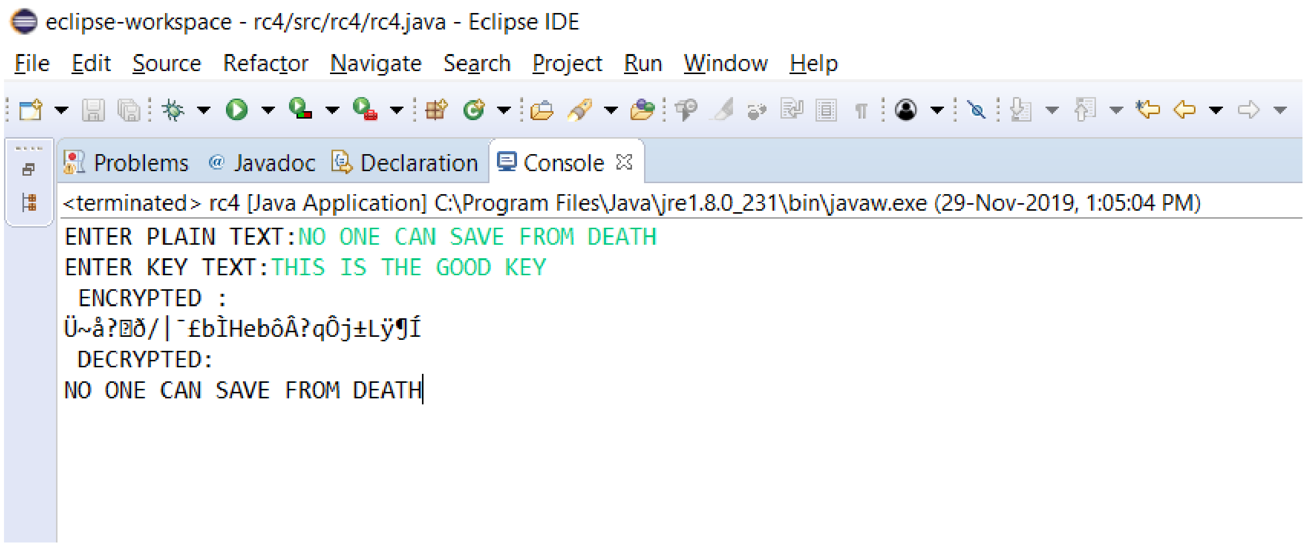
Keystream value search is done by exchanging again between elements S, but one value S stored in the K which is then used as a key stream. More details can be seen in the following pseudo code.

```
for i from 0 to 255
    S[i] = i
endfor
j = 0
    for i from 0 to 255 j=
    (j + S[i] + key[i mod
    keylength])mod 256
    swap values of S[i] and
S[j]
endfor
```

XOR is a logical operation that compares two binary bits. If the difference is worth, it will produce a value of 1. If both bits equal then, the result is 0. Then the recipient will decrypt the message by clicking XOR of return with the same key that generated the message from plain text. Figure 4 describes the encryption process.

CHAPTER 5

RESULT AND SNAPSHOT


eclipse-workspace - rc4/src/rc4/rc4.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Problems
Javadoc
Declaration
Console

```
<terminated> rc4 [Java Application] C:\Program Files\Java\jre1.8.0_231\bin\javaw.exe (29-Nov-2019, 1:05:04 PM)
ENTER PLAIN TEXT:NO ONE CAN SAVE FROM DEATH
ENTER KEY TEXT:THIS IS THE GOOD KEY
 ENCRYPTED :
Ü~å?⍰ð/|¯£bÌHebôÂ?qÔj±Lÿ¶Í
 DECRYPTED:
NO ONE CAN SAVE FROM DEATH
```

Figure 5.1: Output of the program

Figure 5.1 shows the output of the RC4 program (JAVA code) run in eclipse workspace platform

CHAPTER 6

MERITS AND DEMERITS

MERITS

- RC4 is basically a bite oriented algorithm or symmetric key cipher. It encrypts laptop files, personal computers & disks.
- It protects private & confidential data messages sent to and from secure websites. Itis also known as vernam cipher.
- RC4 cipher is extremely fast & uses small amount of ram.
- It is good for small handheld devices.

DEMERITS

- RC4 can be very slow in cases where large data needs to be encrypted by the same computer.
- It requires a third party to verify the reliability of public keys.
- Data transferred through RC4 could be compromised through middlemen who might temper with the public key stream.

CHAPTER 7
CONCLUSION AND FUTURE SCOPE

CONCLUSION

One of the weaknesses of RC4 is the high possibility of similar S-Box table; this occurs because the user key is repeated to fill 256 bytes. To overcome this can use a hash function to verify the authenticity of the cipher text and key. Another shortcoming is that RC4 encryption is the XOR between the data bytes and the pseudo-random byte stream generated from the key, then the attacker will be possible to determine some of the original message byte XOR with two sets of cipher bytes when some of the input messages known.

FUTURE SCOPE

To overcome the disadvantages of this method, can use the initialization vector that is different for each data, so for the same file will produce a different cipher text. It is not the secret values since it is used only so that every encryption process will generate a different cipher text. To further enhance the security of this method, it can also be developed a better initialization key. The use of 256-byte key will allow an intruder to perform repeated permutations. Key modification is necessary to strengthen the security level RC4 algorithm.

REFERENCES

[1]. T. D. B. Weerasinghe, “Analysis of a Modified RC4 Algorithm,” International Journal of Computer Applications, vol. 51, no. 22, 2012.

[2]. L. Stošić dan M. Bogdanović, “RC4 Stream Cipher and Possible Attacks on WEP,” International Journal of Advanced Computer Science and Applications, pp. 110-114, 2012.

[3]. P. Jindal dan B. Singh, “A Survey on RC4 Stream Cipher,” I. J. Computer Network and Information Security, pp. 37-45, 2015.

[4]. A. P. U. Siahaan, “RC4 Technique in Visual Cryptography RGB Image Encryption,” International Journal of Computer Science and Engineering, vol. 3, no. 7, pp. 1-6, 26 04 2016.

[5]. A. P. U. Siahaan, “Blum BlumShub in Generating Key in RC4,” International Journal of Science &Technoledge, vol. 4, no. 10, pp. 1-5, 2016.

[6]. L. M. Nannaka, H. Singarapu dan R. Puli, “Remodelling RC4 Algorithm for Secure Communication for WEP/WLAN Protocol,” Global Journal of Researches in Engineering, Electrical and Electronics Engineering, vol. 12, no. 5, pp. 37-39, 2012.

[7]. N. Sinha, M. Chawda dan K. Bhamidipati, “Enhancing Security of Improved RC4 Stream Cipher by Converting into Product Cipher,” International Journal of Computer Applications, vol. 94, no. 18, pp. 17-21, 2014.