http://ejournal.seaninstitute.or.id/index.php/InfoSains
Jurnal Info Sains : Informatikan dan Sains, Volume 11, No 02 September 2021
E-ISSN.2797-7889, P-ISSN.2089-3329

Use of Rc4 Method In Android-Based Sms Security Application

Oktovantua Butarbutar
Teknik Informatika, STMIK Jakarta STI&K
Email: oktovantua45@gmail.com

Keywords

Information
SystemCryptography
RC4

Abstract. In maintaining the confidentiality of SMS, it takes a way to secure information that is important or confidential, namely by encrypting the SMS text, the level of information security of the message can be increased. One way to use for data and or information security is to use cryptographic system, this application uses RC4 algorithm which has the advantage of having a high level of security and speed process. This makes RC4 the best choice for the encryption process required by the information world towards the next century. The purpose ofbya is to prevent the occurrence of information / data taker without being known by the owner.

1. INTRODUCTION

Data security and confidentiality issues are one of the most important aspects of an information system. This is very much related to how important the information is to be sent and received by people interested in the data. Information will no longer be useful if in the middle of the way data is hijacked or intercepted by unauthorized persons [1]. The information on the data will be lost so that the recipient will get different information . One of the ways used for data or information security is to use cryptographic systems, this application uses RC4 algorithms that have the advantage of having a high level of security and speed processes [2]. This makes RC4 the best choice for the encryption process required by the information world towards the next step. In maintaining the confidentiality of SMS, it takes a way to secure information that is important or confidential, namely by encrypting the SMS text, the level of information security of the message can be increased. The author in addressing the problem of sending messages, trying to create a messaging application with RC4 algorithm to encrypt data running on android operating system so that owners of android-based mobile phones (phones) can exchange SMS data more safely and conveniently [1], [3].

A text messaging service component of most telephones, Internet, and mobile-device systems is known as short message service (SMS) [4]. Standardized communication protocols are used to permit smart phones to transfer short text messages. Short message service is also commonly referred to as a “text message.” The user can conduct a message of up to 160 characters to another device with a SMS. In SMS, longer messages will automatically be fragmented into several parts. This type of text messaging is supported by most cell phones. The formal name for text messaging is SMS. Short message service is a way to conduct short, text-only messages from one phone to another. These messages are usually conducted over a cellular data network. The procedure for conducting SMS is launching the Messages application on the phone. Tap on the Compose Message button. Enter the phone number or name of the contact you want to text. Type your message and finally hit Send. These days, there exist a number of security issues and vulnerabilities related to SMS [3].

2. METHOD

2.1 Rc4 Method

Cryptography is derived from the Greek "Cryptos" meaning "Secret" and "graphein" means "writing". So, cryptography means "Secret Writing". Definition stated in [SCH96] : Cryptography is the science and art of keeping messages safe. The Rivest Code 4 (RC4) cryptographic algorithm is one of the symmetric key algorithms created by RSA Data Security Inc (RSADSI) in the form of a chpper stream [4]. The algorithm was invented in 1987 by Ronald Rivest and became a symbol of

RSA security (short for three inventor names: Rivest Shamir Adleman) [10]. RC4 uses akey length from 1 to 256 bytesthat is used to initialize a 256-byte table. This table is used for the following generations of pseudo randoms that use XOR with plaintext to generate ciphertext [5], [6]. Each element in the table is exchanged at least once. RC4 is one type of ciper stream, which is processing units or data inputs at a time. In this way encryption or decryption can be implemented at variable lengths. This algorithm does not have to wait for a certain amount of data input before it is processed, or add additional bytes to encrypt. Rc4 encryption methods are very fast approximately 10 times faster than DES.

2.2 RC4 Encryption and Decryption

The encryption and decryption processes have the same process so that only one function is performed to run both processes. The following will be given a section describing the series of processes executed to describe or decrypt [8]:

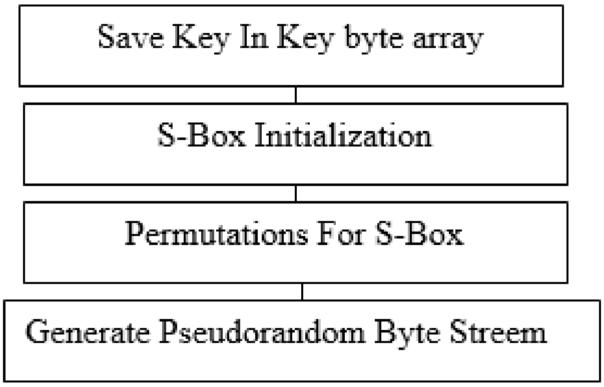

Figure 1 Encryption and Decryption Process

3. RESULTS AND DISCUSSION

3.1 Encryption-Decryption Process Analysis

Rc4 Stream Cipher algorithm to perform encryption, the process begins with the initialization of the first Sbox, S [0], S [1],....., S [255], with the numbers 0 to 255. First fill sequentially [0] = 0, S [1] = 1,...., S [255] = 255. Then initialize another array (another S-Box), e.g. array K with a length of 256. Fill array K with the keys to repeat until the entire array K [0], K [1],....., K [255] are fully populated.

1. S-Box initialization process (Array S)

```
For i = 0 to 255
S [i] = i
```

2. S-Box initialization process (Array K)

```
// Array key array with key length "leght"
For i = 0 to 255
S [i] = i
```

3. Then perform the S-Box randomization step

```
I = 0; j = 0
For i = 0 to 255
{
    J = (j + S [i] + [k] mod 256)
    Swap S [i] And S [j]
}
```

4. Create a byte pseudorandom

```
i = ( i + 1 ) mod 256
j = ( j + S [i]) mod 256
swap S [i] And S [j]
```

t = (S [i] + S [j]) mod 256
K = S [t]

5. Byte K is XOR-kan with plaintext to produce ciphertext or in-XOR- kan with ciphertext to produce plaintext.

Here is the implementation of RC4 algorithm with 256 bytes mode

1.Initialization of S-Box with a length of 256 bytes, with S [0] = 0, S [1] = 1,
S [2] = 2 and S [3] = 3,....., S [255] so that array S becomes:

Table 1. S-Box with a length of 256 bytes

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

2. Initialize the 10-byte key array, Ki. Suppose the key is taken from 10 bytes i.e. "OPEN[space]key" then the sentence will be converted into Decimal form "66 85 75 65 32 107 117 110 99 105". Repeat the key until it meets the entire K array so that array K becomes:

Iter-iKey-charKey[i]Sbox [i]
1 B 66 0
2 U 85 1
3 K 75 2
4 A 65 3
5 [space] 32 4
6 K 107 5
7 U 117 6
8 N 110 7
9 C 99 8
10 I 105 9
11 B 66 10
12 U 85 11
13 K 75 12
14 A 65 13
15 [space] 32 14
16 K 107 15
17 U 117 16
18 N 110 17
19 C 99 18
20 I 105 19
21 B 66 20
22 U 85 21
23 K 75 22
24 A 65 23
25 [space] 32 24
26 K 107 25
27 U 117 26
28 N 110 27
29 C 99 28
30 I 105 29
31 B 66 30
32 U 85 31
33 K 75 32
34 A 65 33
35 [space] 32 34
36 K 107 35
37 U 117 36
38 N 110 37
39 C 99 38
40 I 105 39
41 B 66 40
42 U 85 41
43 K 75 42
44 A 65 43
45 [space] 32 44

46	K	107	45	
47	U	117	46	
48	N	110	47	
49	C	99	48	
50	I	105	49	
51	B	66	50	
52	U	85	51	
53	K	75	52	
54	A	65	53	
55	[space]		32	54
....	....	....	....	
....	....	....	....	
....	....	....	....	
....	....	....	....	
255	[space]		32	254
256	K	107	255	

3. Next mix the operation where it will use the variable i And j to index array S [i] And K [i]. First we give the initial value for i And j with 0. The mixing operation is a repeat of the formula (i + S [i] + K [j]) mod 256 followed by the exchange of S [i] with S [j]. For this example, because we are using an array with a length of 256 bytes so the algorithm becomes:

For i = 0 to 256
j = (j + S [i] + K [i]) mod 256
swap S [i] And S [j]

4. With the algorithm as above So with the initial value i = 0 to i = 256 will generate an array of S as below:

Iteration to1:
i = 0, So
j = (j + S[i] + K [i] mod 256
= (j + S[0] + K [0] mod 256
(0+0+66) mod 256
= 66
Swap S [0] And S [66]

Iteration to2:
i = 0, So
j = (j + S[i] + K [i] mod 256
= (j + S[1] + K [1] mod 256
(66+1+85) mod 256
= 152
Swap S [1] And S [152]

Iteration to3:
i = 1, So
j = (j + S[i] + K [i] mod 256
= (j + S[2] + K [2] mod 256
(152+2+75) mod 256
= 229
Swap S [2] And S [229]

Iteration to4:
i = 3, So
j = (j + S[i] + K [i] mod 256
= (j + S[3] + K [3] mod 256
(229+3+65) mod 256
= 41
Swap S [3] And S [41]

Iteration to5:
i = 4, So
j = (j + S[i] + K [i] mod 256
= (j + S[4] + K [4] mod 256
=(41+4+32) mod 256
= 77
Swap S [4] And S [77]

Iteration to6:
i = 5, So
j = (j + S[i] + K [i] mod 256
= (j + S[5] + K [5] mod 256
= (77+5+107) mod 256
= 189
Swap S [5] And S [189]

Iteration to7:
i = 6, So
j = (j + S[i] + K [i] mod 256
= (j + S[6] + K [6] mod 256
= (189+6+117) mod 256
= 56
Swap S [6] And S [56]

Iteration to 8:
i = 7, So
j = (j + S[i] + K [i] mod 256
= (j + S[7] + K [7] mod 256
= (56+7+110) mod 256
= 173
Swap S [7] And S [173]

Iteration to9:
i = 8, So
j = (j + S[i] + K [i] mod 256
= (j + S[8] + K [8] mod 256
= (173+8+99) mod 256
= 24
Swap S [8] And S [24]

Iteration to10:
i = 9, So
j = (j + S[i] + K [i] mod 256
= (j + S[9] + K [9] mod 256
= (24+9+105) mod 256
= 138
Swap S [9] And S [138]

Iteration to254:
i = 253, So
j = (j + S[i] + K [i] mod 256

= (j + S[253] + K [253] mod 256
= (27+67+65) mod 256
= 159
Swap S [5] And S [159]
Iteration to255:
i = 254, So
j = (j + S[i] + K [i] mod 256
= (j + S[254] + K [254] mod 256
= (159+99+32) mod 256
= 34
Swap S [5] And S [34]
Iteration to256:
i = 255, So
j = (j + S[i] + K [i] mod 256
= (j + S[255] + K [255] mod 256
= (34+94+107) mod 256
= 235
Swap S [255] And S [235]

5. Next is the encryption process which is XOR-kan pseudorandom byte with plaintext, e.g. plaintext "ITU[space]DIA". Plaintext consists of 7 characters So occurs 7 iterations. Before iterating, convert the characters to binary number forms.

Character	Decimal	Biner
I	73	0100 1001
T	84	0101 0100
U	85	0101 0101
[space]	32	0010 0000
D	68	0100 0100
I	73	0100 1001
A	65	0100 0001

Here's Iteration to1: Initialize i And j with i = 0; j = 0;
i = (i + 1) mod 256
= (0 + 1)
= 1
And
j = (j + S[i] mod 256)
= (j + S[1]) mod 256
= (0 + 44) mod 256
= 44
Swap S[1] And S[44]
t = (S[i] + S[j]) mod 256
= (S[1] + S[44]) mod 256
= (71+ 44) mod 256
= 115
K = S[t] = S[15] = 14 = 0000 1110
Byte K di-XOR-with plaintext “I”
Plainteks I
73
0100 1001
Key (K) 0000 1110
XOR 0100 0111
Chipertext 71
G
Iteration to2:
i = (i + 1) mod 256
= (1 + 1)
= 2
And
j = (j + S[i] mod 256)
= (44 + S[2]) mod 256
= (44 + 229) mod 256
= 17
Swap S[2] And S[17]
t = (S[i] + S[j]) mod 256
= (S[2] + S[17]) mod 256
= (117 + 229) mod 256
= 90
K = S[t] = S[90] = 202 = 1000 1000
Byte K di-XOR with plaintext “T”
Plainteks T
84
0100 0100
Key (K) 0000 0010
XOR 0101 1100
Chipertext 92
\

Iteration to3:
i = (i + 1) mod 256
= (2 + 1)
= 3
And
j = (j + S[i] mod 256)
= (j + S[3]) mod 256
= (17 + 160) mod 256
= 177
Swap S[3] And S[177]
t = (S[i] + S[j]) mod 256
= (S[3] + S[15]) mod 256
= (12 + 160) mod 256
= 172
K = S[t] = S[172] = 202 = 1100 1010
Byte K di-XOR-with plaintext “U”
Plainteks U
85
0101 0101
Key (K) 1100 1010
XOR 1001 1111
Chipertext 159

Y

Iteration to4:
i = (i + 1) mod 256
= (3 + 1)
= 4
And
j = (j + S[i] mod 256)
= (j + S[4]) mod 256
= (177 + 106) mod 256
= 27
Swap S[4] And S[27]
t = (S[i] + S[j]) mod 256
= (S[4] + S[27]) mod 256
= (51 + 106) mod 256
= 157
K = S[t] = S[157] = 75 = 0100 1010
Byte K di-XOR-with plaintext “[Space]”
Plainteks [Space]
32
0010 0000
Key (K) 0100 1011
XOR 0110 1011
Chipertext 107
K

Iteration to5:
i = (i + 1) mod 256
= (4 + 1)
= 5
And
j = (j + S[i] mod 256)
= (j + S[5]) mod 256
= (27 + 185) mod 256
= 212
Swap S[5] And S[212]
t = (S[i] + S[j]) mod 256
= (S[5] + S[212]) mod 256
= (116 + 185) mod 256
= 45
K = S[t] = S[45] = 223 = 1101 1111
Byte K di-XOR-with plaintext “D”
Plainteks D
68
0100 0100
Key (K) 1101 1111
XOR 1001 1011
Chipertext 155
>

Iteration to6:
i = (i + 1) mod 256
= (5 + 1)
= 6
And
j = (j + S[i] mod 256)
= (j + S[6]) mod 256
= (212 + 56) mod 256
= 12
Swap S[6] And S[12]
t = (S[i] + S[j]) mod 256
= (S[6] + S[12]) mod 256
= (141 + 56) mod 256
= 197
K = S[t] = S[197] = 42 = 0010 1010
Byte K di-XOR-with plaintext “I”
Plainteks I
73
0100 1001
Key (K) 0010 1010
XOR 0110 0011
Chipertext 99
C

Iteration to7:
i = (i + 1) mod 256
= (6 + 1)
= 7
And
j = (j + S[i] mod 256)
= (j + S[7]) mod 256
= (12 + 48) mod 256
= 60
Swap S[7] And S[60]
t = (S[i] + S[j]) mod 256
= (S[7] + S[60]) mod 256
= (64 + 48) mod 256
= 112
K = S[t] = S[112] = 42 = 0010 0111
Byte K di-XOR-with plaintext “A”
Plainteks A
65
0100 0001
Key (K) 0010 0111
XOR 0110 0110
Chipertext 102
F

So the results of encryption obtained after going through several iterations are as follows: len_tektiter–

iiter-j	sbox [i]		Sbox [j]key		ploianteks		asci_ code		chiper ascii_char
1	1	66	44	44	66	I	73	71	G
2	2	152	229	17	85	T	84	92	\
3	3	229	160	177	75	U	85	159	Ÿ
4	4	77	185	27	65	[space]	32	107	k
5	5	189	56	212	32	D	68	155	>
6	6	56	48	12	107	I	73	99	c
7	7	173	243	60	117	A	65	102	f

6. Next is the decryption process which is XOR-kan pseudo random byte with Chipertext, And Chipertext is "G \ Ÿ k > c f". Chipertext consists of 7 characters so occurs 7 iterations. Before iterating, convert characters to binary number form

Character	Decimal	Biner
G	71	0100 0111
\	92	0101 1100
Ÿ	159	1001 1111
k	107	0110 1011
>	155	1001 1011
c	99	0110 0011
f	102	0110 0110

The data is sent in the form of a chipertext so that once it reaches the recipient the message can be re-converted to plaintext by XOR-kan with the same key.

Chipertext G \ Ÿ k > c f
Kunci BUKA [space] key
PlaintextITU [space] DIA

Berikut iterasi 1:
Plaintext G
71
0100 0111
Key (K) 0000 1110
XOR 0100 1001
Chipertext 73
I

Here's iteration 2 :

Plaintext \
92
0101 1100
Key (K) 0000 1000
XOR 0101 0100
Chipertext 84
T

Berikut iterasi 3:

Plaintext Ÿ
159
1001 1111
Key (K) 1100 1010
XOR 0101 0101
Chipertext 85
U

Here's iteration 4:

Plaintext K
107
0110 1011
Key (K) 0100 1011
XOR 0010 0000
Chipertext 32
[Space]

Here's iteration 5:

Plaintext >
155
1001 1011
Key (K) 1101 1111
XOR 0100 0100
Chipertext 68
D

Here's iteration 6:

Plaintext C
99
0110 0011
Key (K) 0010 1010
XOR 0100 1001
Chipertext 73
I

Here's iteration 7:
Plaintext F
102
0110 0110
Key (K) 0010 0111
XOR 0100 0001
Chipertext 65
A

4. CONCLUSION

With aAndyes SMSencryption and decryption application with RC4 algorithm, it can provide convenience for users to encrypt SMS messages without having to do calculations. This application uses RC4 algorithm which has the advantage of having a high level of security and speed processes. Android-based SMS encryption apps can help users describe SMS messages before they're sent.

REFERENCE

[1] S. K. Fatima, S. G. Fatima, S. A. Sattar, and A. Sheela, "An advanced data security method in WSN," *Int. J. Adv. Res. Eng. Technol.*, vol. 10, no. 2, 2019, doi: 10.34218/IJARET.10.2.2019.026.

[2] S. Pirbhulal, O. W. Samuel, W. Wu, A. K. Sangaiah, and G. Li, "A joint resource-aware and medical data security framework for wearable healthcare systems," *Futur. Gener. Comput. Syst.*, vol. 95, 2019, doi: 10.1016/j.future.2019.01.008.

[3] R. Perangin-angin, I. K. Jaya, and ..., "Analisa Alokasi Memori dan Kecepatan Kriptograpi Simetris Dalam Enkripsi dan Dekripsi," *J. Inf. ...*, 2019.

[4] A. I. Sallam, O. S. Faragallah, and E. S. M. El-Rabaie, "HEVC Selective Encryption Using RC6 Block Cipher Technique," *IEEE Trans. Multimed.*, vol. 20, no. 7, 2018, doi: 10.1109/TMM.2017.2777470.

[5] A. I. Sallam, E. S. M. El-Rabaie, and O. S. Faragallah, "CABAC-based selective encryption for HEVC using RC6 in different operation modes," *Multimed. Tools Appl.*, vol. 77, no. 21, 2018, doi: 10.1007/s11042-018-5994-5.

[6] A. Subandi, M. S. Lydia, R. W. Sembiring, M. Zarlis, and S. Efendi, "Vigenere cipher algorithm modification by adopting RC6 key expansion and double encryption process," in *IOP Conference Series: Materials Science and Engineering*, 2018, vol. 420, no. 1, doi: 10.1088/1757-899X/420/1/012119.

[7] A. Ouertani *et al.*, "Two new secreted proteases generate a casein-derived antimicrobial peptide in Bacillus cereus food born isolate leading to bacterial competition in milk," *Front. Microbiol.*, vol. 9, no. JUN, 2018, doi: 10.3389/fmicb.2018.01148.

[8] Saranya *et al.*, "Spritz — a spongy RC4-like stream cipher and hash function," *IOP Conf. Ser. Mater. Sci. Eng.*, vol. 3, no. 3, 2018.

[9] Rusmala and D. Prasti, "Implementasi Metode Rail Fence Chiper dan Row Transposition Chiper Pada Mata Kuliah Kriptograpi," *Ilm. d'Computare*, vol. 9, 2019.

[10] A. Hidayat, K. A. Azhari, and D. Setiana, "Perbandingan Penggunaan Memory Dan Cpu Menggunakan Kriptograpi AES," *JUTEKIN (Jurnal Tek. Inform.*, vol. 6, no. 2, 2018.