THE INTERNATIONAL JOURNAL OF SCIENCE & TECHNOLEDGE

Blum Blum Shub in Generating Key in RC4

Andysah Putera Utama Siahaan
Faculty, Department of Computer Science,
Universitas Pembangunan Panca Budi, Medan, Sumatera Utara, Indonesia

Abstract:
In the manufacture of RC4 encryption, a user always input the key as a security approval. The encryptor usually makes the core. Sometimes the encryptor does not feel that the key made is breakable. The Blum Blum Shub method can generate random key more secure. It means that the key generated can help the data from being solved by a hijacker. Blum Blum Shub uses two prime numbers to generate a key. When attached in RC4, a user does not need to create its key; and fully produce by the Blum Blum Shub module. The ciphertext will be more secure and robust after being combined with Blum Blum Shub.

Keywords: *Cryptography, Blum Blum Shub, RC4*

1. Introduction

RC4 is one of a cryptographic model that uses 1 to 256 characters as the security key [1][4]. However, the key created is not too secure. When attaching the key, people always makes it which can easily remember. The key generated is repeated until the length of the plaintext is reached. The process of this algorithm is to round the position in SBOX array. The SBOX array has 256 bytes' length. Each post has its number used to generate a pseudorandom number to be xor with the plaintext. The weakness of this technique in making the core is it not safe. The strength core must be applied to make the strength ciphertext as well. Blum Shub method offers the advance of security technique. The algorithm will generate the key. Some of the advantages of using this approach are the total amount of character, and the range of ASCII can be determined beforehand.

2. Theories

Two closely-related pseudo-random sequence generators presented: The lIP generator, with input P a prime, outputs the quotient digits obtained on dividing by P. The x mod Generator with data N, Xo (where N P. Q is a product of distinct primes, each congruent to 3 mod 4, and x0 is a quadratic residue mod N), outputs bob1 b2" where bit parity (xi) and xi+ x mod N [2][3].

The RC4 algorithm is one of symmetric cryptography algorithms which commonly used for encryption and decryption. RC4 derived from the RSA Data Security, Inc [5]. In symmetric ciphers, both the encryption and decryption use the same keys. It is designed to be easily performed even in large amounts of data. Symmetric ciphers can operate in block or stream mode. In block mode, the message will split into several fixed size blocks and each block will be encrypted one by one while in flow mode the message will be encrypted from the first character to the end of the message.

RC4 has the SBOX to save the random position which will be applied to plaintext after. The SBOX will be numbered from 0 to 255 as an array [6]. RC4 uses two indices, i and j in the algorithm. The index i is used to ensuring that an element changed while the index j will make sure that an item changed randomly [7][8]. In essence, this encryption algorithm will generate pseudo-random byte of the key that has xor operation on plaintext to produce the ciphertext. To obtain the plaintext back from the ciphertext, the decryption module performs the same operation as the encryption does. Figure 1 tells how to get the SBOX data. The SBOX obtained by mixing the core and the value of j.

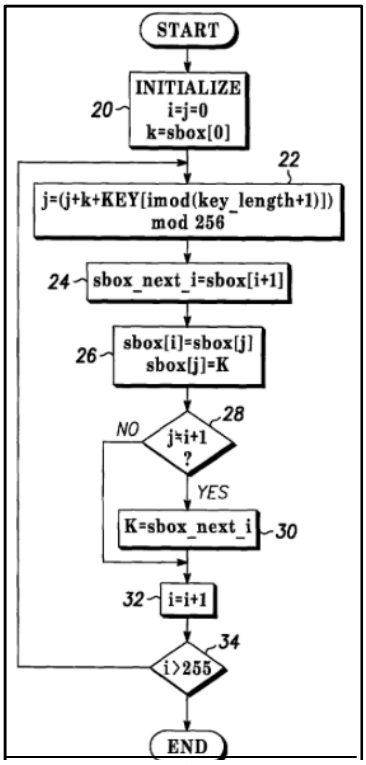

Figure 1: Process of theSBOX generator

After the SBOX is compiled, the encryption or decryption process can be carried on. The main part of this method is to perform the XOR operation to the plaintext which is taken from the SBOX. This is a repeated process since the length of the message is greater than the SBOX or greater than 256 characters. Figure 2 is the flowchart of RC4 encryption and decryption.

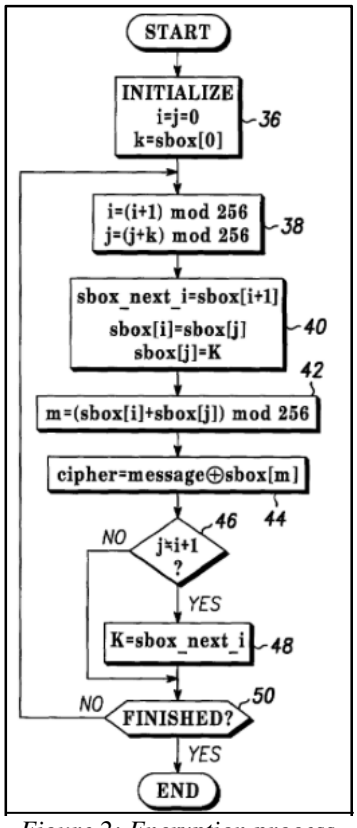

Figure 2: Encryption process

Blum Blum Shub (BBS) is a random number which is generated by a pseudo-random number generator. It was created by Lenore Blum in 1986, Manuel Blum and Michael Shub that is derived from Michael O. Rabin's oblivious transfer mapping. The formula of BBS is shown below:

$$xn + 1 = xn^2 mod\ (p.q) \qquad (1)$$

Where:

p & q : prime numbers
x : seed

3. Methodology

The BBS module takes apart in generating key as input in the RC4 algorithm. The key in RC4 is in ASCII format which is represented by 0 to 255 since the input key inserted by typing them on a keyboard. In this case, the number generated by BBS must be limited in a range of 0 - 255. To perform this case, it needs to be added a modulo expression to turn its number back in that range. The BBS module will be repeated until the maximum request. If the requested key is 20 characters, it will spin to 20 times and result in 20 random numbers in ASCII format. Table 1 shows the result of generating 20 numbers in range 33-127.

1	2	3	4	5	6	7	8	9	10
79	50	75	70	122	107	107	74	95	39
O	**2**	**K**	**F**	**z**	**k**	**k**	**J**	**_**	**'**
11	12	13	14	15	16	17	18	19	20
56	71	35	107	62	57	68	124	48	40
8	**G**	**#**	**k**	**>**	**9**	**D**	**|**	**0**	**(**

Table 1: Result of generating 20 random numbers

The key generated by BBS is "**O2KFzkkJ_'8G#k>9D|0(**". Then it generates the SBOX using the key obtained. Table 2 shows the SBOX value after reposition using SBOX module.

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

Table 2: SBOX value

The SBOX is applied to plaintext and perform the encryption. Assume the plaintext is "THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG" and the key is "O2KFzkkJ_'8G#k>9Dl0(". Table 3 show the result of encryption.

ENCRYPTION					
NO	**PT**		**K**	**CT**	
1	T	84	56	108	l
2	H	72	122	50	2
3	E	69	156	217	Ù
4		32	247	215	×
5	Q	81	133	212	Ô
6	U	85	178	231	ç
7	I	73	141	196	Ä
8	C	67	236	175	¯
9	K	75	172	231	ç
10		32	63	31	
11	B	66	58	120	x
12	R	82	148	198	Æ
13	O	79	129	206	Î
14	W	87	141	218	Ú
15	N	78	130	204	Ì
16		32	182	150	–
17	F	70	25	95	_
18	O	79	240	191	¿
19	X	88	53	109	m
20		32	141	173	
21	J	74	167	237	í
22	U	85	165	240	ð
23	M	77	74	7	
24	P	80	236	188	¼
25	S	83	67	16	
26		32	4	36	$
27	O	79	43	100	d
28	V	86	75	29	
29	E	69	91	30	-
30	R	82	253	175	¯
31		32	248	216	Ø
32	T	84	213	129	
33	H	72	28	84	T
34	E	69	136	205	Í
35		32	254	222	Þ
36	L	76	227	175	¯
37	A	65	152	217	Ù
38	Z	90	163	249	ù
39	Y	89	219	130	‚
40		32	162	130	‚
41	D	68	88	28	
42	O	79	203	132	„
43	G	71	197	130	‚

Table 3: The result of encryption

There are several unprinted/box characters. If the encryption results between 0 and 32, it will be not printed on the screen. It shows like a spacebar only. For example, the value of item number one. The plain character is "T" which ASCII is 84. K is the value that will be xor with the plain character. First, i and j are set to zero. Let's take a look at the illustration below.

Plain Char = T
PT = 84

i = 0

j = 0
i = (i + 1) mod 256
= 1
j = (j + S[i]) mod 256
= 0 + 137
= 137

S[i] = S[1] = 137
S[j] = S[137] = 66 then swap
S[i] = S[1] = 66
S[j] = S[137] = 137

t = (S[i] + S[j]) mod 256
= (66 + 137) mod 256
= 203

K = S[t]
= S[203]
= 56

CT = PT ⊕ K
= 84 ⊕ 56
= 108

The calculation gives the correct result. The plain character 84 is turned to 108 after being encrypted with 56. The decryption does the same way with the encryption. The cipher character 108 will have xor with 56 too, and the final result will be 203 as well. By attaching BBS in RC4, the provided key will be more powerful and hard to guess. Someone does not have to create the key manually. BBS uses the prime numbers to compose the single key randomly. It makes the sequence of the characters are invulnerable.

4. Conclusion

From the discussion earlier, it can be concluded that Blum Blum Shub can give a contribution to RC4. The key is a strong determinant whether the encryption method is powerful. Blum Blum Shub is one of the pseudo-random generators that can be applied in the RC4 technique. The math calculation is not difficult to learn. The RC4 key can be given either by user type or pseudo-random number. All the important thing is to generate the difficult key to produce the powerful ciphertext. The attacker will be hard to guess the key used by the encryption and decryption participants.

5. References

i. Kadry, S., & Smaili, M. (2010). An Improvement of RC4 Cipher Using Vigenère Cipher. Lebanon: Lebanese University.
ii. Blum, L., Blum, M., & Shub, M. (1986). A Simple Unpredictable Pseudo-Random Number Generator. SIAM Journal on Computing, 364-383.
iii. Blum, L., Blum, M., & Shub, M. (1982). Comparison of Two Pseudo-Random Number Generators. Advances in Cryptology: Proceedings of Crypto (hal. 61-78). Plenum.
iv. Siahaan, A. P. (2016, 04 26). RC4 Technique in Visual Cryptography RGB Image Encryption. International Journal of Computer Science and Engineering, 3(7), 1-6. doi:10.14445/23488387/IJCSE-V3I7P101.
v. Sidorenko, A., & Schoenmakers, B. (2005). Concrete Security of the Blum-Blum-Shub Pseudorandom Generator.
vi. Weerasinghe, T. D. (2012). Analysis of a Modified RC4 Algorithm. International Journal of Computer Applications, 51(22).
vii. Stošić, L., & Bogdanović, M. (2012). RC4 Stream Cipher and Possible Attacks on WEP. International Journal of Advanced Computer Science and Applications, 110-114.
viii. Jindal, P., & Singh, B. (2015). A Survey on RC4 Stream Cipher. I. J. Computer Network and Information Security, 37-45.