ISSN: 0067-2904
GIF: 0.851

A Modified Key Scheduling Algorithm for RC4

Sarab M. Hameed*, Israa Nafea Mahmood
Department of Computer Science, College of Science, University of Baghdad, Baghdad, Iraq

Abstract
Rivest Cipher 4 (RC4) is an efficient stream cipher that is commonly used in internet protocols. However, there are several flaws in the key scheduling algorithm (KSA) of RC4. The contribution of this paper is to overcome some of these weaknesses by proposing a new version of KSA coined as modified KSA($mKSA$). In $mKSA$, the initial state of the S array is suggested to contain random values instead of the identity permutation. Moreover, the permutation of the S array is modified to depend on the key value itself. The proposed $mKSA$ performance is assessed in terms of *cipher secrecy*, *randomness test* and *time* under a set of experiments with variable key size and different plaintext size. The results show that the RC4 with $mKSA$ improves the randomness and secrecy with comparable encryption time with the canonical RC4 version.

Keywords*:* Cipher Secrecy, Key Scheduling Algorithm, RC4, Stream Cipher**.**

خوارزمية جدولة المفتاح المعدلة RC4

سراب مجيد حميد*، اسراء نافع محمود
قسم علوم الحاسبات، كلية العلوم، جامعة بغداد، بغداد، العراق

الخلاصة
تعد خوارزمية تشفير ريفست (RC4) من خوارزمية تشفير التدفقي الكفوءة والتي يشيع استخدامها في بروتوكولات الانترنت. ومع ذلك هنالك العديد من العيوب في خوارزمية جدولة المفتاح (KSA) . يساهم هذا البحث في التغلب على بعض نقاط الضعف من خلال اقتراح صيغة جديدة لخوارزمية جدولة المفتاح (KSA) و التي اطلق عليها خوارزمية جدولة المفتاح المعدلة ($mKSA$) . في $mKSA$ الحالة الاولية للمصفوفة S سوف تحتوي علي قيم عشوائية بدلا من تباديل الهوية و التقليب في مصفوفة S سوف يعتمد على قيمة المفتاح. تم تقييم اداء $mKSA$ المقترحة من حيث السرية و اختيار العشوائية و الوقت في اطار مجموعة من التجارب مع اعتماد حجم مختلف للمفتاح و حجم مختلف للرسائل الغير مشفرة. اظهرت النتائج للخوارزمية RC4 التي تستخدم $mKSA$ تحسنا ملحوظا في اختبار العشوائية و السرية مع تحقيق وقتا مقاربة للتشفير في خوارزمية RC4 الاصلية .

1. Introduction
Ron Rivest designed the RC4 algorithm in 1987 but the algorithm kept secret until it was posted to the cypherpunks mailing list in 1994. RC4 is the most acceptable stream cipher; it is used in many internet protocols such as Wired Equivalent Privacy (WEP), Wireless Protected Access (WPA) and Secure Socket Layer/ Transport Layer Security (SSL/TLS). It is also used in application such as Skype [1]. RC4 proves its efficiency in both hardware and software and speed. It is very simple and fast comparable to other encryption algorithms. RC4 algorithm mainly consists of two stages: the Key Scheduling Algorithm (KSA) to generate, from the key, an initial permutation of the S array and the Pseudo Random Generation Algorithm (PRGA) to generate the key stream. The simple structure of the KSA, however, is considered as the major weakness of the RC4 [2].

*Email: sarab_majeed@yahoo.com

The literatures have exposed several weaknesses in RC4. Fluhrer *et al.* [3] analyzed the KSA and discovered that knowing even a small portion of the secret key can bewilder the secrecy gain of the algorithm. Based on the above weakness, Klein [4] presented a new weakness and proved that there is a relation between the internal state and the generated output.

Many modifications were indeed applied to RC4 in order to enhance the security and to reduce its weakness. Xie and Pan [5] presented an improved RC4 using two keys $K1$ and $K2$ and two states state arrays $S1$ and $S2$ in order to break down the relation between the internal state and the generated output. The improved RC4 is proved to be faster and more secure than original RC4. Additional, but simple modification is suggested by Weerasinghe [6] to the RC4 using bit-wise XOR operation between the generated cipher of the original RC4 and the variable j that points to S array in order to increase the secrecy.

In this paper, a new version of KSA is suggested in an attempt to increase the security of RC4 and to get rid of the weakness related to initial permutation of the S array and the permutation process of the S array. The rest of the paper is organized as follows: In Section 2, a brief description of RC4 is presented. In section, we introduce the proposed *modified* key scheduling algorithm coined as $(mKSA)$ for RC4. Section 4 presents and analyzes the results. Finally, section 5 concludes the work of this paper.

2. Description of RC4

RC4 is defined as symmetric key, binary additive stream cipher that uses variable key size ranged between 8 to 2048 bits. The RC4 produces a pseudorandom stream that is XORed with the plaintext to generate the cipher text. To recover the plaintext, the key is regenerated and XORed with the cipher text. The RC4 consists of two main stages the KSA and PRGA [7]. The KSA uses the symmetric key to permute an array S containing 256 entries. S array is initialized with identity permutation ranging from 0 to 255. Then, a 256-iteration loop is used to generate a random permutation of the array S, where the entries of the S array are continually swapped using the key value [8]. To this end, the KSA can be summarized in algorithm1.

Algorithm 1: RC4 Key Scheduling (K)

```
$set\ N \leftarrow 256$
$set\ l \leftarrow |K|$
$set\ j \leftarrow 0$
$\boldsymbol{for}\ i \leftarrow 0\ to\ N-1\ \boldsymbol{do}$
 $set\ S_i \leftarrow i$
$\boldsymbol{for}\ i \leftarrow 0\ to\ N-1\ \boldsymbol{do}$
 $\boldsymbol{begin}$
   $j \leftarrow (j + S_i + k_{i\ mod\ l})\ mod\ N$
   $swap\ (S_i, S_j)$
 $\boldsymbol{endfor}$
```

The objective of PRGA is to generate a sequence of key stream. In the PRGA, two indices i, j are initialized to zero. In each iteration, i is recomputed as $(i+1)\ mod\ 256$ and j is recomputed as$(j + S[i])\ mod\ 256$, and then a swap operation is conducted between$S[i]$ and $S[j]$. The key stream that is $XORed$ with plaintext is generated as $S[(S[i] + S[j])mod\ 256]$. The steps for generating key stream are clarified in algorithm 2.

Algorithm 2: RC4 Pseudo Random Generation (S)

```
$set\ i \leftarrow 0$
$set\ j \leftarrow 0$
$set\ l \leftarrow 0$
$set\ N \leftarrow 256$
$\boldsymbol{while}\ (true)$
$\boldsymbol{begin}$
 $i \leftarrow (i+1)\ mod\ N$
$j \leftarrow (j + S_i)\ mod\ N$
$swap\ (S_i, S_j)$
$set\ t \leftarrow (S_i + S_j)\ mod\ N$
$K'_l \leftarrow S_t$
$l \leftarrow l + 1$
$\boldsymbol{endwhile}$
```

3. The Proposed Key Scheduling Algorithm

Due to the bounded structure of KSA, short characteristics can eventually be reflected from RC4. For example, KSA uses only identity permutation to initialize the S array. Moreover, the permutations of KSA are not uniform randomly. In this section, a new version of KSA called *modified* KSA ($mKSA$) is proposed to generate an initial permutation of the S array and to shuffle the S array in an attempt to harness its strength.

The $mKSA$ initializes the S array with randomly unique values ranging from 0 to 255 instead of the identity permutation. This adds additional difficulties to the attacker to know the true initial state of the S array.

Moreover, the S array permutation is suggested to be depended on the secret key of l bytes length. The secret key contains distinct values representing the indices of the S array. The permutation starts by dividing the S array into groups of l bytes. Then, successive groups of l bytes of the S array are rearranged according to K in which each byte of S is substituted by the corresponding S_K. The $mKSA$ can be summarized in algorithm 3.

Algorithm 3: $\boldsymbol{mKSA}$ (K, S)

```
$set\ S' \leftarrow S$
$set\ c \leftarrow 0$
$set\ N \leftarrow 256$
$l \leftarrow |K|$
$\boldsymbol{for}\ i \leftarrow 0\ to\ N-1\ \boldsymbol{do}$
$\boldsymbol{Begin}$
$\boldsymbol{if}\ (i\ mod\ K_l = 0)$
   $c \leftarrow c + 1$
$set\ t \leftarrow (K_{(i\ mod\ l)} + l \times c)\ mod\ N$
$set\ S_t \leftarrow S'_i$
$\boldsymbol{Endfor}$
```

The proposed $mKSA$ can be described formally, as a function with two inputs K and S and one output (S) as follows.

$mKSA: K \times S \rightarrow S$

$K = \{0, \ldots, 255\}^l$

$S = \{0, \ldots, 255\}^{256}$

The key space of RC4 with $mKSA$ is larger than the original RC4. There are 256! possibilities of guessing the S array and $2^{8\times l}$ possibility of guessing the key since the attacker should guess the key and the true initial status of the S array. While the key space of original RC4 is$2^{8\times l}$. This means that RC4 with $mKSA$ is more resistant to attack than original RC4.

4. Performance Evaluation

Different criteria can be used to measure the security level and performance of a given encryption algorithm. In this paper, three measurements: the *cipher secrecy*, the *randomness test* of the *National Institute of Standards and Technology* (NIST) statistical test suite and *time* are used to evaluate the proposed key scheduling algorithm.

The performance of the proposed $mKSA$ is analyzed under four different key sizes, in bytes 32, 64, 128 and 256 and four different plaintext sizes in bytes 128, 256, 512 and 1024.

4.1 Cipher Secrecy

Elwood Shannon the father of information theory had some theories related to entropy and secrecy which can be used to measure the cryptosystem security. The cipher secrecy is measured in terms of *key equivocation* defined as "the amount of information about the key used that is revealed by the ciphertext observed". In other words, key equivocation is the conditional entropy of K given C , as shown in equation (1) [9].

$$H(K|C) = \sum_{j=1}^{l} \sum_{i=1}^{n} q_j \, p_{ij} \log p_{ij} \tag{1}$$

Where

l is the key size, n is the ciphertext length, q_i and p_{ij} are probability and conditional probability computed as in equations 2 and 3 respectively.

$$q_i = \Pr(C = c_i) \tag{2}$$

$$p_{ij} = \Pr(K = k_i | C = c_i) \tag{3}$$

The secrecy of RC4 with KSA and RC4 with $mKSA$ using variable key size (32, 64,128, and 256) and variable plaintext size (128, 256, 512, and 1024) is measured. The average of the secrecy value for 100 random keys is computed for both algorithms. The obtained results are clarified in Table-1. From the result, one can see that RC4 with $mKSA$ always gains more secrecy RC4 algorithm with the original KSA.

Table 1-Average Secrecy Value for Modified RC4 vs. RC4

Key Size	**Plaintext Size**	**Average Secrecy Value**	
		RC4	**RC4 with $mKSA$**
32	**128**	0.2972	0.7109
	256	0.3006	0.4819
	512	0.2926	0.4179
	1024	0.3018	0.3610
64	**128**	0.7334	1.5277
	256	0.7375	1.0020
	512	0.7423	0.8990
	1024	0.7253	0.8298
128	**128**	1.7315	2.5679
	256	1.7076	1.8891
	512	1.7192	1.8167
	1024	1.7260	1.8088
256	**128**	3.9580	3.9877
	256	3.9572	3.9974
	512	3.9706	4.0082
	1024	3.9698	4.0064

4.2 Randomness Test

All encryption algorithms want their output to be as random and unpredictable as possible. There are multiple ways to measure the randomness of bit stream such as Diehard tests, TestU01 and NIST Statistical Test Suit. In this paper, three tests namely *frequency test, block test* and *run test* in the NIST statistical test Suit are used to measure the randomness of the ciphertext generated from RC4 with KSA and RC4 with mKS.

The frequency test, block test and run test are calculated over 10 random ciphertexts for both encryption algorithms. Then, the average of each test is computed as tabulated in Table 2. In this paper, the significance level, α is set to 0.01 (i.e. $\alpha = 0.01$) . The results show that in each individual test case, the RC4 with $mKSA$ always wins a combination of two randomness tests out of the three randomness tests.

Table 2-Randomness Test for RC4 and R4 with $mKSA$

Plaintext Size	Key Size	Frequency Test		Block Test		Run Test	
		RC4	RC4 with $mKSA$	RC4	RC4 with $mKSA$	RC4	RC4 with $mKSA$
128	**32**	0.32866	**0.42679**	0.39551	**0.499**	0.4877	0.4506
	64	0.40316	**0.44774**	0.43555	**0.4396**	0.3957	0.3881
	128	0.5235	0.4175	0.6415	**0.7844**	0.5133	**0.5899**
	256	0.3179	**0.5465**	0.1357	**0.4352**	0.3912	0.3288
256	**32**	0.4613	**0.7264**	0.4647	0.4186	0.3558	**0.6261**
	64	0.6065	**0.768**	0.3843	0.382	0.5443	**0.6213**
	128	0.7728	0.339	0.0895	**0.4133**	0.7323	**0.793**
	256	0.5396	0.2482	0.0924	**0.1178**	0.8259	**0.9472**
512	**32**	0.3978	0.309225	0.4898	**0.7001**	0.4797	**0.7674**
	64	0.63165	0.5954	0.7221	**0.8497**	0.1841	**0.6296**
	128	0.5901	0.4328	0.2739	**0.6428**	0.5698	**0.6446**
	256	0.4044	**0.5163**	0.8546	0.7007	0.393	**0.5618**
1024	**32**	0.5962	0.3466	0.3483	**0.7012**	0.3953	**0.8614**
	64	0.5383	**0.744**	0.3566	**0.6618**	0.6045	0.392
	128	0.3579	**0.7376**	0.4411	**0.5401**	0.5055	0.4692
	256	0.5856	**0.8172**	0.5509	**0.7275**	0.5795	0.2193

4.3. Encryption Time

The encryption time for both RC4 with KSA and RC4 with $mKSA$ has been measured by calculating the average time for 100 samples. All the tests run under a personal computer with the following configuration: Intel® Core™ i7-3612QM CPU @ 2.10 GHz, 7.87 GB usable RAM and MS windows 7 Professional (64 bit). Table-3 shows the encryption time for both RC4 with KSA and RC4 with $mKSA$ algorithms. The time of encryption is approximately the same for both algorithms, which means that the improvement in security and randomness of the generated ciphertext from the RC4 with $mKSA$ has been achieved without scarifying additional time for encryption.

Table 3-Encryption Time for RC4 and RC4 with $mKSA$

Key Size	Plaintext Size	Encryption Time	
		RC4	RC4 with $mKSA$
32	**128**	11.94433	11.90523
	256	11.37250	11.1380
	512	12.06131	12.46212
	1024	14.75909	14.85684
64	**128**	10.79576	10.90333
	256	11.19665	11.46537
	512	12.25686	12.60875
	1024	14.2304	14.6701
128	**128**	11.41161	11.65113
	256	10.23364	10.64904
	512	10.77119	11.03025
	1024	13.81101	13.64488
256	**128**	12.20332	11.96391
	256	10.54154	10.48295
	512	10.53662	11.04003
	1024	13.59603	14.03585

5. Conclusion

This paper presents a new modified key scheduling algorithm to overcome the weakness of the key scheduling algorithm of the original RC4. The modified algorithm enhances the secrecy of the ciphertext especially when the key size is small and proves to be more random than the original RC4. Furthermore, the time of encryption of both algorithms is comparable.

References

1. Crainicu, B. **2015**. On Invariance Weakness in the KSAm Algorithm. *Procedia Technology, Elsevier*. 19, pp.850–857.
2. P Paul, S. and Preneel, B. **2004**. A New Weakness in the RC4 Keystream Generator and an Approach to Improve the Security of the Cipher. *Fast Software Encryption (FSE)*, pp:–259.
3. Fluhrer, S., Mantin, I. and Shamir, A. **2001**. Weaknesses in the Key Scheduling Algorithm of RC4. *Selected Areas in Cryptography*, 2259, pp:1–24.
4. Klein, A., **2008**, Attacks on the RC4 Stream Cipher, *Designs, Codes and Cryptography*, 48(3), pp: 269-286.
5. Jian, X. and Xiaozhong, P. **2010**. An improved RC4 stream cipher. International Conference on Computer Application and System Modeling, Proceedings, 7 (Iccasm), pp:6–9.
6. D. B. Weerasinghe, T., **2012**. Analysis of a Modified RC4 Algorithm. *International Journal of Computer Applications*, 51(22), pp:12–16.
7. Mousa, A. and Hamad, A. **2006**. Evaluation of the RC4 Algorithm for Data Encryption. *International Journal of Computer Science & Applications*, 3(1), pp:44–56.
8. Sarkar, S., **2014**. Proving empirical key-correlations in RC4. *Information Processing Letters, Elsevier*.114, pp:234–238.
9. Rothe, J. **2005.** *Complexity Theory and Cryptology: An Introduction to Crypto complexity*. Springer, Science & Business Media.