Performance Analysis of RC4 and Some of Its Variants

Suman Das, Hemanta Dey and Ranjan Ghosh

Abstract RC4 is a simple and fast cipher, which has proved itself as robust enough and it is trusted by many organizations. But a number of researchers claimed that RC4 has some weakness and bias in its internal states. To increase its security, some guidelines recommended discarding the first N or $2N$ bytes from the final output stream, where N is generally 256. In this paper, it has been statistically analyzed whether the outputs of the algorithm really acquire more security by discarding more number of initial bytes, like $4N$ or $8N$. The original and modified algorithms are analyzed with NIST Statistical Test Suite and it has been tried to estimate an optimum quantity of output bytes to be discarded.

Keywords Modified RC4 · RC4 security · Stream cipher · Key stream generator · NIST test suite

1 Introduction

RC4 is a simple, efficient, fast and easy-to-implement stream cipher. It contains an initialization routine and a random number generator, where random values are selected from an internal state array and two elements are *swapped* in each step. Based on this table-shuffling principle, RC4 is designed for fast software and hardware implementation and widely used in many protocols, standards and commercial products. RC4 cryptanalysis has been mainly devoted to statistical analysis of the output sequence, or to the initialization weaknesses. RC4 contains a secret array of size N (generally, 256), in which integers (0–$N - 1$) are swapped,

S. Das (✉)
Department of Computer Science and Engineering, University of Calcutta, Kolkata, India
e-mail: aami.suman@gmail.com

H. Dey · R. Ghosh
Institute of Radio Physics and Electronics, University of Calcutta, Kolkata, India
e-mail: hemantadey13@gmail.com

depending upon two index pointers *i* and *j* in a deterministic (for *i*) and pseudo-random (for *j*) way. There are two components of RC4: Key-Scheduling Algorithm (KSA) and Pseudo-Random Generation Algorithm (PRGA). KSA performs initializing the S-Box with random-looking permutation of values and PRGA generates the final key-stream bytes [1].

There are several works on strength and weakness of RC4, which shows that there is significant interest in the cryptographic community for it. It has been argued that there are many biases in the PRGA due to propagation of biases in the KSA. Some researchers argued that if the initial *N* or 2*N* bytes from the key-stream are discarded, then these biases can be minimized, hence the security of RC4 increases.

In this paper, the authors have tried to study how the security of RC4 varies if more and more initial bytes from the key-stream are discarded. Firstly 4*N* (here 1,024) and then 8*N* (here, 2,048) initial bytes are discarded and the outputs are compared with the original algorithm. These variants of RC4 are analyzed statistically, following the guidelines given by NIST (National Institute of Standards and Technology), USA in their Statistical Test Suite, coded by the authors. It is found that discarding as many numbers of bytes as possible does not actually increase the security of RC4, but there is a certain optimum level, which should be maintained to get more secured outputs.

Table A. The RC4 Stream Cipher

KSA

Input: Secret Key *K*

```
for $i = 0, \ldots N-1$
        $S[i] = i;$
next $i$
$j = 0;$
for $i = 0, \ldots, N-1$
        { $j = j + S[i] + K[i]$
          swap($S[i]$, $S[j]$);
        }
next $i$
```

Output: S-Box *S* generated by *K*

PRGA

Input: S-Box *S* – The o/p of KSA

```
$i = 0; j = 0;$
while TRUE
        { $i = i + 1$
          $j = j + S[i]$
          swap($S[i]$, $S[j]$);
          $z = S[S[i] + S[j]];$
        }
```

Output: Random Stream *Z*

2 Motivation

RC4 has gone through tremendous analysis since it has become public. Roos [2] showed some weakness in KSA and identified several classes of weak keys for RC4 with some important technical results. He showed strong correlation between the secret key bytes and the final key-stream generated. He suggested discarding *a number of bytes* from the initial key-stream.

Akgün et al. [3] detected a new bias in the KSA and proposed a new algorithm to retrieve the RC4 key in a faster way. Their framework significantly increases the success rate of key retrieval attack. They showed that KSA leaks information about the secret key if the initial state table is known.

Maitra and Paul [4] revolved the non-uniformity in KSA and proposed for additional layers over the KSA and the PRGA. They named the modified cipher as RC4+, which avoids existing weaknesses of RC4. They presented a three-layer architecture in a scrambling phase after the initialization to remove weaknesses of KSA (KSA+). They also introduced some extra phases to improve the PRGA (PRGA+).

Mironov [5] argued that discarding the initial 12–256 bytes from the output stream of RC4 most likely eliminates the possibility of strong attacks. He estimated the number of bytes to be discarded from the initial key-stream as $2N$ (here, 512) or $3N$ (here, 768) to get a more safe output.

Paul and Preneel [6] described a new statistical weakness in the first two output bytes of RC4 key-stream and presented a new statistical bias in the distribution of the first two output bytes of RC4. They recommended to drop at least the initial $2N$ bytes and argued to introduce more random variables in the PRGA to reduce the correlation between the internal and the external states. They also proposed a new key-stream generator namely RC4A with much less operations per output byte.

Tomasevic and Bojanic [7] used a strategy to favor more promising values that should be assigned to unknown entries in the RC4 table and introduced an abstraction in the form of general conditions about the current state of RC4. They proposed a new technique to improve cryptanalytic attack on RC4, which is based on new information from the tree representation of RC4.

Nawaz et al. [8] introduced a new 32-bit RC4 like faster key-stream generator with a huge internal state, which offers higher resistance against state recovery attacks. This is suitable for high speed software encryption.

Gupta et al. [9] thoroughly studied RC4 designing problem from the view point of throughput. They implemented hardware architecture to generate two key-stream bytes per clock cycle using the idea of loop unrolling and hardware pipelining.

Das et al. [10] eliminated the swap function of KSA by using a mathematical process to fill-up the internal state array of RC4, which has been found giving a better security after statistical analysis.

3 Proposed Modifications to RC4

Roos [1] and others strongly discussed about the weakness of KSA and weak keys in RC4. Roos argued that in KSA, only the line of swap directly affects the state table S while exchanging two elements and hence the previous line $j = j + S[i] + K[i]$ is responsible for calculating the indexes. Here, the variable i is deterministic and j is pseudo-random. Therefore, the swap between two elements may happen once, more than once, or may not happen at all—thus brings a weakness in the KSA. He showed that there is a high probability of about 37 % for an element not to be swapped at all. He proposed to discard some initial bytes, preferably *N* (here, 256), to minimize the effects of these biases.

Mironov [4] used an abstract model to estimate the number of initial bytes that should be dumped from the output stream of RC4. He identified a weakness in the KSA and the PRGA, i.e., the final key-stream of RC4, which appears up to the first 2*N* or 3*N* bytes. He blamed the improper *swap* function as a cause of this bias.

In this paper, the authors tried to identify what is the maximum number of bytes that should be discarded from RC4 key-stream before actual encryption starts. They concluded that the number should be more than the previously estimated ones, but it is not that discarding more and more bytes from the output stream really keeps on increasing the security of RC4. Two sets of data (RC4_1024 and _2048), generated by discarding 4*N* and 8*N* bytes respectively had been analyzed, along with data generated by the original RC4.

Outputs of these variants of RC4 have been tested statistically using the guidance of NIST, by the NIST Statistical Test Suite. For all the algorithms, a same text file has been encrypted 500 times by using 500 same encryption keys, generating 500 ciphertexts for each algorithm, each of which contains at least 1,342,500 bits, as recommended by NIST. The three sets of data are then compared to find out if security varies for these algorithms after the proposed modifications.

4 The NIST Statistical Test Suite

NIST developed a Statistical Test Suite, which is an excellent and exhaustive document consisting of 15 tests developed to test various aspects of randomness in binary sequences produced by cryptographic algorithms [11, 12]. The tests are as follows:

1. *Frequency* (*Monobit*) *Test*: Number of 1's and 0's in a sequence should be approximately the same, i.e., with probability ½.
2. *Frequency Test within a Block*: Whether frequency of 1's in an M-bit block is approximately M/2.
3. *Runs Test*: Whether number of runs of 1's and 0's of various lengths is as expected for a random sequence.
4. *Test for Longest-Run-of-Ones in a Block*: Whether the length of the longest run of 1's within the tested sequence (M-bit blocks) is consistent with the length of the longest run of 1's as expected.
5. *Binary Matrix Rank Test*: Checks for linear dependence among fixed length sub-strings of the sequence, by finding the rank of disjoint sub-matrices of it.
6. *Discrete Fourier Transform Test*: Detects periodic features in the sequence by focusing on the peak heights in the DFT of the sequence.
7. *Non-overlapping Template Matching Test*: Occurrences of a non-periodic pattern in a sequence, using a non-overlapping *m*-bit sliding window.

8. *Overlapping Template Matching Test*: Occurrences of a non-periodic pattern in a sequence, using an overlapping m-bit sliding window.
9. *Maurer's Universal Statistical Test*: Whether or not the sequence can be significantly compressed without loss of information, by focusing on the number of bits between matching patterns.
10. *Linear Complexity Test*: Finds the length of a Linear Feedback Shift Register (LFSR) to generate the sequence—longer LFSRs imply better randomness.
11. *Serial Test*: Determines number of occurrences of the 2^m m-bit overlapping patterns across the entire sequence to find uniformity—every pattern has the same chance of appearing as of others.
12. *Approximate Entropy Test*: Compares the frequency of all possible overlapping blocks of two consecutive/adjacent lengths (m and $m + 1$).
13. *Cumulative Sums Test*: Finds if the cumulative sum of a sequence is too large or small. Focuses on maximal excursion of random walks, which should be near 0.
14. *Random Excursions Test*: Finds if number of visits to a state within a cycle deviates from expected value, calculates the no. of cycles having exactly K visits in a cumulative sum random walk.
15. *Random Excursions Variant Test*: Deviations from the expected visits to various states in the random walk, calculates the number of times that a state is visited in a cumulative sum random walk.

In each test, for a bit sequence, NIST adopted different procedures to calculate the *P-values* (probability values) for different tests from the observed and expected values under the assumption of randomness. The Test Suite has been coded by us and used to study the randomness features of different variants of RC4.

5 Results and Discussions

After analyzing the outputs of the original RC4 and modified ones, using the NIST Statistical Test Suite, as described above, it has been found that though discarding some initial bytes of the key-stream increases the security of RC4, discarding more and more bytes from the outputs do not help to increase the security of RC4—at some point, *the beginning of RC4 ends* [4]. The final analysis and comparison is displayed in Table 1, where the POP (Proportion of Passing) status and Uniformity Distribution of NIST tests for these three algorithms are displayed and compared. The best values of a particular test for each algorithm are shaded (in rows) and then the numbers of shaded cells for each are counted (in columns). The highest count (here, for RC4_1024) gives the best result, which shows that this one has a better security than the other, at least for this particular data-set.

POPs and uniformity distributions generated by RC4_1024 and RC4_2048 for the 15 tests, compared to the expected values [11], are displayed in Tables 2 and 3. Distributions of P-values generated by the algorithms RC4_1024 and RC4_2048 for the 15 tests are displayed in Tables 4 and 5. Here, the interval between 0 and 1 is

Table 1 Comparison of POP status and uniformity distribution generated by the 15 NIST Tests for RC4 and its variants

Test↓	POP status for NIST tests			Uniformity distribution for NIST tests		
	RC4	RC4 1024	RC4 2048	RC4	RC4 1024	RC4 2048
1	0.988000	0.990000	**0.992000**	4.154218^{-01}	**8.831714^{-01}**	7.981391^{-01}
2	**0.992000**	0.988000	0.986000	4.904834^{-01}	1.087909^{-01}	**6.952004^{-01}**
3	0.992000	**0.996000**	**0.996000**	**8.920363^{-01}**	5.181061^{-01}	3.537331^{-01}
4	0.982000	**0.988000**	**0.988000**	5.790211^{-01}	2.343734^{-01}	**5.831447^{-01}**
5	0.984000	0.982000	**0.986000**	2.492839^{-01}	**2.596162^{-01}**	4.788391^{-02}
6	0.980000	0.982000	**0.996000**	**4.170881^{-02}**	1.509358^{-03}	4.901567^{-05}
7	0.990000	**0.996000**	0.995000	**8.272794^{-01}**	4.749856^{-01}	4.446914^{-01}
8	**0.992000**	**0.992000**	0.988000	2.224804^{-01}	**9.673823^{-01}**	9.093595^{-02}
9	0.982000	**0.992000**	0.990000	3.856456^{-02}	3.976884^{-01}	**8.343083^{-01}**
10	**0.992000**	0.988000	0.986000	5.462832^{-01}	7.034170^{-01}	**7.981391^{-01}**
11	0.982000	**0.991000**	0.990000	1.699807^{-01}	**5.707923^{-01}**	1.916867^{-01}
12	**0.992000**	0.990000	0.988000	2.953907^{-01}	**7.981391^{-01}**	6.204653^{-01}
13	0.995000	**0.998000**	0.995000	8.201435^{-01}	**9.705978^{-01}**	5.030520^{-02}
14	0.983500	**0.987500**	0.986000	6.729885^{-02}	**6.204653^{-01}**	6.291943^{-02}
15	0.985889	**0.987667**	0.986111	8.386675^{-02}	**5.328562^{-01}**	1.503405^{-02}
Total:	4	9	5	3	8	4

Table 2 POP status and uniformity distribution generated for RC4_1024

Test↓	Expected POP	Observed POP	Status	Uniformity distribution	Status
1	0.976651	0.990000	Successful	8.831714^{-01}	Uniform
2	0.976651	0.988000	Successful	1.087909^{-01}	Uniform
3	0.976651	0.996000	Successful	5.181061^{-01}	Uniform
4	0.976651	0.988000	Successful	2.343734^{-01}	Uniform
5	0.976651	0.982000	Successful	2.596162^{-01}	Uniform
6	0.976651	0.982000	Successful	1.509358^{-03}	Uniform
7	0.976651	0.996000	Successful	4.749856^{-01}	Uniform
8	0.976651	0.992000	Successful	9.673823^{-01}	Uniform
9	0.976651	0.992000	Successful	3.976884^{-01}	Uniform
10	0.976651	0.988000	Successful	7.034170^{-01}	Uniform
11	0.980561	0.991000	Successful	5.707923^{-01}	Uniform
12	0.976651	0.990000	Successful	7.981391^{-01}	Uniform
13	0.980561	0.998000	Successful	9.705978^{-01}	Uniform
14	0.985280	0.987500	Successful	6.204653^{-01}	Uniform
15	0.986854	0.987667	Successful	5.328562^{-01}	Uniform

Table 3 POP status and uniformity distribution generated for RC4_2048

Test↓	Expected POP	Observed POP	Status	Uniformity distribution	Status
1	0.976651	0.992000	Successful	7.981391^{-01}	Uniform
2	0.976651	0.986000	Successful	6.952004^{-01}	Uniform
3	0.976651	0.996000	Successful	3.537331^{-01}	Uniform
4	0.976651	0.988000	Successful	5.831447^{-01}	Uniform
5	0.976651	0.986000	Successful	4.788391^{-02}	Uniform
6	0.976651	0.996000	Successful	4.901567^{-05}	Non-uniform
7	0.976651	0.995000	Successful	4.446914^{-01}	Uniform
8	0.976651	0.988000	Successful	9.093595^{-02}	Uniform
9	0.976651	0.990000	Successful	8.343083^{-01}	Uniform
10	0.976651	0.986000	Successful	7.981391^{-01}	Uniform
11	0.980561	0.990000	Successful	1.916867^{-01}	Uniform
12	0.976651	0.988000	Successful	6.204653^{-01}	Uniform
13	0.980561	0.995000	Successful	5.030520^{-02}	Uniform
14	0.985280	0.986000	Successful	6.291943^{-02}	Uniform
15	0.986854	0.986111	Successful	1.503405^{-02}	Uniform

Table 4 Distribution of *P*-values generated for RC4_1024

Test↓	1	2	3	4	5	6	7	8	9	10
1	48	45	44	56	50	52	59	46	47	53
2	61	56	39	51	45	44	41	47	67	49
3	51	47	55	51	54	41	43	58	59	41
4	50	60	38	54	48	62	51	54	39	44
5	52	50	51	54	59	47	41	35	62	49
6	48	60	55	26	48	41	62	68	41	51
7	42	44	59	45	57	40	57	53	54	49
8	51	51	59	50	48	43	48	48	51	51
9	41	59	58	38	51	52	53	40	52	54
10	53	47	48	49	58	39	51	44	53	58
11	88	84	97	105	106	95	100	112	102	111
12	41	42	49	53	57	46	51	56	52	53
13	114	84	91	116	93	109	84	82	100	126
14	391	416	424	414	415	393	390	391	402	364
15	902	908	973	894	880	880	907	871	830	955

divided into 10 sub-intervals, and the *P*-values that lie within each sub-interval are counted and displayed. These *P*-values should be uniformly distributed in each sub-interval [11]. Histograms on distribution of *P*-values for two tests (4 and 8) are displayed in Figs. 1a–c and 2a–c respectively.

Table 5 Distribution of *P*-values generated for RC4_2048

Test↓	1	2	3	4	5	6	7	8	9	10
1	45	52	45	58	47	54	53	53	40	53
2	59	49	45	47	54	44	53	41	50	58
3	47	46	58	57	47	47	48	63	50	37
4	53	40	42	57	44	57	51	56	46	54
5	52	50	47	60	60	46	37	45	52	51
6	43	49	50	40	66	35	47	81	37	52
7	44	52	47	49	46	62	39	55	59	47
8	61	38	39	46	56	60	61	47	41	51
9	54	49	55	44	47	61	50	46	49	45
10	56	46	47	44	54	47	54	46	60	46
11	91	83	91	113	104	88	117	99	101	113
12	45	40	46	49	57	53	62	47	49	52
13	90	73	112	83	111	112	104	105	99	111
14	422	372	425	376	445	402	415	385	397	361
15	872	931	972	889	879	876	938	908	856	879

* Horizontal ranges for Table 4: 1: 0.0–0.1, 2: >0.1–0.2, 3: >0.2–0.3, …, 10: >0.9–1

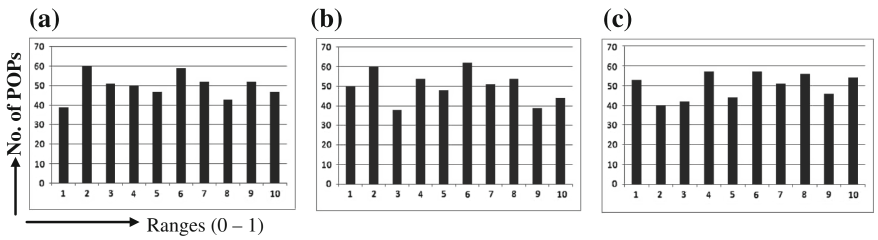

Fig. 1 **a–c** Histograms for *P*-value distribution of test 4 for RC4, RC4_1024 and RC4_2048

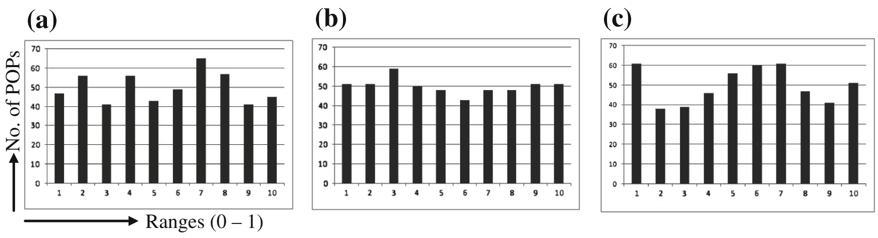

Fig. 2 **a–c** Histograms for *P*-value distribution of test 8 for RC4, RC4_1024 and RC4_2048

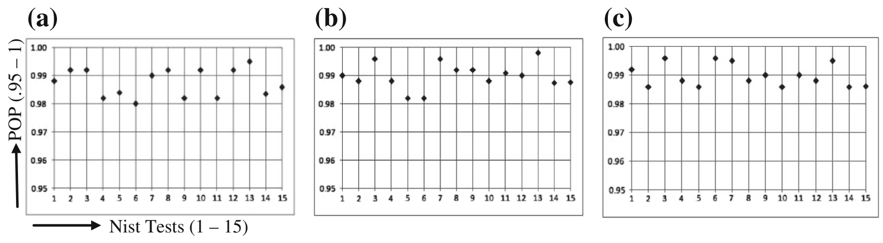

Fig. 3 **a–c** Scattered graphs on POP status on 15 NIST tests for RC4, RC4_1024 and RC4_2048

Scattered Graphs on the POP Status for the 15 tests are displayed in Figs. 3a–c, which examine the proportion of sequences that pass a test. A threshold value (expected POP) has been calculated following the guidance given by NIST. If the proportion falls outside of (i.e., less than) this expected value, then there is evidence that the data is not random [11]. If most of the values are greater than this expected value, then the data is considered to be random. For a particular algorithm, the more number of POPs tend to 1 for the 15 tests, the more random will be the data sequence.

Finally, it has been observed that discarding so many of the initial key-stream bytes does not actually increase the security—saturation occurs after discarding a certain number of bytes. It is clear, though discarding 1,024 bytes is giving a better result than the original RC4, discarding 2,048 bytes is not that satisfactory. Here, in the current data set, after the statistical analysis, the saturation point has been found as 1,024.

6 Conclusion

The RC4_1024 is found to stand in the better merit list comparing to the standard RC4 and RC4_2048. It seems that security in RC4 will be enhanced by discarding a certain number of initial bytes (here, 1,024) from the key-stream. It has been observed that to get more secured key-stream bytes from RC4, an optimum level of discarding the initial bytes from the key-stream should be maintained—discarding as many numbers of bytes as possible does not actually increase its security. Rigorous study is required to find more optimum results in this regard.

References

1. Stinson, D.R.: Cryptography—Theory and Practice. Dept. of Combinatorics & Optimization, University of Waterloo, Ontario (2002)
2. Roos, A.: A Class of Weak Keys in the RC4 Stream Cipher. Post in sci.crypt (1995)

3. Akgün, M., Kavak, P., Demicri, H.: New results on the key scheduling algorithm of RC4. INDOCRYPT, **5365**, 40–52. http://link.springer.com/content/pdf/10.1007/978-3-540-9754-5_4.pdf (2008). (Last accessed on 2 July 2014, Lecture Notes in Computer Science, Springer)
4. Maitra, S., Paul, G.: Analysis of RC4 and proposal of additional layers for better security margin. INDOCRYPT **5365**, 40–52. http://eprint.iacr.org/2008/396.pdf (2008). (Last accessed on 2 July 2014, Lecture Notes in Computer Science, Springer)
5. Mironov, I.: (Not So) Random shuffles of RC4. In: CRYPTO, LNCS 2442, pp. 304–319. California (2002) (Last accessed on 18 Aug 2014)
6. Paul, S., Preneel, B.: A new weakness in the RC4 keystream generator and an approach to improve the security of the cipher. In: FSE 2004, LNCS, vol. 3017, pp. 245–259. Springer, Heidelberg. http://www.iacr.org/archive/fse2004/30170244/30170244.pdf (2004) (Last accessed on 2 July 2014)
7. Tomašević, V., Bojanić, S.: Reducing the state space of RC4 stream cipher. In: Bubak, M. et al. (eds.) ICCS 2004, LNCS 3036, pp. 644–647. Springer, Berlin. http://link.springer.com/chapter/10.1007%2F978-3-540-24685-5_110#page-1 (2004). (Last accessed on 2 July 2014)
8. Nawaz, Y., Gupta, K.C., Gong, G.: A 32-bit RC4-like keystream generator, IACR Eprint archive. http://eprint.iacr.org/2005/175.pdf (2005). (Last accessed on 2 July 2014)
9. Gupta, S.S., Chattopadhyay, A., Sinha, K., Maitra, S. Sinha, B.P.: High-performance hardware implementation for RC4 stream cipher. IEEE Trans. Comput. **82**(4) (2013) (Last accessed on 2 July 2014)
10. Das, S., Dey, H., Ghosh, R.: Comparative study of randomness of RC4 and a modified RC4. Int. Sci. Technol. Congr. IEMCONG-2014, 143–149 (2014)
11. National Institute of Standard & Technology (NIST), Tech. Admin., U.S. Dept. of Commerce, A Stat. Test Suite for RNGs & PRNGs for Cryptographic Applications. http://csrc.nist.gov/publications/nistpubs800/22rec1SP800-22red1.pdf (2010)
12. Kim, S.J., Umeno, K, Hasegawa, A.: Corrections of the NIST Statistical Test Suite for Randomness. Communications Research Lab., Inc. Admin. Agency, Tokyo (2004)