An approach to find out the optimal randomness using parallel s-boxes in RC4

Hemanta Dey

Dept of Master of Computer Application,

Techno India College of Technology,

Kolkata – 700156, India,

hemantadey13@gmail.com

Dr. Uttam Kumar Roy

Dept of Information Technology

Jadavpur University, Salt Lake City,

Kolkata – 700098, India

u_roy@it.jusl.ac.in

Abstract: — *Since last one decade RC4 is one of the best simple, fast and robust stream ciphers trusted by many organizations. It was designed by Ron Rivest in 1987 and kept as a trade secret until it leaked out in 1994. In 1995 Roos [2] find out some weakness and biasness in the internal states of s-boxes in RC4. To resolve all those we contribute some important modifications in RC4 and published Modified RC4 algorithms [12] in different journals. To boost the speed and security of RC4 some researcher modified the original RC4[5] by generating two bits streams in a single clock, using two sets of s-boxes or a single set of s-boxes. S. Paul and B. Preneel [3] contributed some algorithms to improve the weakness in RC4 key stream generator identified by Roos [2] and make it more secure stream cipher.*

In this paper we established two architectures RC42B and RC42D to make optimum randomness using some discarding methods in the layers between KSA and PRGA. In architecture of RC42B we generated two sets of s-boxes using a single key stream in KSA and generated two bits in a single clock in the part of PRGA. We constructed the second set of s-boxes by executing KSA with discarding 1024 bits from the first set of s-boxes. We execute both set of s-boxes within a single cycle and generated two bits one from each in PRGA. In architecture of RC42D we generated second key of size 256 bits by executing RC4 with a key stream of 16 bits. Here we also discarded 1024 bits from each s-box to overcome the weakness and biasness in the internal states. In this architecture two set of s-boxes executed parallel to produce two bits within a single cycle, since two sets are independent. Here we compare our two architectures with other two by S. Maitra and G. Paul[5] namely RC42A, S. Paul and B. Preneel [3] namely RC42C and analyzed the randomness by using NIST Statistical Test Suite. And the comparison of the four results represented that both RC42B and RC42D are always give better results in respect of maximum randomness and improved the security of original RC4.

Keywords: - Modified RC4; Security; Stream Cipher; Key Stream Generator; Statistical NIST Test Suite

I. INTRODUCTION

RC4 is one of the most popular stream cipher in software applications, which was designed by Ron Rivest in 1987. It was kept as a trade secret until it leaked out in 1994. RC4 is most popular efficient, fast, simple and easy-to-implement stream cipher in the field of cryptography. RC4 algorithm has two components namely: Key-Scheduling Algorithm (KSA) and Pseudo-Random Generation Algorithm (PRGA). KSA component base on the s-boxes with a random-looking permutation process where values in s-boxes are shuffled number of times and PRGA component generates the final key-stream bytes. The algorithm of RC4 contain a secret array of size N (256), and in each step, two values of the s-boxes (0 to $N - 1$) swapped by the two index pointers, say i and j, where i is basically a deterministic and j is a pseudo-random pointers. Based on this shuffling process, RC4 becomes one of the strong stream ciphers. So for its acceptance it is implemented in many software and hardware and widely used in many protocols, standards and commercial products. Roos and et al. [1] pointed out some initialization weaknesses of RC4 through some mathematical analysis. They claimed that there are many biases in the output of second component PRGA due to the propagation biasness of KSA. There is significant interest in the cryptographic community from 1994, when the trade secret of RC4 was leaked and as a result several works have been performed on strength and weakness of RC4. Some researchers from cryptographic community argued that if some initial bytes are discarded from the key-stream, then these biases identified by Roos and et al. can be minimized, hence automatically randomness of RC4 increases. We also established some important results regarding the discarding methods and how the security of RC4 varies if more and more initial bytes from the key-stream are discarded [12], we find out that it gives the best results after discarding 4*N* (1024) initial byes from the s-boxes before taking it as PRGA input and comparing all types of discarding using statistical results with the original RC4 algorithm. All variants of RC4 are analyzed statistically through NIST Statistical Test Suite of NIST (National Institute of Standards and Technology), USA. We found out 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 gives the best results in respect to the randomness and returned more secured outputs.

Here we try to establish the enhancement of the security of RC4. The observed existing weakness of original RC4 in the key stream are removed, by using 2 S-Boxes in a single cycle with a key stream, is represented as RC42A. It shows the better result regarding in security aspects with original RC4 and it also executed faster in parallel execution, so total time consumption will be less. We also redesign the modified RC4 to remove the weakness in the key stream identified by S. Paul and B. Preneel to get better result in security aspects in the design RC42D, by removing first 1024 bytes from each S-Box.

RC42A STREAM CIPHER

KSA:

```
Input: Secret Key $K$
for $i = 0, \dots N-1$
$S[i] = i;$
Next $i$
$j = 0;$
for $i = 0, \dots, N-1$
{ $j = j + S[i] + K[i]$
swap($S[i]$, $S[j]$);
}
next $i$
Output: S-Box $S$
```

PRGA:

```
Input: S-Box $S$
$i = 0; j = 0;$
while TRUE
{ $i = i + 1$
$j = j + S[i]$
swap($S[i]$, $S[j]$);
$z = S[S[i] + S[j]];$
}
Output: Random Stream $z$
```

RC42B STREAM CIPHER

KSA:

```
Input: Secret Key $K$
for $i = 0, \dots N-1$
$S1[i] = i;$
Next $i$
$j = 0;$
for $i = 0, \dots, N-1$
{ $j = j + S1[i] + K[i]$
swap($S1[i]$, $S1[j]$);}
Next $i$
for $i = 0, \dots N-1$
$S2[i] = S1[i];$
Next $i$
for $i = 0, \dots, N-1$
{ $j = j + S2[i]$ ;
swap($S2[i]$, $S2[j]$);
}
Next $i$
Output: S-Boxes $S1$ & $S2$
```

PRGA:

```
Input: S-Box $S1$ & $S2$
$i = 0; j = 0;$
while TRUE
{ $i = i + 1$
$j = j + S1[i]$
swap($S1[i]$, $S1[j]$);
$z = S1[S1[i] + S1[j]];$
$j = j + S2[i];$
swap($S2[i]$, $S2[j]$);
$z = S2[S2[i] + S2[j]];$
}
Output: Random Stream $z$
```

RC42C STREAM CIPHER

KSA:

```
Input: Secret Key $K1$
for $i = 0, \ldots N-1$
$S1[i] = i;$
Next $i$
$j = 0;$
for $i = 0, \ldots, N-1$
{ $j = j + S1[i] + K1[i]$
swap($S1[i]$, $S1[j]$);}
Next $i$
for $i = 0, \ldots, N-1$
{ $j = j + S1[i]$ ;
swap($S1[i]$, $S1[j]$);
$K2[i] = S1[S1[i] + S1[j]]$;}
Next $i$
for $i = 0, \ldots N-1$
$S1[i] = i; S2[i] = i;$
Next $i$
$j1 = 0$;  $j2 = 0$;
for $i = 0, \ldots, N-1$
{ $j1 = j1 + S1[i] + K1[i]$
swap($S1[i]$, $S1[j1]$);
$j2 = j2 + S2[i] + K2[i]$
swap($S2[i]$, $S2[j2]$);}
Output: S-Boxes $S1$ & $S2$
```

PRGA:

```
Input: S-Box $S1$ & $S2$
$i = 0; j = 0;$
while $TRUE$
{ $i = i + 1$
$j = j + S1[i]$
swap($S1[i]$, $S1[j]$);
$z = S2[S1[i] + S1[j]]$;
$j = j + S2[i]$;
swap($S2[i]$, $S2[j]$);
$z = S1[S2[i] + S2[j]]$;
}
Output: Random Stream z
```

RC42D STREAM CIPHER

KSA:

```
Input: Secret Key $K1$
for $i = 0, \ldots N-1$
$S1[i] = i;$
Next $i$
$j = 0;$
for $i = 0, \ldots, N-1$
{ $j = j + S1[i] + K1[i]$
swap($S1[i]$, $S1[j]$);}
Next $i$
for $i = 0, \ldots, N-1$
{ $j = j + S1[i]$ ;
swap($S1[i]$, $S1[j]$);
$K2[i] = S1[S1[i] + S1[j]]$;}
Next $i$
for $i = 0, \ldots N-1$
$S1[i] = i; S2[i] = i;$
Next $i$
$j1 = 0$;  $j2 = 0$;
for $i = 0, \ldots, N-1$
{ $j1 = j1 + S1[i] + K1[i]$
swap($S1[i]$, $S1[j1]$);
$j2 = j2 + S2[i] + K2[i]$
swap($S2[i]$, $S2[j2]$);}
for $i = 0, \ldots, 1024$
{ $j = j + S1[i]$ ;
swap($S1[i]$, $S1[j]$); }
Next $i$
for $i = 0, \ldots, 2048$
{ $j = j + S2[i]$ ;
swap($S2[i]$, $S2[j]$); }
Next $i$
Output: S-Boxes $S1$ & $S2$
```

PRGA:

```
Input: S-Box $S$
$i = 0; j = 0;$
while $TRUE$
{ $i = i + 1$
$j = j + S1[i]$
swap($S1[i]$, $S1[j]$);
$z = S1[S1[i] + S1[j]]$;
$j = j + S2[i]$;
swap($S2[i]$, $S2[j]$);
$z = S2[S2[i] + S2[j]]$;
}
Output: Random Stream $z$
```

II. MOTIVATION:

After trade secret of RC4 leaked out in 1994 it has gone through tremendous analysis. Its weakness in KSA is identified with number of classes of weak keys which are recognized by Roos [2] with some important technical results. He showed strong correlation between the initial secret key bytes and the final generated key-stream. Paul and Preneel [3] also presented a new statistical biasness in the distribution of the first two output bytes of RC4. They recommended to discard at least 512 bytes from initial part and argued to introduce more random variables in the PRGA to reduce the correlation between the input and the output bytes. Paul and Preneel [3] also proposed a new key-stream generator, namely RC4A, with much less operations per output byte.

To overcome the biasness on RC4, Maitra and Paul [5] modified its algorithm by introducing a new three-layer architecture between KSA & PRGA. The new algorithm is known as RC4+, which avoids the existing weaknesses of RC4. Its scrambled the bytes more after the initialization phase to remove weaknesses of KSA (KSA+). They also introduced some extra phases to improve PRGA (PRGA+). They revolved the non-uniformity in KSA by using additional layers over the KSA and the PRGA.

Mironov [4] argued that discarding the initial 12 – 256 bytes from the output stream of RC4 most likely eliminates the possibility of strong attacks. He also estimated the number of bytes to be discarded from the initial key-stream as 2*N* (here, 512) or 3*N* (here, 768) to get a more safe output.

Akgün *et al.* [6] showed that if the initial state table is known then KSA output can be leaked information about the secret key which is a new kind of biasness in KSA after Ross detection of biasness. Akgün *et al. also* proposed a new algorithm to retrieve the keys from RC4 in faster way.

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

Sen Gupta *et al.* [8] thoroughly studied the RC4 designing problem and implemented hardware architecture to generate two key-stream bytes per clock cycle, which is much faster than the original RC4.

Nawaz *et al.* [9] introduced a new 32-bit RC4 to implement the concept of high speed software encryption, which has higher resistance against state recovery attacks.

Dey *et al.* [12] eliminated the *swap* function of KSA by using irreducible polynomial concept to fill-up the internal state array of RC4, which removes the initial biases and it has been found that it gives better security.

III. MODIFIED RC4 WITH PARALLEL S-BOXES:

In cryptographic communities RC4 is the popular stream cipher for strong discussions and recommendations about the weakness in RC4 and its improvement. Only few line of simple codes published huge number of research papers in the world. Initially Roos [2] find out that in KSA, the process of exchanging two elements in the s-boxes through swapping process may happen once, more than once, or may not happen at all in some situation which creates the biasness in the output of KSA. Line, $j = j + S[i] + K[i]$, previous to *swap* is responsible for calculating the indexes, where i is as deterministic and j is as pseudo-random. He showed using some bias key sets, in 37% cases elements in s-boxes are not to be swapped at all. To overcome such situation he proposed to discard some initial bytes, preferably 256 bytes to reduce the biasness. In our previous papers, we established the optimum number of bytes that should be discarded from KSA before PRGA encryption process starts. We established that the number should be more than the previously estimated ones, but it is not true that discarding more and more bytes from the KSA output stream really keeps on increasing the randomness of RC4. We established that it gives the best results after discarding 4*N* (1024) initial byes and comparing along all types using statistical results with the original RC4 algorithm.

In this paper we implement 4 types of architecture to design the RC4 algorithms with two S-Boxes among the list design RC42C is proposed by Preenel and et. all and others are compared with this type by selecting it as the benchmark. We established that the modified RC4C2B is better in regarding the security aspects with other 3 architectures. Outputs of these variants of Modified RC4 have been analyzed statistically using the guidance of NIST, Statistical Test Suite. For all the algorithms, a same text file has been encrypted 300 times by using 300 encryption keys, generating 300 cipher-texts for each algorithm, each of which contains at least 1342500 bits, as recommended by NIST. Results generated from these four sets of cipher-texts are then compared to find out if security varies for these algorithms after the proposed modifications.

IV. THE NIST STATISTICAL TEST SUITE

A set of statistical tests for randomness is provided in the package of Statistical Test Suite by National Institute of Standards and Technology (NIST) for detecting deviations of a *binary sequence* from randomness. It is an excellent and exhaustive document consisting of 15 different type tests [10], [11]. The some flavors of the tests are given below:

1) *Frequency (Mono-bit) Test*: This test state that number of 1's and 0's exists in the binary sequence should be approximately in the same ratio, i.e., with probability equal to ½.

2) *Frequency Test within a Block*: This test state that the number of 1's in part of the binary sequence of size M-bit block is near to M/2.

3) *Runs Test*: The focus of this test is to determine the total number of 1's and 0's runs in the binary sequence, where a run is an uninterrupted sequence of identical bits. A run of length *m* consists of exactly *m* identical bits and is bounded before and after with a bit of the opposite value. This test determines whether the oscillation between such zeros and ones is too fast or too slow.

4) *Test for Longest-Run-of-Ones in a Block*: In this test the length of the longest run of 1's within the tested binary sequence of say blocks of M-bit size is check for consistent with the length of the longest run of 1's as expected.

5) *Binary Matrix Rank Test*: This test checks for linear dependence among fixed length sub-strings of the original sequence, by finding the rank of disjoint sub-matrices of it.

6) *Discrete Fourier Transform Test*: This test detects periodic features in the sequence by focusing on the peak heights in the DFT of the sequence that would indicate a deviation from the assumption of randomness. The main intention of this test is to detect the number of peaks which are exceeding the 95 % threshold is significantly different than 5 %.

7) *Non-overlapping Template Matching Test*: In this test checks the occurrences of a non periodic pattern in a sequence, using a non-overlapping *m*-bit sliding window. If the pattern is *not* found, the window slides one bit position. If the pattern is found, then its resume the search and window is reset.

8) *Overlapping Template Matching Test*: This test checks the occurrences of a non-periodic pattern in a sequence, using an overlapping *m*-bit sliding window. In this test when the specific pattern *is* found, the window moves only one bit before resuming the search.

9) *Maurer's Universal Statistical Test*: This test checks whether or not the sequence can be significantly compressed without loss of information, by focusing on the number of bits between matching patterns, since significantly compressed pattern may treated as nonrandom.

10) *Linear Complexity Test*: Its Finds the length of a Linear Feedback Shift Register (LFSR) to generate the sequence, in this context longer LFSRs treat as better randomness.

11) *Serial Test*: Its **f**inds number of occurrences of the 2^m *m*-bit overlapping patterns across the binary sequence where every pattern has the same chance of appearing regarding to as of others.

12) *Approximate Entropy Test*: This test compares the number of occurrences of all possible overlapping blocks of two consecutive / adjacent lengths of size *m* and *m* + 1.

13) *Cumulative Sums Test*: Its finds if the cumulative sum of a sequence which is too large or small. It mainly focuses on maximal excursion of random walks which is near 0.

14) *Random Excursions Test*: This test is basically a series of consecutive eight tests**.** Its finds the number of visits to a state within a cycle which is deviates from the expected value and calculates the number of cycles with exactly K visits in a cumulative sum random walk. The cumulative sum random walk is derived from partial sums after the (0, 1) sequence is transferred to the appropriate (-1, +1) sequence.

15) *Random Excursions Variant Test*: This combined test of the series of eighteen tests. Its finds the deviations from the expected visits to various states in the random walk, and which 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) from the observed and expected results 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.

V. Results and Discussions

Here we analyzing the results of the three modified RC4 with original RC4, using the NIST Statistical Test Suite. As our above discussion, it has been observed that discarding some initial bytes from the s-boxes in after KSA of the key-streams increase randomness in the outputs, hence it enhance the security, but discarding more and more bytes from the outputs do not significantly increase the randomness of RC4 – at some point, *the beginning of RC4 ends* [4]. Here the final comparison is displayed through two tables in Table I and Table II, where the POP (Proportion of Passing) status and Uniformity Distribution of NIST tests for these four algorithms are displayed and compared respectively. In the result best values of a particular test for each algorithm are shaded (in rows) throughout the 15 tests and finally the numbers of shaded cells for each are counted (in columns). The highest number of shaded count gives the better result than the others. In regarding that we conclude the better security of the algorithms than from the other for this particular data-set.

Test No	RC4C2A	RC4C2B	RC4C2C	RC4C2D
1	0.6371194	0.1223252	0.8092489	0.8480268
2	0.8104703	0.4484243	0.1237553	0.2133093
3	0.9319517	0.9414125	0.9087602	0.6620908
4	0.7918798	0.1855552	0.3282969	0.1357197
5	0.5886519	0.5961047	0.1311222	0.2201592
6	0.0017566	0.0245212	0.3345382	0.5101528
7	0.7855618	0.1835471	0.5381822	0.3314077
8	0.2896675	0.2248206	0.5996926	0.07665814
9	0.5817695	0.0229146	0.9421983	0.9442743
10	0.8343083	0.1062459	0.9947196	0.2417409
11	0.8240759	0.8816625	0.3537331	0.08716150
12	0.6163052	0.6625912	0.4559373	0.1756906
13	0.5009345	0.1068773	0.8129051	0.02485492
14	0.4265753	0.5620800	0.3175654	0.4007594
15	0.00361307	0.0005595	0.2137403	0.0003222618
Total:	3	5	4	3

TABLE I. COMPARISON OF POP STATUS GENERATED BY 15 NIST TESTS

Test No	RC4C2A	RC4C2B	RC4C2C	RC4C2D
1	0.993333	0.990000	0.984000	0.994000
2	0.996667	0.986000	0.994000	0.994000
3	0.993333	0.990000	0.990000	0.984000
4	1.000000	0.994000	0.982000	0.994000
5	0.986667	0.974000	0.992000	0.988000
6	0.983333	0.984000	0.986000	0.978000
7	0.990000	0.992000	0.984000	0.990000
8	0.986667	0.990000	0.984000	0.994000
9	0.983333	0.980000	0.984000	0.994000
10	1.000000	0.984000	0.994000	0.986000
11	0.991667	0.995000	0.990000	0.994000
12	0.986667	0.996000	0.986000	0.996000
13	0.995000	0.988000	0.989000	0.989000
14	0.987500	0.987250	0.985500	0.989250
15	0.988519	0.985444	0.983444	0.991556
Total:	5	3	2	6

TABLE II. COMPARISON OF UNIFORM DISTRIBUTION STATUS GENERATED BY 15 NIST TESTS

VI. CONCLUSION

Finally, it has been observed that discarding too many of the initial key-stream in between KSA and PRGA from the S-Boxes with 1024 bytes increase the security in respect to the others – yet we previously established saturation occurs after discarding a certain number of bytes from initial stream. It is clear from the NIST result, discarding 1024 bytes is giving better results than the original RC4 with two s-boxes. Here, in the current data set, after the statistical analysis, the optimal point of discarding in 2 S-Boxes has been found as 4*N*, i.e., here, 1024.

VII. REFERENCES

[1] D. R. Stinson, “Cryptography – Theory and Practice”, 2002, Dept. of Combinatorics & Optimization Univ. of Waterloo, Canada.

[2] A. Roos, “A Class of Weak Keys in the RC4 Stream Cipher”. Post in sci.crypt, 1995.

[3] S. Paul and B. Preneel, “A New Weakness in the RC4 Keystream Generator and an Approach to Improve the Security of the Cipher”, FSE 2004, LNCS, vol. 3017, pp.245-259, Springer, Heidelberg (2004).

[4] I. Mironov, “(Not So) Random Shuffles of RC4”, CRYPTO, LNCS 2442, California, 2002, pp.304–319.

[5] S. Maitra and G. Paul, “Analysis of RC4 and Proposal of Additional Layers for Better Security Margin”, INDOCRYPT, vol.5365, pp.40- 52, 2008, Lecture Notes in Computer Science, Springer.

[6] M. Akgün, P. Kavak and H. Demicri, “New Results on the Key Scheduling Algorithm of RC4”, INDOCRYPT, vol.5365, pp.40-52, 2008, Lecture Notes in Computer Science, Springer.

[7] V. Tomašević and S. Bojanić, “Reducing the State Space of RC4 Stream Cipher”, M. Bubak *et. al* (Eds.): ICCS 2004, LNCS vol.3036, pp.644-647, 2004, Springer-Verlag Berlin Heidelberg.

[8] S. Sen Gupta, A. Chattopadhyay, K. Sinha, S.Maitra and B. P. Sinha,“High-Performance Hardware Implementation for RC4 Stream Cipher”, IEEE Transactions on Computers, vol.82, No. 4, 2013.

[9] Y. Nawaz, K. C. Gupta and G. Gong, “A 32-bit RC4-like Keystream Generator, IACR Eprint archive”, 2005. eprint.iacr.org/2005/175.pdf.

[10] National Institute of Standards & Technology (NIST), Technology Administration, U.S. Dept. of Commerce, “A Statistical Test Suite for RNGs and PRNGs for Cryptographic Applications”, [11] S. J. Kim, K. Umeno and A. Hasegawa, “Corrections of the NIST Statistical Test Suite for Randomness”, Communication Research Lab., Incorporated Administrative Agency, Tokyo, Japan.

[12] S. Das, H. Dey and R. Ghosh, “Comparative Study of Randomness of RC4 and a Modified RC4”, International Science & Technology Congress, IEMCONG-2014, pp.143 -149, Kolkata, India, 2014.