Statistical weaknesses in 20 RC-4 like algorithms and (probably) the simplest algorithm free from these weaknesses - VMPC-R

Bartosz Zoltak

www.vmpcfunction.com
bzoltak@vmpcfunction.com

Abstract. We find statistical weaknesses in 20 RC-4 like algorithms including the original RC4, RC4A, PC-RC4 and others. This is achieved using a simple statistical test. We found only one algorithm which was able to pass the test - VMPC-R. This algorithm, being approximately three times more complex then RC4, is probably the simplest RC4-like cipher capable of producing pseudo-random output.

Keywords: PRNG; CSPRNG; RC4; VMPC-R; stream cipher; distinguishing attack

1 Introduction

RC4 is a popular and one of the simplest symmetric encryption algorithms. It is also probably one of the easiest to modify. There is a bunch of RC4 modifications out there. But is improving the old RC4 really as simple as it appears to be? To state the contrary we confront 20 candidates (including the original RC4) against a simple statistical test. In doing so we try to support four theses:
1. Modifying RC4 is easy to do but it is hard to do well.
2. Statistical weaknesses in almost all RC4-like algorithms can be easily found using a simple test discussed in this paper.
3. The only RC4-like algorithm which was able to pass the test was VMPC-R. It is approximately three times more complex than RC4. We believe that any algorithm simpler than VMPC-R would most likely fail the test.
4. Before introducing a new RC4-like algorithm it might be a good idea to run the statistical test described in this paper.

2 The distant-equalities statistical test

The test measures the numbers of occurrences of 8 different events in the algorithm's output. Let z_i denote the i-th output word of the algorithm:

$$\text{Event } k\text{: } z_i = z_{i+k} \quad \text{for } k \in \{1, 2, 3, ..., 8\}$$

In a random keystream each event would occur with probability $p = 1/N$, where N is the algorithm's word size. The expected number of occurrences of each event in n samples ($E = np$) and the standard deviation ($\sigma = \sqrt{np(1-p)}$) are given by the binomial distribution. The test counts the numbers of occurrences of Events 1,...,8 and compares them with the model $E = np$ using σ as the measure of deviation (the test calculates $d = (f - E)/\sigma$, where f is the measured number), which is a common statistical practice. For large samples the binomial distribution can be approximated with the normal distribution, where e.g. the probability of exceeding the expected value by 3σ is $2^{-8.5}$, by 5σ: $2^{-20.7}$, by 7σ: $2^{-38.5}$.

3 The tested algorithms and their results

All the tested algorithms operate on a subset of variables defined in Table 1. All the algorithms were tested using a KSA which ensures uniform distribution of the values of the internal state. This allowed to test the algorithms in optimal conditions. Possible flaws of the specific Key Scheduling Algorithms dedicated of the analysed ciphers were not analysed here. Instead we concentrated only on the behaviour of the PRNGs operating on properly initialized internal states.
We took advantage of the fact that RC4-like ciphers can be easily scaled down into smaller word sizes. Analysing a scaled down variant magnifies any non-random behaviour the algorithm carries in its design. The approach to analyse RC4 operating on smaller word sizes was applied e.g. in [14], [15], [16].

Table 1. Variables

N : word size
S, S_1, S_2 : N-element permutations of integers $\{0, 1, ..., N-1\}$
i, j, j_1, j_2 : integer variables
+ denotes addition modulo N

3.1 PC-RC4

PC-RC4 was proposed in 2012 by Pardeep, Pushpendra Kumar Pateriya in International Journal of Computer Science and Network [4].

Table 2. PC-RC4 algorithm

repeat steps 1-4:
1. $i = i + 1$
2. $j = j + S[i] + S[j]$
3. swap $S[i]$ with $S[j]$
4. output $S[S[i] + S[j]]$

Table 3. PC-RC4 test results

N	Samples	Event 1	Event 2	Event 3	Event 4	Event 5	Event 6	Event 7	Event 8	Cycle
8	72	+1.75	+3.06	+0.44	+1.75	+3.49	+2.18	+1.75	+2.18	72
8	2^{13}	−25.60	−25.54	+0.07	−25.57	+102.29	−38.34	−25.54	+0.01	No
16	112	+2.69	−2.05	+0.54	−1.62	+3.55	+3.12	−0.32	−0.32	112
16	2^{17}	+269	+11.83	+105	−58.53	+11.79	−23.32	+23.45	+0.04	No
32	480	−3.32	−3.85	+3.92	−3.32	+3.12	−3.58	+1.78	−1.71	480
32	2^{20}	+35.97	−179	+71.94	−95.91	−23.97	−59.94	−35.97	−59.90	No
64	1984	+0.23	−4.68	+10.41	−4.86	+0.23	−5.04	+5.50	−3.23	1984
64	2^{20}	−4.06	−114	+4.19	−102	+130	−114	+20.40	−52.96	No
128	8064	+16.32	−7.65	+8.29	−7.39	+8.67	−7.27	+1.02	−5.23	8064
128	2^{20}	−455	−85.23	−85.27	−85.25	+92.65	−80.97	+89.74	−69.34	No
256	32512	+21.57	−10.56	+0.18	−10.56	+32.58	−9.73	+11.20	−10.29	32512
256	2^{20}	−253	−62.78	+1.15	−62.64	+191	−60.12	+127	−61.23	No

Table 3 presents by how many standard deviations the measured numbers of Events 1,...,8 were different from their expected values ($d = (f - E)/\sigma$) for different word sizes (N=8 operates on 3-bit words, N=16 - 4-bit, ..., N=256 represents an 8-bit implementation). As noted before we are hoping to get absolute values of the deviations lower than 3σ. For example obtaining a deviation of 7σ should happen with probability $2^{-38.5}$ in a truly random source.

The PC-RC4 is a complete disaster in the test even for the full N=256 (8-bit) word size. One problem is that the probability of falling into an extremely short cycle is approximately 50% (this is a rough estimate). The regularly observed cycle lengths for N=8,16,32,64,128,256 are 72, 112, 480, 1984, 8064, 32512 respectively. This is a complete fail and disqualifies the cipher from any practical use.

A nail in the coffin is the behaviour outside of these short cycles. Deviations of over 100σ (!!!) for several Events for any tested word sizes including N=256 with sample sizes not greater than 2^{20} is an absolutely ridiculous result. The scale of the deviations suggest that they should be detected by ANY reasonable statistical test. Instead we read in the PC-RC4 paper [4] that (original quote):

"In PRGA, also increases the randomness on generating the j value that uses as index location pointer for another statement inside the algorithm. This does by using the statement s[j] in j=j+s[i]+s[j]"

This is exactly the opposite to what really happens - changing $j = j + S[i]$ (as in the original RC4) into $j = j + S[i] + S[j]$ (PC-RC4) completely ruins the cipher and yields an algorithm which cannot even be called a flawed PRNG, it simply does not even fit in the PRNG category! In other words the PC-RC4 algorithm should never be used for neither pseudo-random number generation nor for any cryptographic purposes.

This algorithm also shows how easy it is to modify RC4 and even find some justification for the modification while killing all the remains of the security of the cipher. This might also be an example of Bruce Schneier's famous words that building a secure cryptographic system is easy to do badly, and very difficult to do well [8].

3.2 The original RC4

Table 4. RC4 algorithm

repeat steps 1-4:

1. $i = i + 1$
2. $j = j + S[i]$
3. swap $S[i]$ with $S[j]$
4. output $S[S[i] + S[j]]$

Table 5. RC4 test results

N	Samples	Event 1	Event 2	Event 3	Event 4	Event 5	Event 6	Event 7	Event 8
8	2^{20}	$+4.74$	-9.13	$+3.55$	-2.75	$+4.66$	$+0.33$	$+0.61$	-0.77
16	2^{30}	$+27.82$	-47.12	$+4.00$	$+10.52$	$+21.18$	$+6.00$	$+8.31$	$+2.04$
32	2^{32}	$+10.02$	-13.96	$+1.19$	$+7.52$	$+10.33$	$+4.55$	$+5.69$	$+4.20$

The sample size of the N=8 test was limited by the cycle of length 955,496 which we encountered. The N=16 and N=32 tests did not find the ends of their cycles.

In the N=8 test RC4 needed only about 100,000 outputs to exceed -3σ in Event 2 ($z_i = z_{i+2}$). In the full sample of 2^{20} outputs the Event 2 deviation grew to over -9σ. Deviations in Event 1 ($z_i = z_{i+1}$) of 4.74σ, Event 3 ($z_i = z_{i+3}$) of 3.55σ and Event 5 ($z_i = z_{i+5}$) of 4.66σ were also beyond acceptance.

Greater sample size of 2^{30} for N=16 revealed extreme deviations in all but the last Event with peaks of 27.82σ in Event 1 and -47.12σ in Event 2.

The N=32 test was less strict with the sample size increased only 4-fold while the size of the cipher's internal state for N=32 was 2^{75} times larger than for N=16. Despite that the test still found huge deviations of 10.02σ in Event 1, -13.96σ in Event 2 among other unacceptably high ones.

While there is no guarantee that the same deviations would hold for other word sizes it is apparent that the generic construction of RC4 shows significant flaws in the test. We would rather expect proper pseudo-random behaviour regardless of the selected word size. The tests also show quite some similarities in the directions of the excessive deviations for different word sizes which to some extent supports the suspicion that similar deviations could be observed for other word sizes.

3.3 RC4A

RC4A was proposed at FSE 2004 by Souradyuti Paul and Bart Preneel [3].

Table 6. RC4A algorithm

repeat steps 1-7:

1. $i = i + 1$
2. $j_1 = j_1 + S_1[i]$
3. swap $S_1[i]$ with $S_1[j_1]$
4. output $S_2[S_1[i] + S_1[j_1]]$
5. $j_2 = j_2 + S_2[i]$
6. swap $S_2[i]$ with $S_2[j_2]$
7. output $S_1[S_2[i] + S_2[j_2]]$

Table 7. RC4A test results

N	Samples	Event 1	Event 2	Event 3	Event 4	Event 5	Event 6	Event 7	Event 8
8	2^{20}	-0.71	$+5.32$	-0.66	-8.81	$+1.16$	$+0.23$	$+3.08$	-0.01
16	2^{30}	$+1.13$	$+29.53$	$+1.48$	-38.38	$+1.68$	$+5.13$	$+0.87$	$+13.03$
32	2^{32}	-0.11	$+10.61$	$+0.31$	-12.09	$+1.53$	$+2.49$	$+0.79$	$+7.38$

To get comparable results we chose the same sample sizes as for the original RC4. RC4A despite being considerably more complex (it employs two permutations) shows little improvement over RC4 in the test. it does not reach the end of the cycle even for N=8 but this is expected due to the $N \times N!$ times larger internal state. Events 2 ($z_i = z_{i+2}$) and 4 ($z_i = z_{i+4}$) show greatest deviations of over 5σ, 29σ, 10σ and -8σ, -38σ, -12σ respectively.

As in RC4 we can see similarities in the behaviour of the deviations for the tested word sizes which can raise concerns that analogous deviations could be observed for any word size. As noted before - a proper pseudo-random generator (which as RC4 does not utilize a specific word size in its algorithm e.g. in bitwise rotations) should produce quality output regardless of the selected word size.

A conclusion from this test is that RC4A offers little improvement in the statistical properties of the generated keystream over RC4 in this test.

3.4 "Modified RC4"

"Modified RC4" was proposed in 2012 by T.D.B Weerasinghe in International Journal of Computer Applications [6].

Table 8. "Modified RC4" algorithm

repeat steps 1-4:
1. $i = i + 1$
2. $j = j + S[i]$
3. swap $S[i]$ with $S[j]$
4. output $S[S[i] + S[j]]$ xor j

Table 9. "Modified RC4" test results

N	Samples	Event 1	Event 2	Event 3	Event 4	Event 5	Event 6	Event 7	Event 8
8	2^{20}	−45.55	−19.56	−13.93	−7.07	−5.26	−1.60	+0.93	+2.39
16	2^{30}	−507	−375	−292	−204	−155	−106	−80	−53
32	2^{32}	−359	−312	−275	−238	−209	−179	−156	−134
64	2^{32}	−129	−116	−112	−105	−99	−89	−85	−77
128	2^{32}	−45	−43	−42	−40	−39	−38	−37	−35
256	2^{32}	−16.44	−16.90	−15.58	−13.34	−13.20	−15.37	−15.28	−16.28

This algorithm is another example of how easy it is to completely ruin the security of a cipher with a single modification. The sole modification of RC4 comes here in Step 4 in the xor j operation. The author of the new algorithm claims in [6] that (original quote):

"The results show that the modified algorithm is better than the original RC4 in the aspects of secrecy and performance."

A sideline question might be - how can adding an operation to a given algorithm increase its performance?

The sample size for N=8 was limited to 955,496 outputs which was the cycle length we encountered (identical as in the original RC4 which is clear as the state-transformation functions are the same).

The test results speak for themselves. This algorithm produces output of terrible quality. Tests for sample sizes not greater than 2^{32} outputs show catastrophic deviations of over 100σ and even the test for N=256 word size shows extreme deviations of over 15σ.

This algorithm should be avoided by any means!

3.5 "Effective RC4"

"Effective RC4" was proposed in 2013 by T.D.B Weerasinghe at IEEE International Conference on Industrial and Information Systems [7].

Table 10. "Effective RC4" algorithm - original definition with an undefined j variable

repeat steps 1-9:
1. $i = i + 1$
2. $j_1 = j_1 + S_1[i]$
3. swap $S_1[i]$ with $S_1[j]$
4. $j_2 = j_2 + S_2[i]$
5. swap $S_2[i]$ with $S_2[j]$
6. output $S_1[S_1[i] + S_1[j]]$ xor j_1
7. output $S_2[S_2[i] + S_2[j]]$ xor j_2
8. swap $S_1[S_2[j_1]]$ with $S_1[S_2[j_2]]$
9. swap $S_2[S_1[j_1]]$ with $S_2[S_1[j_2]]$

The definition of the algorithm we found used an undefined j variable in steps 3,5,6,7. We believe this was a typing error. We therefore analysed four possible variants of the algorithm and we used the defined variables j_1 and j_2 instead of the undefined j in four different combinations. The side-effect of this situation is that we analysed four RC4-variants instead of only one.

Table 11. "Effective RC4" algorithm - variant 1

repeat steps 1-9:
1. $i = i + 1$
2. $j_1 = j_1 + S_1[i]$
3. swap $S_1[i]$ with $S_1[j_1]$
4. $j_2 = j_2 + S_2[i]$
5. swap $S_2[i]$ with $S_2[j_2]$
6. output $S_1[S_1[i] + S_1[j_1]]$ xor j_1
7. output $S_2[S_2[i] + S_2[j_2]]$ xor j_2
8. swap $S_1[S_2[j_1]]$ with $S_1[S_2[j_2]]$
9. swap $S_2[S_1[j_1]]$ with $S_2[S_1[j_2]]$

Table 12. "Effective RC4" - variant 1 test results

N	Samples	Event 1	Event 2	Event 3	Event 4	Event 5	Event 6	Event 7	Event 8
16	2^{30}	$+1.87$	-336.93	$+0.62$	-164.69	-0.96	-86.65	$+0.64$	-39.31

This version generates extreme deviations in Event 2 (-336σ); Event 4 (-164σ); Event 6 (-86σ); Event 8 (-39σ). These are extremely bad results.

Table 13. "Effective RC4" algorithm - variant 2

repeat steps 1-9:

1. $i = i + 1$
2. $j_1 = j_1 + S_1[i]$
3. swap $S_1[i]$ with $S_1[j_1]$
4. $j_2 = j_2 + S_2[i]$
5. swap $S_2[i]$ with $S_2[j_2]$
6. output $S_1[S_1[i] + S_1[j_2]]$ xor j_1
7. output $S_2[S_2[i] + S_2[j_1]]$ xor j_2
8. swap $S_1[S_2[j_1]]$ with $S_1[S_2[j_2]]$
9. swap $S_2[S_1[j_1]]$ with $S_2[S_1[j_2]]$

Table 14. "Effective RC4" - variant 2 test results

N	Samples	Event 1	Event 2	Event 3	Event 4	Event 5	Event 6	Event 7	Event 8
16	2^{35}	$+7.20$	-1.27	$+1.32$	$+0.53$	-0.55	$+0.23$	-1.22	$+0.73$

After swapping j_1 and j_2 in steps 6 and 7 the results improved significantly. They were however still far from correct with an over 7σ deviation in Event 1 in a sample of 2^{35} outputs for N=16.

Table 15. "Effective RC4" algorithm - variant 3

repeat steps 1-9:

1. $i = i + 1$
2. $j_1 = j_1 + S_1[i]$
3. swap $S_1[i]$ with $S_1[j_2]$
4. $j_2 = j_2 + S_2[i]$
5. swap $S_2[i]$ with $S_2[j_1]$
6. output $S_1[S_1[i] + S_1[j_2]]$ xor j_1
7. output $S_2[S_2[i] + S_2[j_1]]$ xor j_2
8. swap $S_1[S_2[j_1]]$ with $S_1[S_2[j_2]]$
9. swap $S_2[S_1[j_1]]$ with $S_2[S_1[j_2]]$

Table 16. "Effective RC4" - variant 3 test results

N	Samples	Event 1	Event 2	Event 3	Event 4	Event 5	Event 6	Event 7	Event 8
16	2^{35}	$+6.62$	$+8.81$	-1.64	$+1.87$	$+0.82$	-0.89	$+0.79$	$+1.10$

Swapping j_1 and j_2 in steps 3 and 5 (in relation to variant 2) deteriorated the results to some extent by generating two excessive deviations: 8.81σ in Event 2 and 6.62σ in Event 1.

Table 17. "Effective RC4" algorithm - variant 4

repeat steps 1-9:
1. $i = i + 1$
2. $j_1 = j_1 + S_1[i]$
3. swap $S_1[i]$ with $S_1[j_2]$
4. $j_2 = j_2 + S_2[i]$
5. swap $S_2[i]$ with $S_2[j_1]$
6. output $S_1[S_1[i] + S_1[j_1]]$ xor j_1
7. output $S_2[S_2[i] + S_2[j_2]]$ xor j_2
8. swap $S_1[S_2[j_1]]$ with $S_1[S_2[j_2]]$
9. swap $S_2[S_1[j_1]]$ with $S_2[S_1[j_2]]$

Table 18. "Effective RC4" - variant 4 test results

N	Samples	Event 1	Event 2	Event 3	Event 4	Event 5	Event 6	Event 7	Event 8
16	2^{30}	$+0.89$	-38.73	$+1.70$	-16.11	-0.60	-17.26	-1.81	-4.95

Swapping j_1 and j_2 in steps 6 and 7 (in relation to variant 3) made the results even worse with deviations exceeding -38σ in Event 2 and -16σ in Events 4 and 6.

3.6 Another four variants of "Effective RC4"

We also tested another four variants of the "Effective RC4" which we derived from the four variants discussed above but in each one we substituted
6. output $S_1[S_1[i] + S_1[j_x]]$ xor j_1
7. output $S_2[S_2[i] + S_2[j_x]]$ xor j_2

with
6. output $S_2[S_1[i] + S_1[j_x]]$ xor j_1
7. output $S_1[S_2[i] + S_2[j_x]]$ xor j_2

Further in this paper we will refer to these modifications as "type B" variants. They provided no improvement over the four variants of "Effective RC4". In fact the size and direction of the excessive deviations for the corresponding Events for $N = 16$ were similar.

Table 19. "Effective RC4" algorithm - variant 1 type B

repeat steps 1-9:

1. $i = i + 1$
2. $j_1 = j_1 + S_1[i]$
3. swap $S_1[i]$ with $S_1[j_1]$
4. $j_2 = j_2 + S_2[i]$
5. swap $S_2[i]$ with $S_2[j_2]$
6. output $S_2[S_1[i] + S_1[j_1]]$ xor j_1
7. output $S_1[S_2[i] + S_2[j_2]]$ xor j_2
8. swap $S_1[S_2[j_1]]$ with $S_1[S_2[j_2]]$
9. swap $S_2[S_1[j_1]]$ with $S_2[S_1[j_2]]$

Table 20. "Effective RC4" - variant 1 type B test results

N	Samples	Event 1	Event 2	Event 3	Event 4	Event 5	Event 6	Event 7	Event 8
16	2^{30}	$+1.13$	-32.99	-1.49	-180.31	-0.43	-94.54	$+1.79$	-47.43

Variant 1 type B generates extreme deviations in Event 2 (-33σ); Event 4 (-180σ); Event 6 (-94σ) and Event 8 (-47σ). Similarly to those of the original variant 1 these are unacceptably bad results.

Table 21. "Effective RC4" algorithm - variant 2 type B

repeat steps 1-9:

1. $i = i + 1$
2. $j_1 = j_1 + S_1[i]$
3. swap $S_1[i]$ with $S_1[j_1]$
4. $j_2 = j_2 + S_2[i]$
5. swap $S_2[i]$ with $S_2[j_2]$
6. output $S_2[S_1[i] + S_1[j_2]]$ xor j_1
7. output $S_1[S_2[i] + S_2[j_1]]$ xor j_2
8. swap $S_1[S_2[j_1]]$ with $S_1[S_2[j_2]]$
9. swap $S_2[S_1[j_1]]$ with $S_2[S_1[j_2]]$

Table 22. "Effective RC4" - variant 2 type B test results

N	Samples	Event 1	Event 2	Event 3	Event 4	Event 5	Event 6	Event 7	Event 8
16	2^{35}	$+8.70$	0.00	-0.70	$+0.45$	-0.32	$+0.41$	-1.05	$+0.47$

Like in the original variant 1 swapping j_1 and j_2 in steps 6 and 7 yielded a significant improvement over variant 1 type B but still it was not enough to pass the test (over 8.7σ deviation in Event 1 in a sample of 2^{35} outputs for N=16).

Table 23. "Effective RC4" algorithm - variant 3 type B

repeat steps 1-9:

1. $i = i + 1$
2. $j_1 = j_1 + S_1[i]$
3. swap $S_1[i]$ with $S_1[j_2]$
4. $j_2 = j_2 + S_2[i]$
5. swap $S_2[i]$ with $S_2[j_1]$
6. output $S_2[S_1[i] + S_1[j_2]]$ xor j_1
7. output $S_1[S_2[i] + S_2[j_1]]$ xor j_2
8. swap $S_1[S_2[j_1]]$ with $S_1[S_2[j_2]]$
9. swap $S_2[S_1[j_1]]$ with $S_2[S_1[j_2]]$

Table 24. "Effective RC4" - variant 3 type B test results

N	Samples	Event 1	Event 2	Event 3	Event 4	Event 5	Event 6	Event 7	Event 8
16	2^{35}	$+6.07$	-0.06	-3.44	$+0.04$	-1.35	-0.91	-0.87	-1.14

6σ in Event 1 disqualify this candidate and -3.44σ in Event 3 is also a warning sign.

Table 25. "Effective RC4" algorithm - variant 4 type B

repeat steps 1-9:
1. $i = i + 1$
2. $j_1 = j_1 + S_1[i]$
3. swap $S_1[i]$ with $S_1[j_2]$
4. $j_2 = j_2 + S_2[i]$
5. swap $S_2[i]$ with $S_2[j_1]$
6. output $S_2[S_1[i] + S_1[j_1]]$ xor j_1
7. output $S_1[S_2[i] + S_2[j_2]]$ xor j_2
8. swap $S_1[S_2[j_1]]$ with $S_1[S_2[j_2]]$
9. swap $S_2[S_1[j_1]]$ with $S_2[S_1[j_2]]$

Table 26. "Effective RC4" - variant 4 type B test results

N	Samples	Event 1	Event 2	Event 3	Event 4	Event 5	Event 6	Event 7	Event 8
16	2^{30}	$+1.32$	-6.11	$+0.05$	-15.44	-1.02	-8.88	$+0.60$	-5.29

Significant deviations of -6σ in Event 2, -15σ in Event 4, -9σ in Event 6 and -5σ in Event 8 make variant 4 type B fail the test as well.

3.7 "Improved RC4"

"Improved RC4" was proposed in 2010 by J. Xie, X. Pan at International Conference on Computer Application and System Modeling [5].

Table 27. "Improved RC4" algorithm - original definition with an undefined j variable

repeat steps 1-9:
1. $i = i + 1$
2. $j_1 = j_1 + S_1[i]$
3. swap $S_1[i]$ with $S_1[j]$
4. $j_2 = j_2 + S_2[i]$
5. swap $S_2[i]$ with $S_2[j]$
6. output $S_1[S_1[i] + S_1[j]]$
7. output $S_2[S_2[i] + S_2[j]]$
8. swap $S_1[S_2[j_1]]$ with $S_1[S_2[j_2]]$
9. swap $S_2[S_1[j_1]]$ with $S_2[S_1[j_2]]$

We derived the definition of the algorithm from [7] and here we also encountered an undefined j variable in steps 3,5,6,7 and as in case of the "Effective RC4" we analysed four possible combinations of j_1 and j_2 variables replacing the undefined j.

Table 28. "Improved RC4" algorithm - variant 1

repeat steps 1-9:

1. $i = i + 1$
2. $j_1 = j_1 + S_1[i]$
3. swap $S_1[i]$ with $S_1[j_1]$
4. $j_2 = j_2 + S_2[i]$
5. swap $S_2[i]$ with $S_2[j_2]$
6. output $S_1[S_1[i] + S_1[j_1]]$
7. output $S_2[S_2[i] + S_2[j_2]]$
8. swap $S_1[S_2[j_1]]$ with $S_1[S_2[j_2]]$
9. swap $S_2[S_1[j_1]]$ with $S_2[S_1[j_2]]$

Table 29. "Improved RC4" - variant 1 test results

N	Samples	Event 1	Event 2	Event 3	Event 4	Event 5	Event 6	Event 7	Event 8
16	2^{35}	-0.41	$+0.32$	$+0.67$	-212.98	$+0.37$	-27.40	$+0.38$	-30.02

This variant of the algorithm failed a lot showing -213σ in Event 4 accompanied by -27σ in Event 6 and -30σ in Event 8.

Table 30. "Improved RC4" algorithm - variant 2

repeat steps 1-9:

1. $i = i + 1$
2. $j_1 = j_1 + S_1[i]$
3. swap $S_1[i]$ with $S_1[j_1]$
4. $j_2 = j_2 + S_2[i]$
5. swap $S_2[i]$ with $S_2[j_2]$
6. output $S_1[S_1[i] + S_1[j_2]]$
7. output $S_2[S_2[i] + S_2[j_1]]$
8. swap $S_1[S_2[j_1]]$ with $S_1[S_2[j_2]]$
9. swap $S_2[S_1[j_1]]$ with $S_2[S_1[j_2]]$

Table 31. "Improved RC4" - variant 2 test results

N	Samples	Event 1	Event 2	Event 3	Event 4	Event 5	Event 6	Event 7	Event 8
16	2^{35}	-0.91	$+1.14$	-0.21	-11.96	-0.78	-1.31	$+0.93$	$+1.99$

Variant 2 showed significant improvement over variant 1 but still -11.96σ in Event 4 is an unacceptably excessive deviation from the random model.

Table 32. "Improved RC4" algorithm - variant 3

repeat steps 1-9:

1. $i = i + 1$
2. $j_1 = j_1 + S_1[i]$
3. swap $S_1[i]$ with $S_1[j_2]$
4. $j_2 = j_2 + S_2[i]$
5. swap $S_2[i]$ with $S_2[j_1]$
6. output $S_1[S_1[i] + S_1[j_2]]$
7. output $S_2[S_2[i] + S_2[j_1]]$
8. swap $S_1[S_2[j_1]]$ with $S_1[S_2[j_2]]$
9. swap $S_2[S_1[j_1]]$ with $S_2[S_1[j_2]]$

Table 33. "Improved RC4" - variant 3 test results

N	Samples	Event 1	Event 2	Event 3	Event 4	Event 5	Event 6	Event 7	Event 8
16	2^{35}	-0.49	-0.93	$+0.25$	$+16.62$	-2.26	$+2.45$	$+0.13$	$+0.62$

Variant 3 showed some deterioration over variant 2 by producing 16.62σ in Event 4.

Table 34. "Improved RC4" algorithm - variant 4

repeat steps 1-9:
1. $i = i + 1$
2. $j_1 = j_1 + S_1[i]$
3. swap $S_1[i]$ with $S_1[j_2]$
4. $j_2 = j_2 + S_2[i]$
5. swap $S_2[i]$ with $S_2[j_1]$
6. output $S_1[S_1[i] + S_1[j_1]]$
7. output $S_2[S_2[i] + S_2[j_2]]$
8. swap $S_1[S_2[j_1]]$ with $S_1[S_2[j_2]]$
9. swap $S_2[S_1[j_1]]$ with $S_2[S_1[j_2]]$

Table 35. "Improved RC4" - variant 4 test results

N	Samples	Event 1	Event 2	Event 3	Event 4	Event 5	Event 6	Event 7	Event 8
16	2^{35}	-0.62	-1.02	-0.66	-13.68	$+0.35$	-0.11	-0.37	-1.26

Results of variant 4 were similar to those of variant 2 with -13.68σ in Event 4.

3.8 Another four variants of "Improved RC4"

Similarly to as we did with "Effective RC4" we tested four additional variants of "Improved RC4". These were derived in the same manner by substituting
6. output $S_1[S_1[i] + S_1[j_x]]$
7. output $S_2[S_2[i] + S_2[j_x]]$

with
6. output $S_2[S_1[i] + S_1[j_x]]$
7. output $S_1[S_2[i] + S_2[j_x]]$

These four variants (further referred to as type B) produced worse results than the four original variants of "Improved RC4" discussed above. The size and direction of the excessive deviations for the corresponding Events were similar with a few cases of additional extreme results.

Table 36. "Improved RC4" algorithm - variant 1 type B

repeat steps 1-9:
1. $i = i + 1$
2. $j_1 = j_1 + S_1[i]$
3. swap $S_1[i]$ with $S_1[j_1]$
4. $j_2 = j_2 + S_2[i]$
5. swap $S_2[i]$ with $S_2[j_2]$
6. output $S_2[S_1[i] + S_1[j_1]]$
7. output $S_1[S_2[i] + S_2[j_2]]$
8. swap $S_1[S_2[j_1]]$ with $S_1[S_2[j_2]]$
9. swap $S_2[S_1[j_1]]$ with $S_2[S_1[j_2]]$

Table 37. "Improved RC4" - variant 1 type B test results

N	Samples	Event 1	Event 2	Event 3	Event 4	Event 5	Event 6	Event 7	Event 8
16	2^{35}	-0.52	$+1857.47$	-2.55	-177.19	-3.87	-48.22	-0.86	-16.75

Deviations of 1857σ in Event 2 or 177σ in Event 4 speak for themselves.

Table 38. "Improved RC4" algorithm - variant 2 type B

repeat steps 1-9:

1. $i = i + 1$
2. $j_1 = j_1 + S_1[i]$
3. swap $S_1[i]$ with $S_1[j_1]$
4. $j_2 = j_2 + S_2[i]$
5. swap $S_2[i]$ with $S_2[j_2]$
6. output $S_2[S_1[i] + S_1[j_2]]$
7. output $S_1[S_2[i] + S_2[j_1]]$
8. swap $S_1[S_2[j_1]]$ with $S_1[S_2[j_2]]$
9. swap $S_2[S_1[j_1]]$ with $S_2[S_1[j_2]]$

Table 39. "Improved RC4" - variant 2 type B test results

N	Samples	Event 1	Event 2	Event 3	Event 4	Event 5	Event 6	Event 7	Event 8
16	2^{35}	-0.58	$+0.97$	-1.50	-13.35	-1.00	-1.67	-1.75	$+1.40$

Type B variant 2 showed significant improvement over the type B variant 1 (as it was the case of the original variant 1/2 comparison) but -13σ in Event 4 is a clear fail in the test.

Table 40. "Improved RC4" algorithm - variant 3 type B

repeat steps 1-9:

1. $i = i + 1$
2. $j_1 = j_1 + S_1[i]$
3. swap $S_1[i]$ with $S_1[j_2]$
4. $j_2 = j_2 + S_2[i]$
5. swap $S_2[i]$ with $S_2[j_1]$
6. output $S_2[S_1[i] + S_1[j_2]]$
7. output $S_1[S_2[i] + S_2[j_1]]$
8. swap $S_1[S_2[j_1]]$ with $S_1[S_2[j_2]]$
9. swap $S_2[S_1[j_1]]$ with $S_2[S_1[j_2]]$

Table 41. "Improved RC4" - variant 3 type B test results

N	Samples	Event 1	Event 2	Event 3	Event 4	Event 5	Event 6	Event 7	Event 8
16	2^{35}	-0.05	$+1.21$	-1.51	$+17.79$	$+0.55$	$+5.07$	$+1.74$	$+1.09$

Variant 2 type B failed the test by generating 17σ in Event 4 and 5σ in Event 6.

Table 42. "Improved RC4" algorithm - variant 4 type B

repeat steps 1-9:

1. $i = i + 1$
2. $j_1 = j_1 + S_1[i]$
3. swap $S_1[i]$ with $S_1[j_2]$
4. $j_2 = j_2 + S_2[i]$
5. swap $S_2[i]$ with $S_2[j_1]$
6. output $S_2[S_1[i] + S_1[j_1]]$
7. output $S_1[S_2[i] + S_2[j_2]]$
8. swap $S_1[S_2[j_1]]$ with $S_1[S_2[j_2]]$
9. swap $S_2[S_1[j_1]]$ with $S_2[S_1[j_2]]$

Table 43. "Improved RC4" - variant 4 type B test results

N	Samples	Event 1	Event 2	Event 3	Event 4	Event 5	Event 6	Event 7	Event 8
16	2^{35}	-0.69	$+119.09$	-0.39	-3.07	$+0.88$	$+1.23$	-0.37	-1.21

The results of variant 4 type B were extremely bad with 119σ in Event 2.

3.9 Summary of the tests

We tested a total of 20 different modifications of RC4 including the original design. All the algorithms showed significant non-random behaviour in the distant-equalities test. Despite the simplicity of the test (it only measures the frequencies of occurrences of the events that two outputs separated by a given number of ignored outputs are the same) it easily showed that the tested algorithms do not generate these events with the correct $1/N$ probability.

Is it then impossible to find an RC4-like cipher which would pass this simple statistical test?

In 2010-2013 we carried out an experiment to find out. We analysed over 250 different RC4-like algorithms. Out of them we found only one that was able to pass the distant-equalities test. We tested it for different word sizes with considerably greater sample sizes than those needed to compromise the above 20 ciphers (e.g. it was $2^{46.5}$ instead of 2^{30} for $N = 16$) and still did not find any non-random patterns. We described the results, along with the results of several other statistical tests, in [2].

4 The only algorithm passing the distant-equalities test - VMPC-R

VMPC-R was proposed by Bartosz Zoltak in 2013 in Cryptology ePrint Archive Report 2013/768 [2].

VMPC-R is what we believe to be the simplest RC4-like algorithm producing pseudorandom output.

In contrast to RC4, the 3-bit variant of which failed the test with only about 100,000 outputs, the 3-bit variant of VMPC-R passed the test with over 122 trillion ($2^{46.8}$) undistinguishable from random outputs.

Table 44. VMPC-R CSPRNG

N : word size; for practical use $N = 256$ a, b, c, d, e, f, n : integer variables P, S : N-element permutations of integers $\{0, 1, ..., N-1\}$ Let + denote addition modulo N
1. $a = P[a + c + S[n]]$ 2. $b = P[b + a]$ 3. $c = P[c + b]$ 4. $d = S[d + f + P[n]]$ 5. $e = S[e + d]$ 6. $f = S[f + e]$ 7. output $S[S[S[c + d]] + 1]$ 8. swap $P[n]$ with $P[f]$ 9. swap $S[n]$ with $S[a]$ 10. $n = n + 1$ 11. go to step 1

4.1 VMPC-R results

To probe the algorithm's output we used over 100 computers and generated a total of over 2 quadrillion ($10^{15.3} = 2^{51}$) outputs. Table 45 shows the results of the distant-equalities test.

Table 45. VMPC-R distant-equalities test results

N	Samples	Event 1	Event 2	Event 3	Event 4	Event 5	Event 6	Event 7	Event 8
2	504	0.00	−0.03	0.33	−0.03	−0.03	−0.48	0.10	−0.39
3	48,687	−0.28	0.05	0.52	0.47	−1.48	1.49	−0.30	−0.47
4	$2^{22.9}$	−0.10	−0.28	−0.03	−1.27	0.24	2.26	0.38	0.08
5	$2^{28.4}$	−0.69	1.33	−0.29	−1.18	−0.68	1.67	1.28	−1.01
6	$2^{36.7}$	0.72	−0.14	−0.80	−0.74	0.49	0.40	0.09	−0.73
7	$2^{40.8}$	0.27	0.34	−0.19	1.01	−0.62	0.44	2.01	0.46
8	$2^{46.8}$	−1.46	−0.27	−1.25	1.20	1.39	−0.65	0.56	−0.88
16	$2^{46.5}$	0.00	−1.99	0.82	0.46	−1.20	−0.79	−1.75	−1.06
32	$2^{46.7}$	1.64	−0.46	0.55	−1.18	−0.29	−0.31	−1.45	0.46
64	$2^{46.8}$	−0.66	0.25	0.88	−0.77	0.37	0.46	0.51	0.73
128	$2^{47.2}$	0.83	−1.22	0.15	0.33	0.70	−0.24	0.72	−0.07
256	$2^{50.4}$	0.00	−0.24	−0.78	1.34	1.47	−1.23	−0.14	0.24

In the first test we forced the algorithm to work in a single-bit mode ($N = 2$). Even with that extent of simplification the algorithm passed the tests. Here the algorithm's state generated twelve 42 output-long cycles. As an exception from the other word sizes (where the longest cycle usually comprised the majority of the state space) for $N = 2$ we averaged the results for each of the observed 12 cycles. The averages (found in the table) as well as the results for each cycle separately were well within acceptable random deviations. This test comprised 98% of the state space (504 of the 512 possible values of the state).

In the tests for $N \in \{3, 4, 5, 6\}$ we limited the analyses to the size of the longest observed cycle. This was equivalent to examining 62% 82%, 33% and 78% of the complete state space for the respective word sizes. None of the measured probabilities showed any non-random behaviour

in the tests. We also probed several of the remaining shorter cycles (not reported in the table) and found no anomalies there, either.

For $N = 7$ with 1.9 trillion ($2^{40.8}$) outputs tested (about 10% of the state space) we did not encounter a repeated cycle and we observed proper pseudorandom behaviour in the tests.

$N = 8$ (3-bit words) was the first of the big size tests which required significant computational power. It is the smallest size which offers state space large enough ($10^{15.5}$) not to hamper the scale of the test with cycles yet it is small enough to expose possible deviations vividly. We tested over 122 trillion ($2^{46.8}$) outputs and they showed to be undistinguishable from the random model in the tests.

We continued the test with over 100 trillion-word samples for $N \in \{16, 32, 64, 128\}$ and did not find any non-random anomalies in the algorithm's output in the tests.

For $N = 258$ (8-bit) we increased the sample size to over 1.5 quadrillion outputs ($2^{50.4}$) as $N = 256$ is the actual word size for possible practical applications. The results here were also well within the acceptable deviations from the random model.

Summarizing the experiments - we did not manage to find any non-random patterns in over 2 quadrillion outputs of VMPC-R in any of the statistical tests performed for any of the tested word sizes.
Some further analysis of the VMPC-R algorithm including an extended battery of statistical tests, analysis of cycle lengths, resistance against key/internal state recovery attacks, some analysis of its KSA and test-vectors can be found in [2].

Table 46 presents the Key Scheduling Algorithm of VMPC-R.

Table 46. VMPC-R Key Scheduling Algorithm

$N, P, S, a, b, c, d, e, f, n$: as in Table 44 K : key; array of k integers; $k \in \{1, 2, ..., N\}$ V : initialization vector; array of v integers; $v \in \{1, 2, ..., N\}$ $R = \lceil k^2/(6N) \rceil \cdot N$ i : temporary integer variable Let + denote addition modulo N
0. $a = b = c = d = e = f = n = 0$ $P[i] = S[i] = i$ for $i \in \{0, 1, ..., N-1\}$ 1. $KSARound(K, k)$ 2. $KSARound(V, v)$ 3. $KSARound(K, k)$ 4. $n = S[S[S[c + d]] + 1]$ 5. generate N outputs with VMPC-R CSPRNG
Function $KSARound(M, m)$ definition: 6. $i = 0$ 7. repeat steps 8-18 R times: 8. $a = P[a + f + M[i]] + i$; $i = (i + 1) \bmod m$ 9. $b = S[b + a + M[i]] + i$; $i = (i + 1) \bmod m$ 10. $c = P[c + b + M[i]] + i$; $i = (i + 1) \bmod m$ 11. $d = S[d + c + M[i]] + i$; $i = (i + 1) \bmod m$ 12. $e = P[e + d + M[i]] + i$; $i = (i + 1) \bmod m$ 13. $f = S[f + e + M[i]] + i$; $i = (i + 1) \bmod m$ 14. swap $P[n]$ with $P[b]$ 15. swap $S[n]$ with $S[e]$ 16. swap $P[d]$ with $P[f]$ 17. swap $S[a]$ with $S[c]$ 18. $n = n + 1$

5 Conclusions

We applied a simple distant-equalities statistical test to analyse the quality of output generated by 20 RC4-like ciphers. The test revealed strong deviations from the random model in all the analysed designs. We confronted these results with those of the VMPC-R algorithm which was created in a research project aimed at finding the simplest RC4-like cipher capable of passing the distant-equalities test. En route to the final version of VMPC-R over 250 RC4-like variants were analysed, some using a single permutation and some using two permutations along with several integer variables as the internal state. We believe that if there existed a simpler RC4-like algorithm than VMPC-R to pass the distant-equalities test, we would have found it and VMPC-R would be it. We conclude (and in this conclusion we could be wrong) that it is impossible to design an RC4-like algorithm which would be significantly simpler than VMPC-R and which would pass the distant-equalities test with sample sizes not smaller than those we analysed.

References

1. Bartosz Zoltak: VMPC One-Way Function and Stream Cipher Proceedings of FSE 2004, LNCS, vol. 3017, Springer-Verlag, 2004, pages 210-225.
2. Bartosz Zoltak: VMPC-R Cryptographically Secure Pseudo-Random Number Generator Alternative to RC4. Cryptology ePrint Archive: Report 2013/768. http://eprint.iacr.org/2013/768.
3. Souradyuti Paul, Bart Preneel A New Weakness in the RC4 Keystream Generator and an Approach to Improve the Security of the Cipher. Proceedings of FSE 2004, LNCS, vol. 3017, Springer-Verlag, 2004, pages 245-259.
4. Pardeep, Pushpendra Kumar Pateriya: PC-RC4 Algorithm: An Enhancement Over Standard RC4 Algorithm. International Journal of Computer Science and Network (IJCSN) Volume 1, Issue 3, June 2012 www.ijcsn.org ISSN 2277-5420.
5. J. Xie, X. Pan: An Improved RC4 Stream Cipher. 2010 International Conference on Computer Application and System Modeling, (ICCASM 2010), pp. (V7) 156-159, 2010.
6. T.D.B Weerasinghe: Analysis of a Modified RC4 Algorithm. International Journal of Computer Applications (0975 8887) Volume 51 No.22, August 2012. Cryptology ePrint Archive: Report 2014/174. http://eprint.iacr.org/2014/174.
7. T.D.B Weerasinghe: An Effective RC4 Stream Cipher. 8th IEEE International Conference on Industrial and Information Systems (ICIIS), 2013 Cryptology ePrint Archive: Report 2014/171. http://eprint.iacr.org/2014/171.
8. Bruce Schneier: Security Pitfalls in Cryptography https://www.schneier.com/essay-028.html
9. On the (In)security of Stream Ciphers Based on Arrays and Modular Addition Souradyuti Paul and Bart Preneel Proceedings of ASIACRYPT 2006, LNCS, vol. 4284, Springer-Verlag, 2006, pages 69-83.
10. Alexander Maximov: Two Linear Distinguishing Attacks on VMPC and RC4A and Weakness of RC4 Family of Stream Ciphers. Proceedings of FSE 2005, LNCS, vol. 3557, Springer-Verlag, 2005, pages 342-358.
11. Itsik Mantin: Predicting and Distinguishing Attacks on RC4 Keystream Generator. Proceedings of Eurocrypt 2005, LNCS vol. 3494 of LNCS, Springer-Verlag, 2005, pages 491-506
12. Yukiyasu Tsunoo, Teruo Saito, Hiroyasu Kubo, Maki Shigeri, Tomoyasu Suzaki, Takeshi Kawabata: The Most Efficient Distinguishing Attack on VMPC and RC4A ECRYPT Stream Cipher Project, Report 2005 / 037
13. Lars R. Knudsen, Willi Meier, Bart Preneel, Vincent Rijmen, Sven Verdoolaege: Analysis Methods for (Alleged) RC4. Proceedings of ASIACRYPT 1998, LNCS, vol. 1514, Springer-Verlag, 1998.
14. Scott R. Fluhrer, David A. McGrew: Statistical Analysis of the Alleged RC4 Keystream Generator. Proceedings of FSE 2000, LNCS, vol. 1978, Springer-Verlag, 2001.
15. Itsik Mantin, Adi Shamir: A Practical Attack on Broadcast RC4. Proceedings of FSE 2001, LNCS, vol. 2355, Springer-Verlag, 2002.
16. Jovan Dj. Golic: Linear Statistical Weakness of Alleged RC4 Keystream Generator. Proceedings of EUROCRYPT 1997, LNCS, vol. 1233, Springer-Verlag, 1997.