VMPC-R
Cryptographically Secure Pseudo-Random Number Generator Alternative to RC4

Bartosz Zoltak

www.vmpcfunction.com
bzoltak@vmpcfunction.com

Abstract. We present a new Cryptographically Secure Pseudo-Random Number Generator. It uses permutations as its internal state, similarly to the RC4 stream cipher. We describe a statistical test which revealed non-random patterns in a sample of $2^{16.6}$ outputs of a 3-bit RC4. Our new algorithm produced $2^{46.8}$ undistinguishable from random 3-bit outputs in the same test. We probed 2^{51} outputs of the algorithm in different statistical tests with different word sizes and found no way of distinguishing the keystream from a random source. The size of the algorithm's internal state is 2^{3424} (for an 8-bit implementation). The algorithm is cryptographically secure to the extent we were able to analyse it. Its design is simple and easy to implement. We present the generator along with a key scheduling algorithm processing both keys and initialization vectors.

Keywords: PRNG; CSPRNG, RC4; stream cipher; distinguishing attack

1 Introduction

In this paper we analyse Pseudo-Random Number Generators (PRNG) which use permutation(s) as their internal state. The predecessor of this approach, the RC4 algorithm by Ron Rivest, found many software applications, however a significant number of attacks were found against it, many of them distinguishing the produced output from random, e.g. [2], [11], [15], [16], [18].

In this paper we try to find out how complex an RC4-like PRNG would need to be to actually produce pseudorandom output. Our objective is to find a Cryptographically Secure Pseudo-Random Number Generator (CSPRNG) which and can be used both for pseudorandom number generation as such and as a secure stream cipher, where encryption is performed by xoring the consecutive outputs of the algorithm with the consecutive words of plaintext.

2 Design scope

In [1] Paul and Preneel generalized the RC4-like design scope as "stream ciphers based on arrays and modular addition". These algorithms use one or more arrays and several additional variables as their internal state. They usually perform memory access, modular addition and cyclic rotation of bits. Apart from RC4 a significant number of algorithms following this design principle were proposed like IA, IBAA, ISAAC, Py, Py6, PyPy, NGG, GGHN, HC-256, RC4A, VMPC.

In [1] Paul and Preneel describe their successful distinguishing attacks against IA, ISAAC, Py6, NGG and GGHN and point out several risk-factors of the array-based algorithms which can lead to distinguishing attacks and which we used as indicators of what to avoid in the new design:

1. Their general attack framework for array-based ciphers originates in part from the use of large array elements (16 or more bits) along with short array indices (8-bit). This is characteristic of the high speed 16/32 bit ciphers. They also indicate that extremely high performance of such a cipher could be a warning sign that the algorithm may be vulnerable to their attacks.

2. They mount their attacks based on the fact that arrays allow for easy fixing any amount of bits. They investigate the consequences of fixing selected bits to reveal non-random behavior of the cipher's output.

3. In their attacks they exploit the fact that if the cyclic rotation of bits is determined by array elements then fixing some bits of these elements could result in the rotation operation not taking place thus simplifying the round function and opening more attack possibilities.

We chose to cut off any rotation operations and rely only on memory access to the internal permutation(s) and on modular addition. Further in the paper we refer to this design scope as plain-permutation-based (also used in RC4, RC4A and VMPC). Despite the performance compromise (the assumptions actually forced the algorithm to operate on 8-bit words to remain practical) this choice appears to provide some resistance to the above points.

Resistance to 1. In permutations array elements and array indices have the same length (8 bits usually).

Resistance to 2. Fixing bits of a permutation cannot be done as freely because changing one value forces another value to change (we can only swap elements of permutations).

Resistance to 3. With no cyclic rotations of bits there is no threat of neutralizing them.

3 Research procedure

We introduce a statistical test which can easily find non-random patterns in the output of RC4 and which can be applied to any keystream without investigating the details of the PRNG's algorithm. We then used the test to assess the candidates for the new design.

We took advantage of the fact that plain-permutation-based ciphers (lacking the bit rotations) can be scaled down into smaller word-sizes. Analysing a scaled down variant magnifies any non-random behavior the algorithm carries in its design. The approach to analyse RC4 operating on smaller word sizes was applied e.g. in [15], [18], [16].

After investigating over 250 candidate designs we found an algorithm a 3-bit variant of which (operating on integers 0,...,7) was able to produce a stream of $2^{46.8}$ (122 trillion) 3-bit outputs which were undistinguishable from random in our test while a 3-bit variant of RC4 revealed non-random patterns after just about $2^{16.6}$ = 100,000 outputs in the same test.

We tested the final candidate in an extended battery of statistical tests for different word sizes.

We analysed several other aspects of the algorithm's cryptographic security which, to the extent to which we were able to investigate, did not show any weaknesses in the design.

Along with the CSPRNG we propose its Key Scheduling Algorithm (KSA) digesting both keys and initialization vectors. We describe some analysis of the KSA's security and the results of several statistical tests on it.

The algorithm we present in this paper is what we believe is the simplest plain-permutation-based CSPRNG which produces pseudorandom output. It is more complex and slower than RC4 but we will argue that it still remains simple in design and easy to implement.

4 Our universal distant-equalities statistical test

Our objective was to devise a test which does not investigate any algorithm-specific characteristics so that it can be quickly applied to verify any new candidate.

The test was inspired by the approach used by Fluhrer and McGrew in [15] to find anomalies in digraph probabilities in RC4 and by Maximov [4] and [5] who found that two consecutive outputs of VMPC are 0 when $i = 0$ with the biased probability of $2^{-16} \cdot (1 - 2^{-7.98})$. Some inspiration came also from Mantin's analysis of RC4 in [11]. Our 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 algorithm operates on integers $0, ..., N - 1$). 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: $2^{-38.5}$.

4.1 RC4 in the test

Table 1. RC4 algorithm

```
N : word size
S : N-element permutation of integers {0, ..., N − 1}
i, j : integer variables
Let + denote addition modulo N
------------------------------------------------------
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 2. RC4 test results

N	Samples	Event 1	Event 2	Event 3	Event 4	Event 5	Event 6	Event 7	Event 8
16	$2^{29.9}$	+28.18	−44.98	+3.14	+8.95	+18.28	+5.90	+7.77	+0.74

Table 2 presents by how many standard deviations the measured numbers of events 1,...,8 were different from their expected values ($d = (f - E)/\sigma$) in a sample of 10^9 outputs.

We chose $N = 16$ (4-bit words) to present the results. Although for $N = 8$ (3-bit words) the biases were revealed much earlier, tests for $N = 8$ were frequently interrupted by cycles. We can see huge deviations of $+28\sigma$ or -44σ for $N = 16$. For $N = 8$ RC4 needed only about 100,000 outputs to exceed -3σ for event 2 ($z_i = z_{i+2}$).

The objective of the test was not to find the lower bounds for the distinguishers' requirements but to reveal the biases with a universal test and to verify that the test can be used as an assistance in our search for the new design.

5 Searching for the new design

As noted in Section 2 we were searching for a plain-permutation-based algorithm (not using bit rotations). Our objective was to make the algorithm as simple as possible but complex enough to produce pseudorandom output in our tests. Before reaching the final version we examined over 250 candidates which all failed in our distant-equalities test. The variations we tested came from combining the following factors:

1. The number of permutations and variables.
2. The way the variables were updated by the permutation.
3. The way the variables feedbacked each other.
4. The way the output word was computed from the variables and the permutation.
5. The way the variables controlled the swap operations on the permutations.

The purpose of this stage was to find one candidate to carry out the extended testing on. Each version first underwent our test for small word sizes (3-bit) to expose non-random deviations quicker. If the candidate failed the test at this stage it was rejected.

Initially we tested approximately 50 designs which used a single permutation and different numbers of additional variables. Some versions used two swap operations controlled by different combinations of variables. We observed huge gaps separating each of the candidates from the random model in our test and we were finally unable to build a proper keystream generator using a single permutation in our design scope.

We extended our search to algorithms using two permutations. For the sake of simplicity we limited the search space to one swap operation per each permutation per one output. Adding the second permutation significantly raised the complexity of the algorithm and broadened the number of variations.

Adding the second permutation did not make the search an instant success. Instead it significantly slowed it down due to the large number of candidates and the fact that they generally performed better in the test and it took more time to find the deviations. The majority of our two-permutation candidates used 5 variables. From over 160 different combinations of them none could pass the test with significantly over 1 billion outputs.

As the final step we chose the combinations of factors 2 and 3 which appeared to work best and analysed a 7-variable scheme in 36 configurations of different variables taking part in the output computing step (factor 4) and in the two swap operations (factor 5). Out of them 20 quickly failed. Out of the remaining 16 we found a few which showed promising results. From them we chose the one which performed best. In contrast to RC4, the 3-bit variant of which failed the test with only about 100,000 outputs and with the majority of other candidates struggling to pass the 1 billion outputs mark, the 3-bit variant of the final candidate passed the test with over 122 trillion ($2^{46.8}$) undistinguishable from random outputs.

5.1 The final candidate

We named the new algorithm VMPC-R where VMPC stands for the name of the function (summarized in Section 5.7) employed in the output computing step 7 and R stands for random. Specification of the algorithm can be found in Table 3.

Table 3. 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 $\{0, ..., N-1\}$
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
```

5.2 The extended battery of tests

The final candidate underwent a series of additional statistical tests. Apart from our distant-equalities test we measured several other parameters which were commonly exploited in distinguishing attacks against similar ciphers. Let N denote the algorithm's word size, z_i the i-th output word; n is the internal variable of VMPC-R. The battery included the following measurements:

- 8 events in our universal test: $z_i = z_{i+k}$ for $k \in \{1, 2, 3, ..., 8\}$
- N numbers of occurrences of each possible output value z_i
- N^2 numbers of occurrences of each possible pair $[n, z_i]$
- N^2 numbers of occurrences of each possible pair $[z_i, z_{i+1}]$
- N^3 numbers of occurrences of each possible combination $[n, z_i, z_{i+1}]$

5.3 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 4 presents the sample sizes we tested for different word sizes. Table 5 shows the results of the distant-equalities test only. We do not quote the results of the remaining tests from the battery due to their large capacity.

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

Table 4. VMPC-R statistical tests sample sizes

Word size	Number of output words tested
N=2 (1-bit)	$504 = 10^{2.7} = 2^{9.0}$
N=3	$48\ 687 = 10^{4.7} = 2^{15.6}$
N=4 (2-bit)	7.8 million $= 10^{6.9} = 2^{22.9}$
N=5	365.8 million $= 10^{8.6} = 2^{28.4}$
N=6	113.8 billion $= 10^{11.1} = 2^{36.7}$
N=7	1.9 trillion $= 10^{12.3} = 2^{40.8}$
N=8 (3-bit)	122 trillion $= 10^{14.1} = 2^{46.8}$
N=16 (4-bit)	103 trillion $= 10^{14} = 2^{46.5}$
N=32 (5-bit)	114 trillion $= 10^{14.1} = 2^{46.7}$
N=64 (6-bit)	121 trillion $= 10^{14.1} = 2^{46.8}$
N=128 (7-bit)	163 trillion $= 10^{14.2} = 2^{47.2}$
N=256 (8-bit)	1.5 quadrillion $= 10^{15.2} = 2^{50.4}$

Table 5. VMPC-R distant-equalities test results

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

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 behavior 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 in all the tests 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. In the sections to follow we try to investigate some other aspects of the algorithm's cryptographic security.

5.4 Key/state recovery attacks against VMPC-R

Unlike in case of distinguishing attacks, array-based stream ciphers usually show high resistance to key (or internal state) recovery attacks. For some time the fastest state recovery algorithm for RC4 was by Mister and Tavares [13]. At CRYPTO 2008 [6] Maximov proposed an improved attack against RC4 requiring about 2^{241} operations. One difficulty in mounting these attacks against this family of ciphers is the fact that many secret words of the internal state are used to produce a single word of output. In RC4 3 words of the internal permutation are used per one output word. VMPC-R uses 11 words of its permutations to produce one output (4 elements of P and 7 of S). We roughly estimate that the total number of possible values of the unique elements of P and S used to produce 50 VMPC-R outputs would be greater than the total keyspace of a 2048-bit (256-byte) secret key. We don't expect the key/state recovery attacks to be a significant threat to the security of the proposed cipher.

Another area where the algorithm's complexity might strengthen its resistance is attacks derived from fortuitous states introduced by Fluhrer and McGrew. In [15] they presented RC4 states in which only X elements of the permutation are involved in the computation of X successive outputs. They showed how this fact can be used to determine some parts of the permutation with nontrivial probability. Paul and Preneel note in [1] that their attack approach has its origins in the fortuitous states. We believe that VMPC-R using 11 permutation elements per round controlled by the set of 7 interdependent variables would make introducing fortuitous states significantly harder, if possible.

We also expect the level of complexity of the round function to be a considerable obstacle in mounting fault analysis attacks against VMPC-R as opposed to RC4 for which successful fault attacks were published by Hoch and Shamir in 2004 [9] or Biham, Granboulan and Nguyen in 2005 [10].

5.5 Cycle lengths of VMPC-R

We assess the probability of VMPC-R repeating a cycle to be negligibly low for $N = 256$. The size of the internal state is determined by the size of the P and S permutations and the 7 variables a, b, c, d, e, f, n to be $N!^2 \times N^7$. For $N = 256$ it is over 10^{1030} or 2^{3424} possible values. Because the n variable is increased by 1 modulo N in each iteration the internal state would be changed in N iterations before it could repeat. In these iterations P and S would undergo N swap operations indexed by n, a and f while a, b, c, d, e, f would be updated N times. We believe that the probability that the variables would repeat their values after these N iterations should not be significantly different from $N!^{-2} \times N^{-6}$ or 2^{-3416} for $N = 256$. In our statistical tests for $N = 8$ ($2^{51.6}$ possible values of the internal state) we did not encounter a single case of a repeated cycle in the sample of $2^{46.8}$ outputs. In Table 6 we show three longest cycles we observed for $N \in \{3, 4, 5, 6\}$.

Table 6. VMPC-R observed cycle lengths

Word size	Total state space	3 longest cycles
N=3	78 732	48 687; 6 945; 6 126
N=4	9 437 184	7 766 992; 833 100; 369 056
N=5	1 125 000 000	365 826 825; 219 688 515; 155 601 705
N=6	145 118 822 400	113 795 459 358; 10 758 771 978; 9 768 433 476

5.6 First outputs of VMPC-R

Statistical properties of the first outputs of the cipher (generated directly after the key scheduling algorithm) were targets of several distinguishing attacks. E.g. in [16] Mantin and Shamir found that the second output of RC4 is equal 0 with twice the expected probability. In [2] Paul and Preneel discovered a bias in the first two output bytes of the RC4 keystream. [17] and [12] added to those.

Anomalies here can result from the flaws in either the keystream generator or the key setup algorithm. Given the proper statistical behavior of VMPC-R output in our tests and given the security features of the key scheduling algorithm discussed in Section 6 we do not expect the cipher's output to behave differently directly after the key setup than it did in our statistical tests.

We additionally verified that fact by separate tests where only the first 8 outputs of the cipher were analysed using the battery described in Section 5.2. We performed the tests both for the full and for several reduced variants of the key scheduling algorithm, one of them performing only about 30% of the operations of the full KSA. None of the tests showed any statistical anomalies in the first outputs of VMPC-R.

5.7 VMPC one-way function. An additional layer of security

VMPC-R employs the VMPC one-way function in step 7. VMPC stands for Variably Modified Permutation Composition. The function was proposed by Zoltak at FSE 2004 [7]. It naturally fits the design of permutation-based ciphers. The function can be implemented with three elementary *mov* processor instructions per word. It is defined as:

$$Q[x] = P[M_1[P[M_2[P[x]]]]]$$

where M_1 and M_2 are any non-secret permutations such that $M_1[x] \neq M_2[x]$. An example VMPC function is $Q[x] = P[P[P[x]] + 1]$, where + denotes addition modulo permutation size.

The M_i permutations corrupt the cycle structure of P in such a way that deriving any information about P from Q requires guessing significant portions of P (e.g. 34 guessed elements is the estimate for 256-element permutations). Any occurrence of $M_1[x] = M_2[x]$ (against the definition) would weaken the function's one-way properties. A function $Q[x] = P[M_1[P[M_1[P[x]]]]]$

would be trivial to invert for any value of M_1. The original estimate appears to still stand that inverting the function for 256-element permutations requires an average effort of 2^{260}.

In strict terms one-way functions have not been proved to exist. The existence of one would imply $P \neq NP$. The VMPC function with no polynomial-time algorithm for inverting known can be regarded as believed-to-be-one-way.

6 The key scheduling algorithm of VMPC-R

Specification of the algorithm can be found in Table 7.

Table 7. VMPC-R key scheduling algorithm

```
$N, P, S, a, b, c, d, e, f, n$ defined in Table 3
$k$ : length of key; $k \in \{1, ..., N\}$
$K$ : key; array of $k$ integers
$v$ : length of initialization vector; $v \in \{1, ..., N\}$
$V$ : initialization vector; array of $v$ integers
$i$ : 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 $KSARound(Y, y)$ definition:
    6. $i = 0$;   $R = N \cdot \lceil y^2/(6N) \rceil$
    7. repeat steps 8-18 $R$ times:
        8.  $a = P[a + f + Y[i]] + i$;   $i = (i + 1) \bmod y$
        9.  $b = S[b + a + Y[i]] + i$;   $i = (i + 1) \bmod y$
        10. $c = P[c + b + Y[i]] + i$;   $i = (i + 1) \bmod y$
        11. $d = S[d + c + Y[i]] + i$;   $i = (i + 1) \bmod y$
        12. $e = P[e + d + Y[i]] + i$;   $i = (i + 1) \bmod y$
        13. $f = S[f + e + Y[i]] + i$;   $i = (i + 1) \bmod y$
        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. $n = n + 1$
```

6.1 Statistical properties of the KSA

We investigated the statistical properties of the KSA in two areas. One was whether the elements of the generated permutations and the variables follow the uniform distribution. The second was whether permutations and variables generated from different keys are uncorrelated. We performed all the tests on the KSA only for the full $N = 256$ version.

To verify the results in the first area we measured the probabilities of occurrences of each of the possible N values in all the N indices in both permutations (P and S) and in all the 7 variables (a, b, c, d, e, f, n) for different key sizes. All the 132864 measured probabilities stayed well within acceptable random deviations from the expected $1/N$ which is in line with the uniform distribution.

The second objective was to verify whether the slightest change in the input key would cause the proper avalanche effect to hide any possible correlations between the generated permutations and variables. We approached this test according to the strict avalanche criterion (SAC) defined by Webster and Tavares in [20]. The criterion is satisfied when changing a single input bit changes each output bit with probability 0.5. We measured the number of unchanged corresponding elements of the permutations and the number of cases of unchanged variables generated with two different keys.

We tested the consequences of a change of a single byte (this included flipping a single bit) in a pseudorandom key and changing a byte in four different types of non-random keys. We then repeated the tests with the operation of changing the byte substituted by an operation of appending a new byte to the key.

To get the worst-case result we chose several scenarios which appeared to hinder the avalanche effect the most. All the tested keys were of maximum length of $k = 256$ bytes (2048 bits) and we changed the byte of the key which was the last to be input to the KSA. In any other case (smaller keys or changing any other byte of the key) the avalanche effect would be more intensive as the changes of the key would be digested by the algorithm more times or earlier. Additionally we reduced the the number of rounds of the $KSARound$ function from the original $R = 11008$ (for $k = N = 256$) to only $R = 256$. We chose the following pairs of keys for the test:

- K_1: 256 random bytes; $K_2[i] = K_1[i]$, $i \in \{0, ..., 254\}$; $K_2[255] \neq K_1[255]$
- K_1: 255 random bytes; $K_2[i] = K_1[i]$, $i \in \{0, ..., 254\}$; $K_2[255]$=random
- $K_1[i] = i$, $i \in \{0, ..., 255\}$; $K_2[i] = K_1[i]$, $i \in \{0, ..., 254\}$; $K_2[255] \neq K_1[255]$
- $K_1[i] = i$, $i \in \{0, ..., 254\}$; $K_2[i] = K_1[i]$, $i \in \{0, ..., 254\}$; $K_2[255]$=random
- $K_1[i] = 255 - i$, $i \in \{0, ..., 255\}$; $K_2[i] = K_1[i]$, $i \in \{0, ..., 254\}$; $K_2[255] \neq K_1[255]$
- $K_1[i] = 255 - i$, $i \in \{0, ..., 254\}$; $K_2[i] = K_1[i]$, $i \in \{0, ..., 254\}$; $K_2[255]$=random
- $j \in \{0, ..., 255\}$: $K_1[i] = j$, $i \in \{0, ..., 255\}$; $K_2[i] = K_1[i]$, $i \in \{0, ..., 254\}$; $K_2[255] \neq K_1[255]$
- $j \in \{0, ..., 255\}$: $K_1[i] = j$, $i \in \{0, ..., 254\}$; $K_2[i] = K_1[i]$, $i \in \{0, ..., 254\}$; $K_2[255]$=random

To obtain a significant security margin for the measured avalanche effect we performed the tests both for the full version of the KSA and for the reduced version using only steps 1 and 5. In the tests we compared the generated permutations and the 7 variables generated by the KSA for K_1 and K_2 and found that in each test the numbers of unchanged elements in the resulting permutations were well within acceptable random deviations from the expected 1 and that the probability that any of the 7 variables remained unchanged was not significantly deviated from the expected $1/N$.

Step 1 alone almost provided the proper avalanche effect. The result we recorded for step-1-only variant was about 1.3 unchanged corresponding elements of the generated permutations rather than 1 we would expect. For shorter 128-byte (1024-bit) keys the tests for step-1-only variant produced even better results at around 1.05 identical corresponding permutation elements (as expected, the shorter key with the other factors unchanged intensified the avalanche effect).

The KSA limited only to steps 1 and 5 was enough to achieve 1 in all the tests. Steps 2 and 3 approximately double the amount of mixing performed by steps 1 and 5 this way intensifying the avalanche effect.

Given the fact that the tests were successful even for the significantly reduced versions of the KSA we believe that the analysed statistical properties of the algorithm are in line with the random model and the algorithm's avalanche effect is ensured with a significant security margin.

6.2 Some comments on the design

The number of rounds. The *KSARound* function performs $R = N \cdot \lceil y^2/(6N) \rceil$ iterations. This value ensures that each word of a y-word key or initialization vector updates the internal state at least y times. This follows an intuition that the probability of deriving identical internal states from keys differing in only one bit or one word should not be different from the probability $N!^{-2} \times N^{-7}$ of deriving identical internal states from two random keys. For $N = 256$ and key sizes $k \in \{1, 2, ..., 39\}$ (keys up to 312 bits) $R = N$. For $N = 256$ and key sizes $k \in \{40, 41, ..., 55\}$ (keys from 320 to 440 bits) $R = 2N$. And so on.

Step 3 of the algorithm. Apart from intensifying the avalanche effect step 3 provides an additional layer of security against key-recovery attacks.

Let's consider a reduced version without step 3 and assume that a successful state recovery attack has been accomplished. Since all the data steps 2-5 operate on is known by the adversary it would be possible to revert these steps and obtain the value of the internal state after step 1. This would enable to process steps 2-5 for any new message and decrypt it without finding the actual secret key.

In the full KSA this attack would provide the internal state after step 3. However here the adversary would face a hard problem of reverting step 3 which takes the secret key as a parameter. As a result of the avalanche effect any new message encrypted with the same key but different initialization vector would have the internal state uncorrelated with the one the adversary holds which would give the adversary no advantage in decrypting any new message.

Step 5 of the algorithm. A number of approaches against array-based ciphers exploited the internal state right after the key-setup to mount distinguishing attacks. Some were mentioned in Section 5.6. One idea to improve the RC4 KSA was to drop several first outputs of the cipher. Although we believe VMPC-R KSA provides proper statistical behavior, we propose to add this step as an additional layer. A measurable benefit it provides is the intensification of the avalanche effect.

No non-secret values. Our objective was to keep all the state variables secret. Leaving the value of the counter variable (n in our algorithm) known raises unnecessary risk as the adversary knows the indices to the arrays used by some parts of the cipher's round function. In fact several attacks mentioned in Section 5.6 exploit this situation. Although we don't believe this to be a significant security improvement, the effort required to ensure this property (setting variable n to a secret value in step 4) is only symbolic.

No obvious equivalent keys. To make introducing different keys producing the same internal state a harder task we use both the value of the word of the key ($Y[i]$) and its index i as input. Both are mixed non-linearly in the $a = P[a + f + Y[i]] + i$ step 8 (and analogously in steps 9-13).

Flexibility. The KSA's compatibility with a natural in software applications 256-byte array might open more doors to its possible alternative applications. Such arrays containing additional information could be used to influence the internal state. Apart from the key and the initialization vector, the algorithm could accept any amount of parameters like additional or session keys, personal information or hardware ID by calling *KSARound* with these parameters after step 2 and before step 3.

7 Performance and test values

We measured the performance of the CSPRNG and the KSA using a moderately optimized assembler implementation on an Intel Pentium 4 processor clocked at 3.0 GHz. Performance figures on a multi-core processor like Intel i7 (also clocked at 3.0 GHz) were very similar with the algorithms running on a single core. Table 8 shows the results of the CSPRNG (Mbyte refers to one million bytes (8-bit outputs) produced for $N = 256$). Table 9 presents the figures for the full KSA digesting both the key and the IV. Performance results for 8,16,32,64,128 and 256-byte keys (and IVs) are the same. Table 10 contains test output values of the algorithms.

Table 8. VMPC-R CSPRNG performance

Mbytes/s	cycles/byte
54.4	55.1

Table 9. VMPC-R KSA performance

runs/s	cycles/run
28686	104580

Table 10. Test-output of VMPC-R

input:								
K; $k = 9$	{11, 22, 33, 144, 155, 166, 233, 244, 255}							
V; $v = 8$	{255, 250, 200, 150, 100, 50, 5, 1}							
output of KSA:								
P index	0	1	2	3	252	253	254	255
P value	97	218	106	125	139	86	36	126
S index	0	1	2	3	252	253	254	255
S value	152	143	19	154	92	25	24	157
output of CSPRNG:								
output index	0	1	2	3	254	255	256	257
output value	49	161	79	69	85	237	96	243
output index	1000	1001	10000	10001	100000	100001	1000000	1000001
output value	181	184	136	99	67	27	253	231

Table 11. Test-output of VMPC-R

input:								
K; $k = 32$	{104, 9, 46, 231, 132, 149, 234, 147, 224, 97, 230, 127, 124, 109, 34, 171, 88, 185, 158, 23, 116, 69, 90, 195, 208, 17, 86, 175, 108, 29, 146, 219} (X=123; repeat 32 times:{X=X·134775813+1; output=X mod 256})							
V; $v = 32$	{149, 234, 147, 224, 97, 230, 127, 124, 109, 34, 171, 88, 185, 158, 23, 116, 69, 90, 195, 208, 17, 86, 175, 108, 29, 146, 219, 72, 105, 14, 71, 100} (X=132; repeat 32 times:{X=X·134775813+1; output=X mod 256})							
output of KSA:								
P index	0	1	2	3	252	253	254	255
P value	76	44	167	7	250	147	240	51
S index	0	1	2	3	252	253	254	255
S value	239	59	110	207	98	23	178	227
output of CSPRNG:								
output index	0	1	2	3	254	255	256	257
output value	219	178	157	119	2	155	62	20
output index	1000	1001	10000	10001	100000	100001	1000000	1000001
output value	3	239	236	81	195	11	186	127

Table 12. Test-output of VMPC-R

input:								
K; $k = 256$	{147, 224, 97, 230, 127, 124, 109, 34, 171, 88, 185, 158, 23, 116, 69, 90, 195, 208, 17, 86, 175, 108, 29, 146, 219, 72, 105, 14, 71, 100, 245, 202, 243, 192, 193, 198, 223, 92, 205, 2, 11, 56, 25, 126, 119, 84, 165, 58, 35, 176, 113, 54, 15, 76, 125, 114, 59, 40, 201, 238, 167, 68, 85, 170, 83, 160, 33, 166, 63, 60, 45, 226, 107, 24, 121, 94, 215, 52, 5, 26, 131, 144, 209, 22, 111, 44, 221, 82, 155, 8, 41, 206, 7, 36, 181, 138, 179, 128, 129, 134, 159, 28, 141, 194, 203, 248, 217, 62, 55, 20, 101, 250, 227, 112, 49, 246, 207, 12, 61, 50, 251, 232, 137, 174, 103, 4, 21, 106, 19, 96, 225, 102, 255, 252, 237, 162, 43, 216, 57, 30, 151, 244, 197, 218, 67, 80, 145, 214, 47, 236, 157, 18, 91, 200, 233, 142, 199, 228, 117, 74, 115, 64, 65, 70, 95, 220, 77, 130, 139, 184, 153, 254, 247, 212, 37, 186, 163, 48, 241, 182, 143, 204, 253, 242, 187, 168, 73, 110, 39, 196, 213, 42, 211, 32, 161, 38, 191, 188, 173, 98, 235, 152, 249, 222, 87, 180, 133, 154, 3, 16, 81, 150, 239, 172, 93, 210, 27, 136, 169, 78, 135, 164, 53, 10, 51, 0, 1, 6, 31, 156, 13, 66, 75, 120, 89, 190, 183, 148, 229, 122, 99, 240, 177, 118, 79, 140, 189, 178, 123, 104, 9, 46, 231, 132, 149, 234} (X=234; repeat 256 times:{X=X·134775813+1; output=X mod 256})							
V; $v = 8$	{255, 250, 200, 150, 100, 50, 5, 1}							
output of KSA:								
P index	0	1	2	3	252	253	254	255
P value	10	34	13	239	209	9	154	220
S index	0	1	2	3	252	253	254	255
S value	253	106	200	178	75	251	129	209
output of CSPRNG:								
output index	0	1	2	3	254	255	256	257
output value	201	85	155	17	187	48	55	198
output index	1000	1001	10000	10001	100000	100001	1000000	1000001
output value	110	179	189	210	4	15	253	83

8 Conclusions

We presented a new Cryptographically Secure Pseudo-Random Number Generator, VMPC-R. We devised a universal statistical test which we used to probe over 250 candidates before we found an algorithm which according to the test produces pseudorandom output. In contrast to the 16 or 32 bit array-based algorithms VMPC-R employs the original 8-bit RC4-like design to which we referred to as plain-permutation-based. While it hampers the cipher's performance, applying permutations as the arrays (along with the 8-bit word size) and resigning from using array-dependent cyclic rotation of bits operations appeared to resist several intrinsic weaknesses of array-based ciphers outlined by Paul and Preneel. Our algorithm is more complex and slower than RC4. The scale of the added complexity was the minimal we found indispensable to pass our measures of statistical quality. We conclude that according to our analyses VMPC-R appears to be the simplest RC4-like algorithm capable of producing pseudorandom output.

In the possible further research it might be worth verifying whether the algorithm indeed generates as pseudorandom output and is as secure as we believe it is and whether it is indeed impossible to find a simpler design in this scope which would match the results of VMPC-R.

References

1. 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.
2. 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.
3. Souradyuti Paul, Bart Preneel: Analysis of Non-fortuitous Predictive States of the RC4 Keystream Generator. Proceedings of INDOCRYPT 2003, LNCS, vol. 2904, Springer-Verlag, 2003, pages 52-67.
4. 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.
5. Alexander Maximov: Some Words on Cryptanalysis of Stream Ciphers. Ph.D. Thesis, Lund University 2006, ISBN 91-7167-039-4
6. Alexander Maximov, Dmitry Khovratovich: New State Recovery Attack on RC4 Proceedings of CRYPTO 2008, LNCS, vol. 5157, Springer-Verlag, 2008, pages 297-316.
7. Bartosz Zoltak: VMPC One-Way Function and Stream Cipher Proceedings of FSE 2004, LNCS, vol. 3017, Springer-Verlag, 2004, pages 210-225.
8. Kamil Kulesza: On inverting the VMPC one-way function. Department of Applied Mathematics and Theoretical Physics, University of Cambridge, http://talks.cam.ac.uk/talk/index/4950
9. Jonathan J. Hoch, Adi Shamir Fault Analysis of Stream Ciphers. Proceedings of CHES 2004, LNCS, vol. 3156, Springer-Verlag, 2004
10. Eli Biham, Louis Granboulan, Phong Q. Nguyen: Impossible Fault Analysis of RC4 and Differential Fault Analysis of RC4. Proceedings of FSE 2005, LNCS, vol. 3557, Springer-Verlag, 2005, pages 359-367.
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. Serge Mister, Stafford E. Tavares: Cryptanalysis of RC4-like Ciphers. Proceedings of SAC 1998, LNCS, vol. 1556, Springer-Verlag, 1999.
14. 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.
15. Scott R. Fluhrer, David A. McGrew: Statistical Analysis of the Alleged RC4 Keystream Generator. Proceedings of FSE 2000, LNCS, vol. 1978, Springer-Verlag, 2001.
16. Itsik Mantin, Adi Shamir: A Practical Attack on Broadcast RC4. Proceedings of FSE 2001, LNCS, vol. 2355, Springer-Verlag, 2002.
17. Scott Fluhrer, Itsik Mantin, Adi Shamir: Weaknesses in the key scheduling algorithm of RC4. Proceedings of SAC 2001, LNCS, vol. 2259, Springer-Verlag 2001.
18. Jovan Dj. Golic: Linear Statistical Weakness of Alleged RC4 Keystream Generator. Proceedings of EUROCRYPT 1997, LNCS, vol. 1233, Springer-Verlag, 1997.

19. Alexander L. Grosul, Dan S. Wallach: A Related-Key Cryptanalysis of RC4. Technical Report TR-00-358, Department of Computer Science, Rice University, 2000.
20. A. F. Webster; Stafford E. Tavares. On the design of S-boxes. Proceedings of CRYPTO 1985, LNCS, vol. 218, Springer-Verlag, 1986.