Message Authentication (MAC) Algorithm For The VMPC-R (RC4-like) Stream Cipher

Bartosz Zoltak

www.vmpcfunction.com
bzoltak@vmpcfunction.com

Abstract. We propose an authenticated encryption scheme for the VMPC-R stream cipher. VMPC-R is an RC4-like algorithm proposed in 2013. It was created in a challenge to find a bias-free cipher within the RC4 design scope and to the best of our knowledge no security weakness in it has been published to date. The contribution of this paper is an algorithm to compute Message Authentication Codes (MACs) along with VMPC-R encryption. We also propose a simple method of transforming the MAC computation algorithm into a hash function.

Keywords: stream cipher; RC4; VMPC-R; distinguishing attack; bias; Message Authentication Code (MAC); hash function

1 Introduction

Stream ciphers follow a natural concept of encrypting data with a pseudorandom keystream. One of the many approaches to stream cipher construction is an RC4-like design scope. A large number of weaknesses were found in RC4: [12], [7], [8], [16], [22], [23], [25] list just a small subset of them. Many of them are distinguishing attacks, revealing statistical deviations (biases) of various statistical properties of the generated keystream from a perfectly random model.

Many attempts to improve the design of RC4 were made, however none of the ones we are aware of (RC4A, VMPC, IA, IBAA, ISAAC, Py, Py6, PyPy, NGG, GGHN, HC-256, Spritz, to name just a few) managed to resist new distinguishing attacks. Spritz [5] was an attempt by the designer of the original RC4 himself, Ronald Rivest and Jacob Schuldt. Spritz was attacked in [2], [6].

In 2010-2013 we made our own attempt to contribute to the challenge. We devised a battery of statistical tests specifically for the job, utilized over 100 computers to perform the testing, considered hundreds of design ideas and arrived at a single algorithm capable of passing the tests. We probed over $10^{15,3}$ outputs of the final candidate with different word sizes and found no bias in its output. See [3] for more details.

We also tested a large number of the known RC4-like algorithms and our battery revealed biases in any of them, including Spritz. See [4] and [2].

To the best of our knowledge, VMPC-R produces (by far) the highest quality output in terms of statistical properties among all the RC4-like ciphers we know of. Although VMPC-R was published in 2013, we are still not aware of any publication reporting any weakness in it. The suffix "R" in the name of the algorithm, VMPC-R, stands for Random.

One disadvantage of VMPC-R is its low efficiency. On a 3 GHz desktop PC processor in assembler implementation it obtains the efficiency of about 70 Mbytes/s (or 560 Mbits/s). We believe that the possible scope of its practical applications are limited to either low-bandwidth or small-data-size applications. One example could be the encryption of text. To illustrate, 1000 A4 pages of text could be encrypted with VMPC-R in about 0,05 of a second.

However the purpose of designing VMPC-R was not as much practical, as theoretical. VMPC-R was our attempt to answer a simple question - how complex an RC4-like algorithm would need to be to actually produce a keystream indistinguishable from a truly random source.

Note that VMPC-R is the simplest algorithm that we were able to build to pass the statistical tests we threw at it. This way the results were both a success and a failure: we found an unbiased (according to our results) RC4-like algorithm, but its complexity was so high that it hampered its efficiency below the level required to consider it a practical general-purpose encryption algorithm.

2 Contribution of this paper

The contribution of this paper is the proposition of an authenticated encryption scheme for VMPC-R. In practical applications message authentication is often an indispensable feature. It provides the means of verification whether the message was correctly decrypted. For this purpose, a Message Authentication Code (MAC, which is conceptually a checksum of the message and the key) is attached to the ciphertext. The MAC is also evaluated upon decryption and compared with the value stored with the ciphertext. If both are identical, the decrypted message can be deemed identical with the original plaintext. We also propose a simple method to transform the MAC algorithm into a hash function.

3 The VMPC-R Stream Cipher

Specification of VMPC-R can be found in Table 1. The internal state of the algorithm is initialized with the Key Scheduling Algorithm (KSA) specified in Table 2.

Table 1. VMPC-R Stream Cipher

N : word size; $N \in \{2, ..., 256\}$. Proposition: $N = 256$ a, b, c, d, e, f, n : integer variables P, S : N-element permutations of integers $\{0, ..., N-1\}$ $+$ denotes addition modulo N
Function *VMPC-R-Stream*(x): reset output repeat x times { append *VMPC-R-Stream-Data* to output *VMPC-R-Stream-Swap* }
Function *VMPC-R-Stream-Data*: $a = P[a + c + S[n]]$ $b = P[b + a]$ $c = P[c + b]$ $d = S[d + f + P[n]]$ $e = S[e + d]$ $f = S[f + e]$ output $S[S[S[c + d]] + 1]$
Function *VMPC-R-Stream-Swap*: swap $P[n]$ with $P[f]$ swap $S[n]$ with $S[a]$ $n = n + 1$

Table 2. VMPC-R Key Scheduling Algorithm

$N, P, S, a, b, c, d, e, f, n$ defined in Table 1
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
$+$ denotes addition modulo N

Function *VMPC-R-KSA*(K, k, V, v, z):
  if $(z = 0)$ set $\{a = b = c = d = e = f = 0; \forall i \in \{0, ..., N-1\}(P[i] = S[i] = i)\}$
  KSARound(K, k)
  KSARound(V, v)
  KSARound(K, k)
  $n = S[S[S[c + d]] + 1]$
  VMPC-R-Stream(N)

Function *KSARound*(Y, y):
  set $\{n = 0; i = 0\}$
  repeat $N \cdot \lceil y^2/(6N) \rceil$ times {
    $a = P[a + f + Y[i]] + i; \quad i = (i + 1) \mod y$
    $b = S[b + a + Y[i]] + i; \quad i = (i + 1) \mod y$
    $c = P[c + b + Y[i]] + i; \quad i = (i + 1) \mod y$
    $d = S[d + c + Y[i]] + i; \quad i = (i + 1) \mod y$
    $e = P[e + d + Y[i]] + i; \quad i = (i + 1) \mod y$
    $f = S[f + e + Y[i]] + i; \quad i = (i + 1) \mod y$

    swap $P[n]$ with $P[b]$
    swap $S[n]$ with $S[e]$
    swap $P[d]$ with $P[f]$
    swap $S[a]$ with $S[c]$

    $n = n + 1$ }

4 Authenticated encryption with VMPC-R

Message authentication is usually obtained using hash functions, which fed with the key, produce message-and-key-unique Message Authentication Codes (MAC). We propose to produce MACs without hash functions by integrating some additional operations with our cipher and utilizing its internal state. We also transform the proposed MAC algorithm into a hash function. The scheme can also work with many other RC4-like stream ciphers. Its specification can be found in Table 3.

Table 3. VMPC-R-MAC authenticated encryption scheme

$N, P, S, a, b, c, d, e, f, n$ defined in Table 1 K, k, V, v defined in Table 2 q : security level; $q \in \{4, ..., 16\}$. Proposition: $q = 8$ T : array of q integers M : array of q^2 integers h : integer variable L : length of message Z : message; array of L integers L_M : length of the MAC; $L_M \in \{1, ..., q^2\}$. Proposition: $L_M = q^2$ i, j : temporary integer variables $+$ denotes addition modulo N
set $\{h = 0; \forall i \in \{0, ..., q-1\}(T[i] = 0); \forall i \in \{0, ..., q^2-1\}(M[i] = 0)\}$ *VMPC-R-KSA*$(K, k, V, v, 0)$ *EncryptMAC* *Mix* *VMPC-R-KSA*$(M, q^2, T, q, 1)$ $MAC =$ *VMPC-R-Stream*(L_M)

Function *EncryptMAC*:
for $(j = 0, ..., L-1)$ repeat {
$Z[j] = Z[j]$ xor *VMPC-R-Stream-Data*
for $(i = 0, ..., q-2)$ repeat {
$T[i] = P[T[i] + T[i+1] + i]$ }
$T[q-1] = P[T[q-1] + e + Z[j]]$
for $(i = 0, ..., q-1)$ repeat {
$M[h+i] = M[h+i]$ xor $T[i]$ }
$h = (h + q) \mod q^2$
VMPC-R-Stream-Swap }

Function *Mix*:
for $(j = 1, ..., 2q)$ repeat {
VMPC-R-Stream-Data
for $(i = 0, ..., q-2)$ repeat {
$T[i] = P[T[i] + T[i+1] + j + i]$ }
$T[q-1] = P[T[q-1] + b + j + q - 1]$
for $(i = 0, ..., q-1)$ repeat {
$M[h+i] = M[h+i]$ xor $T[i]$ }
$h = (h + q) \mod q^2$
VMPC-R-Stream-Swap }

4.1 Design rationale

The general idea is to update the T array with the consecutive ciphertext words in such a way that a change of any plaintext/ciphertext word changes each element of T with probability 1. At the same time these changes are spread through the M array by leaving footprints of T on the consecutive q-element blocks of the q-times larger M array.

After encryption is finished the T and M arrays are mixed further with the *Mix* function to ensure that a change of last words of plaintext/ciphertext are spread through T and M with the required intensity and to thwart certain attack scenarios which will be discussed in Section 4.3.

The $+i$ operation in the $T[i] = P[T[i] + T[i+1] + i]$ step ensures that the T elements will not fall into a temporary state where from N consecutive equal T elements, $N-1$ would remain equal in the next iteration.

The extent of spreading the unconditional changes of T onto M appears to be hard to control and according to the most efficient message forgery attack we could find (see Section 4.6) the security of the scheme is $N^{q(q+5)/2-1}$. For $q = 8$ it evaluates to N^{-51} (408 bits with $N = 256$).

4.2 The unconditional changes to T

For notation simplicity let's assume $h = 0$. P is a permutation, therefore the $T[q-1] = P[T[q-1]+e+Z[j]]$ operation would change the value of $T[q-1]$ with probability 1 if $Z[j]$ was changed. As a result $M[q-1] = M[q-1]$ xor $T[q-1]$ would also change with probability 1.

In the next iteration the change of $T[q-1]$ will spread further onto T regardless of what new Z values are fed to the algorithm: $T[q-2] = P[T[q-2]+T[q-1]+q-2]$ will change with probability 1. As a result $M[2q-2] = M[2q-2]$ xor $T[q-2]$ will change with probability 1, too. The process will continue through q iterations and in effect, $T[q-1]$, $T[q-2]$, $T[q-3]$,..., $T[0]$ and $M[q-1]$, $M[2q-2]$, $M[3q-3]$,..., $M[q^2-q]$ will be changed with probability 1. This property will be investigated further in Section 4.6.

4.3 The *Mix* function

The first objective of the *Mix* function is to spread the last words of the message onto all the elements of T and M. We analyse it further in the diffusion effect in Section 4.4.

The second objective is to thwart forgery attacks which increase the length of the message. The same input data transformed by the encryption phase by $T[i] = P[T[i] + T[i+1] + i]$ and in each iteration of the *Mix* function by $T[i] = P[T[i] + T[i+1] + j + i]$ will produce different outputs in each of the cases (the value of j is different in each iteration of *Mix*). As a result the longer message which enters *Mix* will perform different operations on T than the shorter one which will already be in iteration 2 (or further).

To illustrate the problem, let's consider a reduced version of the *Mix* function which performs the same operations on T as the encryption phase does, i.e. $T[i] = P[T[i] + T[i+1] + i]$ and $T[q-1] = P[T[q-1] + e]$. Suppose an adversary appends zero to the original message Z_1 and obtains one-word-longer message Z_2. Z_2 would enter *Mix* one iteration later than Z_1. The first *Mix* iteration for Z_1 would change T in the same way as the last iteration of encryption of Z_2 would. T would stay unchanged up to after the last iteration of *Mix* for Z_1. The last iteration for Z_2 would leave all the q elements of T unchanged with an unacceptably high probability of N^{-q}. Such an attack is not possible in the full version of the algorithm.

Additionally, the variables (e and b) used in the $T[q-1] = P[T[q-1] + e + Z[j]]$ step of the encryption phase and in the $T[q-1] = P[T[q-1] + b + j + q - 1]$ step of the *Mix*

function are different from each other in order to eliminate the trivial possibility of obtaining equal $T[q-1]$ elements by appending one integer $x = j + q - 1$ to the original message Z.

An attack approach to append $2q$ words to have both *Mix* functions produce the same outputs at only a cost of N^{-q} (of equalling the T arrays right before *Mix*) would result in more changes done to M by the longer message before reaching *Mix* than are necessary by our forgery attack (Section 4.6) and the approach would exceed our attack's complexity.

4.4 Diffusion effect of the *Mix* function

The number of iterations of the *Mix* function is set at $2q$ which appears to provide proper diffusion effect of changes of the last word of plaintext/ciphertext onto the T and M arrays. We tested two scenarios where we encrypted messages Z_1 and Z_2 with the same key and the same initialization vector and analysed the resulting T arrays: T_1 and T_2 and the M arrays: M_1 and M_2. Let L_1 and L_2 denote the lengths of Z_1 and Z_2, respectively. The two scenarios were:

- $L = L_1 = L_2; \forall i \in \{0, ..., L-2\}(Z_1[i] = Z_2[i]);\ Z_1[L-1] \neq Z_2[L-1]$
- $L_2 = L_1 + 1; \forall i \in \{0, ..., L_1 - 1\}(Z_1[i] = Z_2[i]);\ Z_2[L_2 - 1] =$random

We chose these scenarios as the worst-case ones for the diffusion effect (the last word of the message has the smallest influence on T and M). The tests showed that the correlations between T_1 and T_2 and between M_1 and M_2 are indistinguishable from the correlations we would expect from two randomly generated arrays. The diffusion effect will be further magnified by the step *VMPC-R-KSA*$(M, q^2, T, q, 1)$.

4.5 Security and implementation of VMPC-R-MAC

The most efficient message forgery attack we could find, described in Section 4.6, determines the security of the scheme at $N^{q(q+5)/2-1}$. For practical implementations we propose to choose $N = 256$ and $q = 8$, which evaluates the security level to 256^{-51} (408 bits). If more was needed, we propose to increment q by multiples of 4 due to implementation easiness of the $M[h+i] = M[h+i]$ xor $T[i]$ step on 32 (or 64)-bit processors, which can execute 4 such operations with a single xor instruction.

If speed is the priority, the scheme can be implemented with $q = 4$, which would yield the security level of 256^{-17} (136 bits).

The proposed upper limit of $q = 16$ is set by the capacity of the VMPC-R-KSA algorithm, which can process up to 256-byte arrays. At $q = 16$ the security level reaches $256^{-167} = 1336$ bits.

However, the theoretical upper limit for q does not exist. The only modification the algorithm would require to accept $q > 16$ would be a higher-capacity final-mixing function in place of the currently used VMPC-R-KSA.

4.6 Our message forgery attack

The most efficient message forgery attack we could devise begins and ends before entering the *Mix* function and changes several words of the message to obtain an unchanged *MAC* with probability $N^{q(q+5)/2-1}$. Let T_g and M_g denote the values of the T and M arrays recorded for the genuine message. For notation simplicity let's assume $h = 0$. The attack is analogous for any other value of h.

The attack begins with a change of q consecutive words of the message in iterations $\{i, i+1, ..., i+q-1\}$. These changes should be made in such a way to make the $T[q-1] = P[T[q-1]+e+Z[j]]$ step change the following elements of T, compared to T_g (changes taking place with probability 1 after any change of $Z[j]$ are marked with *):

in iteration $i+0$: change $T[q-1]$*
in iteration $i+1$: change $T[q-2]$*, $T[q-1]$
in iteration $i+2$: change $T[q-3]$*, $T[q-2]$, $T[q-1]$
...
in iteration $i+q-1$: change $T[0]$*, $T[1], T[2], ..., T[q-1]$

If these changes take place, the following elements of M will be changed with probability 1:

in iteration $i+0$: $M[q-1]$
in iteration $i+1$: $M[2q-2]$, $M[2q-1]$
in iteration $i+2$: $M[3q-3]$, $M[3q-2]$, $M[3q-1]$
...
in iteration $i+q-1$: $M[q^2-q]$, $M[q^2-q+1]$, $M[q^2-q+2]$,...,$M[q^2-1]$

Here a total of $1+2+3+...+q = q(1+q)/2$ elements of M will be different from those in M_g. At this point the attack requires its first low-probability-event which is the necessary condition to proceed. The adversary needs to obtain $q-1$ equations in the next iteration $i+q$:

$$T[x] = T_g[x] \text{ for all } x \in \{0, 1, ..., q-2\}$$

This can be achieved with probability $N^{-(q-1)}$ and it stops T from making more changes to M.

To successfully forge the message, the adversary has to revert the changes already made to M. This can be completed by proceeding analogously, i.e. by changing q consecutive words of the message in iterations $\{q+i, q+i+1, ..., q+i+q-1\}$. These changes should be made again in such a way to change the following elements of T, compared to T_g:

in iteration $q+i+0$: change $T[q-1]$
in iteration $q+i+1$: change $T[q-2]$*, $T[q-1]$
in iteration $q+i+2$: change $T[q-3]$*, $T[q-2]$, $T[q-1]$
...
in iteration $q+i+q-1$: change $T[0]$*, $T[1]$, $T[2]$,..., $T[q-1]$

If these changes take place, the following elements of M will be changed with probability 1:

in iteration $q+i+0$: $M[q-1]$
in iteration $q+i+1$: $M[2q-2]$, $M[2q-1]$
in iteration $q+i+2$: $M[3q-3]$, $M[3q-2]$, $M[3q-1]$
...
in iteration $q+i+q-1$: $M[q^2-q]$, $M[q^2-q+1]$, $M[q^2-q+2]$,...,$M[q^2-1]$

At this point the attack requires two more low-probability-events to succeed. The first one is to obtain q equations in the next iteration $i+2q$:

$$T[x] = T_g[x] \text{ for all } x \in \{0, 1, ..., q-1\}$$

The adversary will achieve this with probability N^{-q}. Note that $T[q-1]$ also needs to return to its genuine value of $T_g[q-1]$ here. The final event is to have all the $q(1+q)/2$ changed elements of M return to their genuine M_g values. This would take place with probability $N^{-q(1+q)/2}$.

At this point the attack is successful with probability $N^{-(q-1+q+q(1+q)/2)} = N^{q(q+5)/2-1}$. For $q = 4$ it evaluates to N^{-17} (136 bits for $N = 256$). For the proposed $q = 8$: N^{-51} (408 bits), for q=12: N^{-101} (808 bits), and for $q = 16 : N^{-167}$ (1336 bits).

As far as we could analyse the algorithm this is the fastest way to revert all the unavoidable changes of T and M caused by the smallest change of the message (a change of a single word). We expect that attacks applying more changes to the message would cause more extensive changes to T and M and that these changes would be not easier to control than in our attack, which would result in the same or higher complexities.

4.7 The VMPC-R-HASH function

An attempt to extend RC4 into a hash function was proposed by Chang, Gupta Mridul Nandi in 2006 [18]. Collisions were found however by Indesteege and Preneel in 2008 [19]. The approach we take here is different. A hash function can be obtained by setting a constant value of the key and the initialization vector for the VMPC-R-MAC algorithm. We propose the following values:

$$K_H = V_H = \{0, 0, 0, 0, 0, 0\}$$

To produce the VMPC-R-HASH value of message Z_H, input K_H and V_H to the VMPC-R-KSA (Table 2) and process Z_H with the VMPC-R-MAC (Table 3).

The obvious implementation modification of the VMPC-R-MAC algorithm is to substitute the following steps in the *EncryptMAC* function:

$$Z[j] = Z[j] \text{ xor } \textit{VMPC-R-Stream-Data}$$
$$T[q-1] = P[T[q-1] + e + Z[j]]$$

with these steps, respectively (i.e. use a temporary integer variable *temp* in place of encrypting the message):

$$temp = Z[j] \text{ xor } \textit{VMPC-R-Stream-Data}$$
$$T[q-1] = P[T[q-1] + e + temp]$$

The detailed analysis of this hash function reaches beyond the scope of this paper. However, given the security analysis of the VMPC-R-MAC scheme and its resistance to our message forgery attack of $N^{q(q+5)/2-1}$ we assume that the VMPC-R-HASH should provide significant resistance to collision attacks.

5 VMPC-R MAC test vectors

Table 4 gives the output words generated by the VMPC-R cipher with $N = 256$ for a given 8-word key (K) and 8-word initialization vector (V). The table also gives the values of P and S after the Key Scheduling Algorithm (before encryption).

Table 4. Test output of VMPC-R Stream Cipher

K; $k = 8$	{11, 22, 33, 144, 155, 166, 233, 244}							
V; $v = 8$	{255, 250, 200, 150, 100, 50, 5, 1}							
P index	0	1	2	3	252	253	254	255
P value	233	177	250	165	43	123	169	201
S index	0	1	2	3	252	253	254	255
S value	235	158	236	32	10	29	145	30
output index	0	1	2	3	254	255	256	257
output value	253	15	246	141	70	145	94	212
output index	1000	1001	10000	10001	100000	100001	1000000	1000001
output value	187	151	6	108	8	21	65	215

Table 5 gives the output MAC of both the VMPC-R-MAC and the VMPC-R-HASH algorithms for $N = 256$, $q = 8$.

Table 5. Test output of VMPC-R-MAC and VMPC-R-HASH

K; $k = 8$	{0, 0, 0, 0, 0, 0, 0, 0}									
V; $v = 8$	{0, 0, 0, 0, 0, 0, 0, 0}									
Z; $L = 1000002$	{0,1,...,254,255,0,1,...65} ($Z[i] = i$ modulo 256)									
MAC/HASH index	0	1	2	3	4	5	6	7	8	9
MAC/HASH value	250	137	167	97	207	190	8	142	158	57
MAC/HASH index	10	11	12	13	14	15	16	17	18	19
MAC/HASH value	223	124	214	55	86	168	73	35	121	18
ciphertext index	0	1	2	3	254	255	256	257		
ciphertext value	62	79	39	154	145	123	200	171		
ciphertext index	1000	1001	10000	10001	100000	100001	1000000	1000001		
ciphertext value	209	135	59	18	66	112	90	155		

6 Conclusions

We proposed an algorithm to compute Message Authentication Codes for the VMPC-R stream cipher. The design is cipher-specific in a sense that it utilizes some data from the cipher's internal state but it could be easily ported to work with different RC4-like ciphers. We also proposed a simple method of deriving a hash function from it. The algorithm is scalable (using the value of the q parameter) in terms of security-performance tradeoff.

We described our best message forgery attack against it, which for the recommended value of $q = 8$ succeeds with probability N^{-51}, which for the $N = 256$ used in practical applications evaluates to 408 bits of security.

The primary objective of designing the scheme was uncompromised security. This came at the cost of performance. While the bare VMPC-R stream cipher achieves a not-so-noble speed of about 70 Mbytes/s on a 3 GHz desktop PC, the full authenticated encryption runs at the speed of only about 30 Mbytes/s (using assembler implementations).

This performance disqualifies the scheme as a general purpose authenticated encryption algorithm. However in applications not requiring high bandwidth or operating on small data sizes (like e.g. text), the scheme works well and according to our research offers a very high level of security.

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: Statistical weakness in Spritz against VMPC-R: in search for the RC4 replacement. IACR Cryptology ePrint Archive Report 2014/985. https://eprint.iacr.org/2014/985.pdf
3. Bartosz Zoltak: VMPC-R Cryptographically Secure Pseudo-Random Number Generator Alternative to RC4. IACR Cryptology ePrint Archive Report 2013/768. https://eprint.iacr.org/2013/768.pdf
4. Bartosz Zoltak: Statistical weaknesses in 20 RC4-like algorithms and (probably) the simplest algorithm free from these weaknesses - VMPC-R IACR Cryptology ePrint Archive Report 2014/315. https://eprint.iacr.org/2014/315.pdf
5. Ronald L. Rivest, Jacob C. N. Schuldt: Spritz - a spongy RC4-like stream cipher and hash function. http://people.csail.mit.edu/rivest/pubs/RS14.pdf. Presented at CRYPTO 2014 Rump Session. IACR Cryptology ePrint Archive Report 2016/856. https://eprint.iacr.org/2016/856.pdf.
6. Subhadeep Banik, Takanori Isobe: Cryptanalysis of the Full Spritz Stream Cipher. IACR Cryptology ePrint Archive Report 2016/092. https://eprint.iacr.org/2016/092.pdf
7. Subhamoy Maitra: The Index j in RC4 is not Pseudo-random due to Non-existence of Finney Cycle. IACR Cryptology ePrint Archive Report 2015/1043. https://eprint.iacr.org/2015/1043.pdf
8. Sourav Sen Gupta, Subhamoy Maitra, Goutam Paul, and Santanu Sarkar: (Non-)Random Sequences from (Non-)Random Permutations - Analysis of RC4 stream cipher. IACR Cryptology ePrint Archive Report 2011/448. https://eprint.iacr.org/2011/448.pdf
9. 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.
10. Alexander Maximov: Some Words on Cryptanalysis of Stream Ciphers. Ph.D. Thesis, Lund University 2006, ISBN 91-7167-039-4
11. Alexander Maximov, Dmitry Khovratovich: New State Recovery Attack on RC4 Proceedings of CRYPTO 2008, LNCS, vol. 5157, Springer-Verlag, 2008, pages 297-316.
12. 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.
13. 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.
14. 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.
15. 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.
16. 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
17. 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
18. Donghoon Chang, Kishan Chand Gupta, and Mridul Nandi RC4-Hash: A New Hash Function based on RC4 Proceedings of INDOCRYPT 2006, LNCS, vol. 4329, Springer-Verlag, 2006, pages 80-94.
19. Sebastiaan Indesteege, Bart Preneel: Collisions for RC4-Hash Proceedings of ISC 2008, LNCS, vol. 5222, Springer-Verlag, 2008, pages 355-366.
20. Serge Mister, Stafford E. Tavares: Cryptanalysis of RC4-like Ciphers. Proceedings of SAC 1998, LNCS, vol. 1556, Springer-Verlag, 1999.
21. 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.
22. Scott R. Fluhrer, David A. McGrew: Statistical Analysis of the Alleged RC4 Keystream Generator. Proceedings of FSE 2000, LNCS, vol. 1978, Springer-Verlag, 2001.
23. Itsik Mantin, Adi Shamir: A Practical Attack on Broadcast RC4. Proceedings of FSE 2001, LNCS, vol. 2355, Springer-Verlag, 2002.
24. Scott Fluhrer, Itsik Mantin, Adi Shamir: Weaknesses in the Key Scheduling Algorithm of RC4. Proceedings of SAC 2001, LNCS, vol. 2259, Springer-Verlag 2001.
25. Jovan Dj. Golic: Linear Statistical Weakness of Alleged RC4 Keystream Generator. Proceedings of EUROCRYPT 1997, LNCS, vol. 1233, Springer-Verlag 1997.
26. Alexander L. Grosul, Dan S. Wallach: A Related-Key Cryptanalysis of RC4. Technical Report TR-00-358, Department of Computer Science, Rice University, 2000.