VMPC One-Way Function and Stream Cipher

Bartosz Zoltak

http://www.vmpcfunction.com
bzoltak@vmpcfunction.com
This paper was presented at FSE'04, Delhi, India, 5-7.FEB.2004

Abstract. A simple one-way function along with its proposed application in symmetric cryptography is described. The function is computable with three elementary operations on permutations per byte. Inverting the function, using the most efficient method known to the author, is estimated to require an average computational effort of about 2^{260} operations. The proposed stream cipher based on the function was designed to be efficient in software implementations and, in particular, to eliminate the known weaknesses of the alleged RC4 keystream generator while retaining most of its speed and simplicity.

Keywords: one-way function; stream cipher; cryptanalysis; RC4; lower bound

1 Introduction

A simple transformation of permutations appearing to be hard to invert is described together with its proposed practical application in a software-efficient symmetric encryption algorithm.

The transformation, here termed "VMPC" function as an abbreviation of "Variably Modified Permutation Composition", is a combination of elementary operations on permutations and integers. The simplest variant of the function can be coded with three basic "MOV" instructions from the Intel 80x86 processor instruction set per byte. When applied on 256-element permutations (a comfortable size in practical cryptographic applications), the function requires an estimated average of 2^{260} computational operations to be inverted using the most efficient method known to the author.

The very low computational cost required to obtain practical one-way property makes the function a plausible candidate for cryptographic applications. The simplicity of the function could also raise a question whether it might be possible to prove a lower bound on the complexity of inverting it. This currently is an open problem and a possible subject of future research.

A proposition of an encryption algorithm constructed as a stream cipher based on the VMPC function is described in sections 8-14. The cipher was designed to be both efficient in software implementations and to resist the known attacks against this kind of algorithms (like the alleged RC4 keystream generator) – in particular against attacks distinguishing the keystream from a truly random source and attacks against the cipher's Key Scheduling Algorithm (KSA).

2 Definition of the VMPC function

Notation:

n, P, Q : P, Q : n-element permutations. For simplicity of further references P and Q are assumed to be one-to-one mappings $A \rightarrow A$; $A = \{0, 1, \ldots, n-1\}$
k : Level of the function; $k < n$
$+$: addition modulo n

Definition:

A k-level VMPC function, referred to as VMPC_k, is such transformation of P into Q, where

$$Q[x] = P[P_k[P_{k-1}[\ldots[P_1[P[x]]]\ldots]]],$$

$x \in \{0, 1, \ldots, n-1\}$,
P_i is an n-element permutation such that $P_i[x] = \mathrm{f}_i(P[x])$, where f_i is any function such that $P_i[x] \neq P[x] \neq P_j[x]$ for $i \in \{1, 2, ..., k\}, j \in \{1, 2, ..., k\}, i \neq j$.
For simplicity of further references f_i is assumed to be $\mathrm{f}_i(x) = x + i$

Example: $Q = \mathrm{VMPC}_1(P) : Q[x] = P[P[P[x]] + 1]$

3 Difficulty of inverting the VMPC function

n-element permutation P has to be recovered from the n-element permutation Q, where $Q = \mathrm{VMPC}_k(P)$.

By definition each element of Q is formed by $k + 2$, usually different, elements of P. One element of Q can be formed by many possible configurations of elements of P (e.g. for $Q = \mathrm{VMPC}_1(P) : Q[X] = Y$ can be formed by $P[X] = a, P[a] = b, P[b + 1] = Y$ for any reasonable combination of values of a and b).

All possible configurations are equally likely to be correct. If any of them is chosen, it needs to be verified with all of those elements of Q which use any of the elements of P included in the picked configuration.

Each element of P is usually used to form $k + 2$ different elements of Q thus $(k+2) \times (k+1)$ new elements of Q usually need to be inverted (all $k+2$ elements of P used to form each of these elements of Q need to be revealed) to verify the elements of P from the picked configuration.

Because the cycle structure of P is corrupted by the addition operation(s) it is usually impossible to find two different elements of Q, which share at least $k + 1$ elements of P.

Instead only such element of Q can usually be found, name it $Q[x]$, which shares only one of the $k+2$ elements of P with another element of Q. This forces k elements of P used to form $Q[x]$ to be guessed to invert $Q[x]$.

However at each new guessed element of P there usually occur $k+1$ new elements of Q which use this element of P and which need to be inverted to verify the guess.

The algorithm falls into a loop, where at every step usually k new elements of P need to be guessed to enable continuation of verification of the previously guessed elements. In consequence the $k+2$ elements of P picked at the beginning of the process indirectly depend on all n elements of Q.

The described scenario is the case usually. In some circumstances the verification process can be simplified by benefiting from coincidences (where for example it is possible to find two elements of Q, which share more than one element of P (e.g. for $Q = \mathrm{VMPC}_1(P) : Q[2] = 3 : P[2] = 4, P[4] = 8, P[9] = 3$ and $Q[1] = 8 : P[1] = 9, P[9] = 3, P[4] = 8$)).

A proposed algorithm for inverting the VMPC function (Section 6) was optimized to benefit from the possible coincidences. The average number of elements of P which need to be guessed for $n = 256$ was reduced to about 34 for 1-level VMPC function, to about 57 for 2-level VMPC, to about 77 for 3-level VMPC and to about 92 for 4-level VMPC function.

Searching through half of the possible states of these elements of P requires on average about 2^{260} steps for 1-level VMPC function, about 2^{420} for 2-level VMPC, about 2^{550} for 3-level VMPC and about 2^{660} steps for 4-level VMPC function.

4 A 3-instruction implementation of the VMPC function

Implementation of 1-level VMPC function, where $Q[x] = P[P[P[x]] + 1]$, for 256-element permutations P and Q in assembly language is described.

Assume that :

- Pm is a 257-byte array indexed by numbers from 0 to 256. The P permutation is stored in the Pm array at indexes from 0 to 255 ($Pm[0...255] = P$) and $Pm[256] = Pm[0]$
- the EAX 32-bit register specifies which element of the Q permutation to compute ("AL" always denotes 8 least significant bits of EAX, here EAX=AL)

Execute:

Table 1. Assembler implementation of 1-level VMPC function

Instruction	Description
MOV AL, [Pm] + EAX	Store (EAX=AL)-th element of Pm in AL
MOV AL, [Pm] + EAX	Store (EAX=AL)-th element of Pm in AL
MOV AL, [Pm] + EAX+1	Store ((EAX=AL)+1)-th element of Pm in AL

The 3 MOV instructions in Table 1 store the EAX-th element of permutation Q, where Q=VMPC$_1$(P), in the AL (and EAX) register.

5 Example values of the VMPC function

Values of 1,2,3 and 4-level VMPC function of an example 10-element permutation P are given in Table 2:

Table 2. Example values of the VMPC function

index	0	1	2	3	4	5	6	7	8	9
P	2	0	4	3	6	9	7	8	5	1
Q$_1$=VMPC$_1$(P)	9	3	8	6	5	4	1	7	2	0
Q$_2$=VMPC$_2$(P)	0	9	2	5	8	7	3	1	6	4
Q$_3$=VMPC$_3$(P)	3	4	9	5	0	2	7	6	1	8
Q$_4$=VMPC$_4$(P)	8	5	3	1	6	7	0	2	9	4

6 Proposed algorithm for inverting the VMPC function

The algorithm outputs an n-element permutation P satisfying Q = VMPC$_k$(P).

Notation:

P : n-element table the searched permutation will be stored in
X, Y : temporary variables
$Argument, Value, Base, Parameter$ of an element of P:
For an element $P[x] = y : x$ is termed the $Argument$; x can be either the $Base$ or the $Parameter$. y is termed the $Value$; y is the $Parameter$ or the $Base$ respectively.

Example: For an element $P[3] = 5$: If *Argument* 3 is the *Parameter*, *Value* 5 is the *Base*.

1.1) Reveal one element of P by assuming $P[X] = Y$, where
    X and Y are any values from $\{0, 1, ..., n-1\}$
1.2) Choose at random whether X is the *Base* and Y the *Parameter*
    or Y the *Base* and X the *Parameter* of the element $P[X] = Y$.
    Denote $P[X] = Y$ as the *Current* element of P.
2) Reveal all possible elements of P by running the Deducing Process
    (see Sect. 6.1)
3) If n elements of P are revealed with no contradiction:
    Terminate the algorithm and output P
4) If fewer than n elements of P are revealed with no contradiction:
    4.1) Reveal a new element of P by running the Selecting Process (see Sect. 6.2).
        Denote the revealed element as the *Current* element of P.
    4.2) Save the *Parameter* of the *Current* element of P
    4.3) Go to step 2
5) If a contradiction occurred in step 2:
    5.1) Remove all elements of P revealed in step 2 when
        the *Current* element of P had been revealed
    5.2) Increment modulo n the *Parameter* of the *Current* element of P
    5.3) If the *Parameter* of the *Current* element of P returned
        to the value saved in step 4.2:
        5.3.1) Remove the *Current* element of P
        5.3.2) Denote the element, which had been the *Current* element of P
            directly before the element removed in step 5.3.1 became
            the *Current* one, as the *Current* element of P
        5.3.3) Go to step 5.1
6) Go to step 2

6.1 The Deducing Process

The Deducing Process reveals all possible elements of P given Q and given the already revealed elements of P.

Notation as in Section 6, with:

C, A : temporary variables
Statement y : A sequence of $k+2$ elements of P used to compute $Q[y]$
Word x of *Statement* y : The x-th element of the sequence of $k+2$ elements of P used to compute $Q[y]$

Example: For $Q = \text{VMPC}_2(P) : Q[x] = P[P[P[P[x]] + 1] + 2]$:

Assume that $P[2] = 3, P[3] = 5, P[6] = 2, P[4] = 7$, which forms $Q[2] = 7$.
The elements $P[2] = 3, P[3] = 5, P[6] = 2, P[4] = 7$ form *Statement* 2.
The element $P[2] = 3$ is *Word* 1 of *Statement* 2; $P[3] = 5$ is *Word* 2 of *Statement* 2, etc.

1.1) Set C to 0
1.2) Set A to 0
2) If the element $P[A]$ is revealed:
  2.1) If the element $P[A]$ and k other revealed elements of P fit a general pattern of $k + 1$ *Words* of any *Statement*:
    Deduce the remaining *Word* of that *Statement* (see *Example 6.1.1*)
  2.2) If the deduced *Word* is not a revealed element of P:
    2.2.1) Reveal the deduced *Word* as an element of P
    2.2.2) Set C to 1
  2.3) If the deduced *Word* contradicts any already revealed element of P (see *Example 6.1.2*):
    Output a contradiction and terminate the Deducing Process
3.1) Increment A
3.2) If A is lower than n: Go to step 2
3.3) If C is equal 1: Go to step 1.1

Example 6.1.1. For $Q = \text{VMPC}_2(P) : Q[x] = P[P[P[P[x]] + 1] + 2]$:

Assume that $Q[0] = 9$ and that the following elements of P are revealed:
$P[0] = 1$, $P[1] = 3$, $P[8] = 9$.
Word 3 of *Statement* 0 can be deduced as $P'[4] = 6$ $(P'[3 + 1] = 8 - 2)$

Example 6.1.2. For $Q = \text{VMPC}_2(P) : Q[x] = P[P[P[P[x]] + 1] + 2]$:

Assume that $Q[7] = 2$ and that the following elements of P are revealed:
$P[1] = 8$, $P[9] = 3$, $P[5] = 2$ and $P[6] = 1$.
Word 1 of *Statement* 7, deduced as $P'[7] = 1$,
contradicts the already revealed element $P[6] = 1$

6.2 The Selecting Process (for k not higher than 4)

The Selecting Process selects such new element of P to be revealed which is expected to maximize the number of elements of P possible to deduce in further steps of the inverting algorithm. The Selecting Process outputs a selected *Base* and a randomly chosen *Parameter* of a new element of P.

Notation as in Section 6.1, with:

B, R : temporary variables
Ta, Tv : temporary tables
$Weight$: table of numbers: $Weight[0; 1; 2; 3; 4] = (0; 2; 5; 9; 14)$.
Example: $Weight[3] = 9$

1.1) Set Ta and Tv to 0
1.2) Set B to 0
1.3) Set R to 1
2) Count the number of revealed elements of P which fit a general pattern of $Words$ of a $Statement$ in which an unrevealed element of P with $Argument$ B would be $Word$ R. Increment $Ta[B]$ by $Weight$ of this number (see *Example 6.2.1*)
3) Count the number of revealed elements of P which fit a general pattern of $Words$ of a $Statement$ in which an unrevealed element of P with $Value$ B would be $Word$ R. Increment $Tv[B]$ by $Weight$ of this number
4.1) Increment R
4.2) If R is lower than $k + 3$: Go to step 2
4.3) Increment B
4.4) If B is lower than n: Go to step 1.3
5.1) Pick any index of Ta or Tv for which the number stored in tables Ta and Tv is maximal
5.2) If the index picked in step 5.1 is an index of Ta:
   5.2.1) Store this index in variable X
   5.2.2) Generate a random number $Y \in \{0, 1, \ldots, n-1\}$, such that an element of P with $Value$ Y is not revealed
   5.2.3) Output $P'[X] = Y$, where X is the $Base$ and Y is the $Parameter$
5.3) If the index picked in step 5.1 is an index of Tv:
   5.3.1) Store this index in variable Y
   5.3.2) Generate a random number $X \in \{0, 1, \ldots, n-1\}$, such that an element of P with $Argument$ X is not revealed
   5.3.3) Output $P'[X] = Y$, where Y is the $Base$ and X is the $Parameter$

Example 6.2.1. For $Q = \text{VMPC}_2(P) : Q[x] = P[P[P[P[x]] + 1] + 2]$:

Assume that $B = 8$, $R = 2$, $Q[6] = 9$ and that $P[6] = 8$ and $P[2] = 9$ are revealed.
There are two revealed elements of P which fit a general pattern of $Words$ of a $Statement$ in which $P[8]$ would be $Word$ 2: $P[6] = 8$ and $P[2] = 9$:

$Word$ 1	$Word$ 2	$Word$ 3	$Word$ 4
$P[6] = 8$	$P[8] = y$	$P[y+1] = 0$	$P[2] = 9$

$(Ta[8] = Ta[8] + Weight[2] = Ta[8] + 5)$

7 Example complexities of inverting the VMPC function

Complexity of inverting the VMPC function (Table 3) was approximated as an average number of times the Deducing Process (step 2) needs to be run by the inverting algorithm described in Section 6 until permutation P satisfies $Q = \mathrm{VMPC}_k(P)$.

Average numbers of elements of P which need to be assumed are given in Table 3 in brackets.

Complexities of inverting the VMPC function of the following levels were approximated:

$Q = \mathrm{VMPC}_1(P) : Q[x] = P[P[P[x]] + 1]$
$Q = \mathrm{VMPC}_2(P) : Q[x] = P[P[P[P[x]] + 1] + 2]$
$Q = \mathrm{VMPC}_3(P) : Q[x] = P[P[P[P[P[x]] + 1] + 2] + 3]$
$Q = \mathrm{VMPC}_4(P) : Q[x] = P[P[P[P[P[P[x]] + 1] + 2] + 3] + 4]$

Table 3. Example complexities of inverting the VMPC function

Function n	VMPC_1	VMPC_2	VMPC_3	VMPC_4
6	$2^{4,1}$ (2,3)	$2^{5,5}$ (3,1)	$2^{6,1}$ (3,3)	$2^{6,9}$ (3,8)
8	$2^{5,5}$ (2,7)	$2^{7,5}$ (3,4)	$2^{8,8}$ (4,0)	$2^{9,8}$ (4,4)
10	$2^{7,1}$ (3,0)	$2^{9,7}$ (4,0)	$2^{11,5}$ (4,7)	$2^{13,0}$ (5,2)
16	$2^{11,5}$ (3,8)	$2^{16,6}$ (5,4)	$2^{20,4}$ (6,6)	$2^{23,3}$ (7,5)
32	2^{24} (6,0)	2^{37} (9,1)	2^{47} (11,5)	2^{54} (13,4)
64	2^{53} (10,2)	2^{84} (16,2)	2^{108} (21,0)	2^{127} (24,9)
128	2^{117} (18)	2^{190} (30)	2^{245} (40)	2^{292} (47)
256	2^{260} (34)	2^{420} (57)	2^{550} (77)	2^{660} (92)

Example: For 1-level VMPC function applied on 256-element permutations: on average about 34 elements of P need to be assumed. Searching through half of the possible states of these elements requires about 2^{260} calls to the Deducing Process.

8 Design objectives for the VMPC Stream Cipher and its KSA

The Cipher should require no initial outputs to be discarded directly after running the KSA.

Probability that the Cipher's output will enter a short cycle should be negligibly low.

Output generated by the Cipher should be free from statistical biases.

Effort required to recover the internal state from the Cipher's output should be higher than a brute-force search of all possible 512-bit keys.

The KSA should resist related-key attacks and attacks against the scheme of using the Initialization Vector (IV).

The KSA should provide random-looking diffusion of changes of one byte of the key of size up to 512 bits onto the generated permutation and onto output generated by the Cipher.

9 Description of the VMPC Stream Cipher

The algorithm generates a stream of 8-bit values.

Variables:

P : 256-byte table storing a permutation initialized by the VMPC KSA
s : 8-bit variable initialized by the VMPC KSA
n : 8-bit variable
L : desired length of the keystream in bytes

Table 4. VMPC Stream Cipher

1. $n = 0$
2. Repeat steps 3-6 L times:
3. $s = P[(s + P[n])$ modulo 256]
4. Output $P[(P[P[s]] + 1)$ modulo 256]
5. $Temp = P[n]$
$P[n] = P[s]$
$P[s] = Temp$
6. $n = (n + 1)$ modulo 256

10 Description of the VMPC Key Scheduling Algorithm

The VMPC Key Scheduling Algorithm (KSA) transforms a cryptographic key and (optionally) an Initialization Vector into a 256-element permutation P and initializes variable s.

Variables as in Section 9, with:

c : fixed length of the cryptographic key in bytes, $16 \leq c \leq 64$
K : c-element table storing the cryptographic key
z : fixed length of the Initialization Vector in bytes, $16 \leq z \leq 64$
V : z-element table storing the Initialization Vector
m : 16-bit variable

Table 5. VMPC Key Scheduling Algorithm

1. $s = 0$
2. for n from 0 to 255: $P[n] = n$

3. for m from 0 to 767: execute steps 4-6:
   4. $n = m$ modulo 256
   5. $s = P[(s + P[n] + K[m \text{ modulo } c]) \text{ modulo } 256\ ]$
   6. $Temp = P[n]$
      $P[n] = P[s]$
      $P[s] = Temp$

7. If Initialization Vector is used: execute step 8:

8. for m from 0 to 767: execute steps 9-11:
   9. $n = m$ modulo 256
   10. $s = P[(s + P[n] + V[m \text{ modulo } z]) \text{ modulo } 256\ ]$
   11. $Temp = P[n]$
       $P[n] = P[s]$
       $P[s] = Temp$

11 VMPC Stream Cipher test vectors

Table 6 gives 16 test output-bytes generated by the VMPC Stream Cipher for a given 16-byte key (K) and a given 16-byte Initialization Vector (V):

Table 6. Test-output of the VMPC Stream Cipher

K; $c = 16$ [hex]	96, 61, 41, 0A, B7, 97, D8, A9, EB, 76, 7C, 21, 17, 2D, F6, C7
V; $z = 16$ [hex]	4B, 5C, 2F, 00, 3E, 67, F3, 95, 57, A8, D2, 6F, 3D, A2, B1, 55

Output-byte position [dec]	0	1	2	3	252	253	254	255
Output-byte value [hex]	A8	24	79	F5	B8	FC	66	A4

Output-byte position [dec]	1020	1021	1022	1023	102396	102397	102398	102399
Output-byte value [hex]	E0	56	40	A5	81	CA	49	9A

12 Performance of the VMPC Stream Cipher

Performance of a moderately optimized 32-bit assembler implementation of the VMPC Stream Cipher and its KSA, measured on an Intel Pentium 4, 2.66 GHz processor, is given in tables 7 and 8.

Table 7. Perfomance of the VMPC Stream Cipher

MBytes/s	MBits/s	cycles/byte
210	1680	12.7

Table 8. Perfomance of the VMPC KSA for 128-, 256- and 512-bit keys

keys/s	milliseconds/key
310 000	0.0032

13 Analysis of the VMPC Stream Cipher

13.1 Theoretical probability of equal consecutive outputs

Probability of two consecutive outputs being equal appears to be an important parameter for a cipher based, as VMPC is, on an internal permutation (P). A sole permutation is obviously distinguishable from a truly random stream as its values never repeat. The construction of a cipher based on an internal permutation should corrupt the regular structure of the permutation in such way as to force the outputs to repeat with a random-looking probability.

This section explains theoretically why the probability of consecutive outputs generated by the VMPC Stream Cipher being equal is the same as we would expect from a random oracle, i.e. that $Prob(out[x] = out[x+1]) = 2^{-N}$, where N is the word-size of the Cipher in bits; the standard value of N is 8.

To compute the probability, two scenarios need to be considered: (1) - there is no swap in step 5 (Table 4) and (2) - there is a swap in step 5.

In (1): $Prob$(no-swap)=$Prob(s[x] = n[x]) = 2^{-N}$

As a result of (1) permutation P will have the same arrangement of elements in steps x and $x+1$. This implies a distinction into two sub-scenarios - (1a) where $s[x] = s[x+1]$ and (1b) where $s[x] \neq s[x+1]$, which directly affects whether $out[x] = out[x+1]$ or $out[x] \neq out[x+1]$.

In (1a): $Prob(s[x] = s[x+1]) = 2^{-N}$
and (1aR): $Prob(out[x] = out[x+1]) = 1$

In (1b): $Prob(s[x] \neq s[x+1]) = 1 - 2^{-N}$
and (1bR): $Prob(out[x] = out[x+1]) = 0$

In (2): $Prob$(swap)=$Prob(s[x] \neq n[x]) = 1 - 2^{-N}$
and (2R): $Prob(out[x] = out[x+1]) = 2^{-N}$,
regardless of the relation between $s[x]$ and $s[x+1]$ because P (and $\text{VMPC}_1(P)$) in steps x and $x+1$ are different permutations. This probability was also confirmed experimentally.

By combining the probabilities in scenarios (1)(1a)(1aR), (1)(1b)(1bR) and (2)(2R) we get:

$$Prob(out[x] = out[x+1]) =$$

$$= 2^{-N} \times 2^{-N} \times 1 + 2^{-N} \times (1 - 2^{-N}) \times 0 + (1 - 2^{-N}) \times 2^{-N} = 2^{-N}$$

which ends the proof.

13.2 Recovering the Cipher's internal state

A method analogous to the Forward Tracking Algorithm proposed by Mister and Tavares in [3] was applied to break the VMPC Stream Cipher. Following this approach an average of over 2^{900} computational operations is estimated to be required to recover the Cipher's internal state from its output.

13.3 Digraph and trigraph probabilities

Frequencies of occurrence of each of the possible 2^{16} pairs of consecutive output values ($out[x]$, $out[x+1]$) were measured in a stream of $2^{40.1}$ output bytes. None of the measured frequencies showed a statistically significant deviation from its expected value of 1 / 65536.

Frequencies of occurrence of each of the possible 2^{24} triplets of two consecutive output values and the n variable ($out[x]$, $out[x+1]$, n) were measured in a stream of $2^{41.85}$ output bytes. None of the measured frequencies showed a statistically significant deviation from its expected value of 1 / 16777216.

Frequencies of occurrence of each of the possible 2^{24} trigraphs of consecutive output values ($out[x]$, $out[x+1]$, $out[x+2]$) were measured in a stream of $2^{41.6}$ output bytes. None of the measured frequencies showed a statistically significant deviation from its expected value of 1 / 16777216.

13.4 Single-output probabilities

Frequencies of occurrence of each of the possible 2^{8} output values ($out[x]$) were measured in a stream of $2^{41.85}$ output bytes. None of the measured frequencies showed a statistically significant deviation from its expected value of 1 / 256.

Frequencies of occurrence of each of the possible 2^{16} configurations of an output value and the n variable ($out[x]$, n) were measured in a stream of $2^{39.4}$ output bytes. None of the measured frequencies showed a statistically significant deviation from its expected value of 1 / 65536.

13.5 First-outputs probabilities

Frequencies of occurrence of each of the possible 2^{8} values on each of the first 256 byte-positions of the keystream generated directly after running the KSA were measured in a sample of $2^{40.3}$ bytes of the Cipher's output for $2^{32.3}$ different keys. None of the measured frequencies showed a statistically significant deviation from its expected value of 1 / 256. [In [6] Mantin and Shamir show that the second output of RC4 takes on value 0 with probability 1 / 128 rather than 1 / 256.]

13.6 Short cycles

Probability of entering a cycle not longer than X Following Knuth's [1], probability of entering a cycle not longer than X for an n-element random permutation is X/n.

To compare cycle lengths in the output of the VMPC Stream Cipher to cycle lengths in a random permutation, the Cipher was scaled down to use M-element permutations for $M \in \{4, 5, \ldots, 10\}$.

The total number of $M! \times M^2$ possible internal states of the Cipher is determined by all possible configurations of permutation P and variables s and n.

The observed cycle lengths, listed in the Appendix, do not show an appreciable difference from a model of cycles in a random $(M! \times M^2)$-element permutation. Probability of entering a cycle not longer than X by the VMPC Stream Cipher is conjectured from this to be approximately $X/(256! \times 256^2) \cong X/2^{1700}$. An example estimate is that probability that the Cipher's output will enter a cycle not longer than 2^{850} is about 1 / 2^{850}.

Finney states In [10] Finney defined a theoretical class of internal states of RC4 which produce a short cycle of length 65280 by swapping $P[n] = 1$ in each step (the KSA of RC4 prevents the cipher's internal state from entering this class). The class is diagnosed by $n + 1 = s$ and $P[n + 1] = 1$.

Such phenomenon is possible because step $s = s+P[n]$ of the state-transformation function of RC4 retains the linear structure of $P[n]$ in variable s ($P[n]$, after the increment of n, is always equal 1).
The VMPC Stream Cipher uses an additional table-lookup ($s = P[s + P[n]]$), which, assuming that P was properly initialized, corrupts a possible linear structure of $P[n]$ (or s) and prevents situations analogous to Finney states from occurring.

13.7 Binary derivatives of bit output sequences

This family of tests was inspired by Golic's [8], where the author describes a statistical bias in the second binary derivative of the least significant bit output sequence of RC4. The author finds that the bias allows the attacker to distinguish RC4 output from a truly random source using about $64^N/225$ outputs, where N is the cipher's word-size in bits (e.g. for $N = 8$ the required length is about 2^{40}) [1].

---

[1] Authors of [5] consider this estimate somewhat optimistic and suggest that the required keystream length for $N = 8$ is about $2^{44.7}$

Output generated by the VMPC Stream Cipher showed no bias in this family of tests. The following objectives were taken in testing VMPC here:

$N = 7$ word-size was chosen to make the tested algorithm as close to the real 8-bit one as possible while significantly decreasing the output-sequence length required to reveal the bias for RC4 (for 7-bit RC4 - about $2^{34.2}$ outputs according to the original estimates in [8]). First, second and third binary derivatives of all 7 bits output sequences were tested (21 frequencies of ($out_k[x] + out_k[x+A]$=1) were measured for $k \in \{0, 1, \ldots, 6\}$, $A \in \{1, 2, 3\}$, where $out_k[x]$ denotes k-th bit of x-th output word).

In a sequence of $2^{44.8}$ (about $10^{13.5}$) VMPC outputs tested according to this approach none of the measured frequencies showed a statistically significant deviation from its expected value of 0.5.

13.8 Equal neighboring outputs probabilities

Frequencies of occurrence of situations where there occurs a given number (0, 1, 2, 3, 4, 5 and over 5) of direct (generated consecutively) and indirect (separated by one output byte) equal neighboring outputs in the consecutive 256-byte sub-streams of the Cipher's output and the average total number of direct and indirect equal neighboring outputs – showed no statistically significant deviation from their expected values in a sample of $2^{43.1}$ bytes of the Cipher's output.

13.9 Statistical tests on the Cipher's output

Keystreams generated by the VMPC Stream Cipher were tested by two popular batteries of statistical tests – the DIEHARD battery [11] and the NIST statistical tests suite [12]. No bias was found by any of the 15 tests included in the DIEHARD battery or by any of the 16 tests from the NIST suite.

14 Analysis of the VMPC Key Scheduling Algorithm

The VMPC Key Scheduling Algorithm was tested for diffusion of changes of the cryptographic key onto the generated permutation and onto the Cipher's output. A change of one byte of the cryptographic key of size 128, 256 and 512 bits appears to cause a random-looking change in the generated permutation and in the Cipher's output.

The KSA was designed to provide the diffusion without the use of the Initialization Vector and the tests were run without the IV. The Initialization Vector would obviously mix the generated permutation further, which would improve the diffusion effect.

14.1 Given numbers of equal permutation elements probabilities

Frequencies of occurrence of situations where in two permutations, generated from keys differing in one byte, there occurs a given number (0, 1, 2, 3, 4, 5) of equal elements in the corresponding positions and the average number of equal elements in the corresponding positions – showed no statistically significant deviation from their expected values in samples of $2^{33.2}$ pairs of 128-, 256- and 512-bit keys.

14.2 Given numbers of equal Cipher's outputs probabilities

Frequencies of occurrence of situations where in two 256-byte streams generated by the VMPC Stream Cipher directly after running the KSA for keys differing in one byte, there occurs a given number (0, 1, 2, 3, 4, 5) of equal values in the corresponding byte-positions and the average number of equal values in the corresponding byte-positions – showed no statistically significant deviation from their expected values in samples of $2^{33.2}$ pairs of 128-, 256- and 512-bit keys.

14.3 Equal corresponding permutation elements probabilities

Frequencies of occurrence of situations where the elements in the corresponding positions of permutations generated from keys differing in one byte are equal (for each of the 256 positions) – showed no statistically significant deviation from their expected value in samples of $2^{33.2}$ pairs of 128-, 256- and 512-bit keys.

15 Conclusions

A simple one-way function together with a description of the most efficient method of inverting it found have been presented. An open problem is whether the simplicity of the function helps make a hypothetical attempt to prove a lower bound on the complexity of inverting it worth undertaking.

A proposed stream cipher which employs the function was given together with some analysis of the cipher's cryptographic strength, statistical properties of the cipher's output and statistical properties of the cipher's Key Scheduling Algorithm.

The analyses performed so far did not reveal any weakness in the design and indicated that the cipher has a number of security advantages over the alleged RC4 keystream generator while retaining most of its speed and simplicity.

Appendix: Cycle lengths observed in the output of the VMPC Stream Cipher

The observed cycle lengths in the output of the scaled down variants of the Cipher for $M \in \{4, 5, \ldots, 10\}$ are listed in Table 9. M denotes the number of elements in the P permutation. All addition operations performed by the Cipher here are additions modulo M.

Table 9. VMPC Stream Cipher cycle lengths

M	Cycle lengths
4	200, 88, 40, 36, 12, 8
5	1 860, 640, 295, 110, 45, 25, 20, 5
6	15 510, 5 580, 2 508, 936, 516, 510, 252, 90, 12, 6
7	215 089, 23 821, 3 990, 2 485, 1 015, 392, 70, 56, 28, 14
8	2 401 728, 79 504, 53 512, 42 120, 2 136, 1 032, 288, 96, 24, 16 (2 different cycles of length 16 possible), 8
9	20 355 471, 2 908 098, 2 728 890, 1 359 855, 949 725, 609 174, 299 592, 125 091, 27 306, 13 068, 6 219, 5 067, 2 853, 2 538, 180, 90, 18 (3 different cycles of length 18 possible), 9
10	113 748 840, 99 425 590, 75 813 290, 37 178 940, 20 169 740, 9 955 030, 3 239 140, 2 349 150, 572 500, 363 830, 45 520, 8 730, 7 520, 700, 390, 370, 40 (17 different cycles of length 40 possible), 20, 10 (2 different cycles of length 10 possible)

References

1. Donald E. Knuth: The Art of Computer Programming, vol. 1. *Fundamental Algorithms* Third Edition. Addison Wesley Longman, 1997.
2. Donald E. Knuth: The Art of Computer Programming, vol. 2. *Seminumerical Algorithms* Third Edition. Addison Wesley Longman, 1998.
3. Serge Mister, Stafford E. Tavares: Cryptanalysis of RC4-like Ciphers. Proceedings of SAC 1998, LNCS, vol. 1556, Springer-Verlag, 1999.
4. 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.
5. Scott R. Fluhrer, David A. McGrew: Statistical Analysis of the Alleged RC4 Keystream Generator. Proceedings of FSE 2000, LNCS, vol. 1978, Springer-Verlag, 2001.
6. Itsik Mantin, Adi Shamir: A Practical Attack on Broadcast RC4. Proceedings of FSE 2001, LNCS, vol. 2355, Springer-Verlag, 2002.
7. Scott Fluhrer, Itsik Mantin, Adi Shamir: Weaknesses in the Key Scheduling Algorithm of RC4. Proceedings of SAC 2001, LNCS, vol. 2259, Springer-Verlag 2001.
8. Jovan Dj. Golic: Linear Statistical Weakness of Alleged RC4 Keystream Generator. Proceedings of EUROCRYPT 1997, LNCS, vol. 1233, Springer-Verlag 1997.
9. Alexander L. Grosul, Dan S. Wallach: A Related-Key Cryptanalysis of RC4. Technical Report TR-00-358, Department of Computer Science, Rice University, 2000.
10. Hal Finney: An RC4 Cycle That Can't Happen. Post in sci.crypt, 1994.
11. George Marsaglia: DIEHARD battery of statistical tests with documentation. http://stat.fsu.edu/~geo/diehard.html
12. NIST statistical tests suite with documentation. http://csrc.nist.gov/rng