VMPC-MAC: A Stream Cipher Based Authenticated Encryption Scheme

Bartosz Zoltak

http://www.vmpcfunction.com
bzoltak@vmpcfunction.com

Abstract. A stream cipher based algorithm for computing Message Authentication Codes is described. The algorithm employs the internal state of the underlying cipher to minimize the required additional-to-encryption computational effort and maintain general simplicity of the design. The scheme appears to provide proper statistical properties, a comfortable level of resistance against forgery attacks in a chosen ciphertext attack model and high efficiency in software implementations.

Keywords: Authenticated Encryption, MAC, Stream Cipher, VMPC

1 Introduction

In the past few years the interest in message authentication algorithms has been concentrated mostly on modes of operation of block ciphers. Examples of some recent designs include OCB [4], OMAC [7], XCBC [6], EAX [8], CWC [9]. Parallely a growing interest in stream cipher design can be observed, however along with a relative shortage of dedicated message authentication schemes. Regarding two recent proposals – Helix and Sober-128 stream ciphers with built-in MAC functionality – a powerful attack against the MAC algorithm of Sober-128 [10] and two weaknesses of Helix [12] were presented at FSE'04.

This paper gives a proposition of a simple and software-efficient algorithm for computing Message Authentication Codes for the presented at FSE'04 VMPC Stream Cipher [13].

The proposed scheme was designed to minimize the computational cost of the additional-to-encryption MAC-related operations by employing some data of the internal-state of the underlying cipher. This approach allowed to maintain simplicity of the design and achieve good performance in software implementations. The scheme appears to ensure a proper diffusion effect, remain practically secure in the chosen ciphertext attack model and enable to increase the security level, in analogous attacks, by adjusting a parameter d.

Section 2 outlines general characteristics of the scheme, Section 3 describes the employed primitives – the VMPC Stream Cipher and its Key Scheduling Algorithm, Section 4 defines a d-level VMPC-MAC Scheme, sections 5-7 discuss the security of the scheme and sections 8-9 quote software performance numbers and test-vectors.

2 General features of the VMPC-MAC Scheme

The scheme is based on an internal state T transformed along with the encryption in a manner determined by the ciphertext data and the internal state of the cipher. This stage is further referred to as the Encryption Phase.

After the Encryption Phase, T undergoes a transformation in the Post-Processing Phase, which is to generate an undistinguishable from random diffusion effect, discussed in Section 5.

In the third and final MAC Generation Phase T is transformed into a desired-length Message Authentication Code by a compression function. This phase is considered to provide a security margin to the scheme. The statistical tests (Section 5) and a collision-finding attack (Section 7) indicate that the scheme remains secure also without this phase. One practical benefit of the MAC Generation Phase is the fact that it allows to comfortably adjust the size of the MAC tag, here proposed at 160 bits.

To minimize the number of employed primitives, the compression function in the MAC Generation Phase is derived from the Key Scheduling Algorithm of the underlying stream cipher and from the cipher itself. The properties of the cipher and its KSA will be outlined in section 3 to elucidate the choice.

3 Employed primitives

This section outlines the VMPC Stream Cipher and its Key Scheduling Algorithm. VMPC was presented at FSE'04 as a simple and software-efficient stream cipher with a specified KSA and Initialization Vector management routine.

3.1 VMPC Stream Cipher

The VMPC Stream Cipher generates a stream of 8-bit words, as specified in Table 1.

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
$+$: addition modulo 256

Table 1. VMPC Stream Cipher

```
1. $n = 0$

2. Repeat steps 3-6 $L$ times:
   3. $s = P[s + P[n]]$
   4. Output $P[P[P[s]] + 1]$
   5. $Temp = P[n]$
      $P[n] = P[s]$
      $P[s] = Temp$
   6. $n = n + 1$
```

According to [13] the cipher generates keystreams undistinguishable from a truly random source and has a number of statistical advantages over the popular RC4 keystream generator. The risk of the cipher's output falling into a short cycle was concluded to be negligibly low (an example estimate says that probability that the output will enter a cycle not longer than 2^{850} is about 2^{-850}). The average computational complexity of recovering the cipher's internal state from its output is estimated at about 2^{900} operations. An RC4-specific problem of the so-called Finney states (which theoretically can make the keystream fall into a cycle of length 65280, for the typical 8-bit version of the cipher) is automatically avoided by VMPC. A number of tests showed no bias in the statistical characteristics of the cipher's output, including many of the tests which revealed the weaknesses of RC4. The cipher can also be considered efficient in software implementations – it is claimed to perform at a rate of about 12.7 clock-cycles per byte on a Pentium 4 processor.

3.2 VMPC Key Scheduling Algorithm

The VMPC Key Scheduling Algorithm transforms a cryptographic key K and – optionally – an Initialization Vector V into the 256-element permutation P and initializes variable s.

Variables as for VMPC Stream Cipher, 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
$+$: addition modulo 256

Table 2. 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. $s = P[s + P[n] + K[m \text{ modulo } c]]$
   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. $s = P[s + P[n] + V[m \text{ modulo } z]]$
   11. $Temp = P[n]$
       $P[n] = P[s]$
       $P[s] = Temp$
```

According to [13] the algorithm provides an undistinguishable from random diffusion of changes of one bit or byte of the key K of size up to 64 bytes onto the generated P permutation and onto output generated by the cipher.

The diffusion effect of the KSA has consequences for the design of the compression function employed in the MAC Generation Phase. Section 3.3 outlines the analyses of this aspect of the KSA following [13].

3.3 Diffusion effect of the VMPC KSA

The algorithm was tested for diffusion of changes of the key onto the generated permutation and onto the output of the VMPC Stream Cipher. A change of one bit or byte of the key of size 128, 256 and 512 bits appears to cause an undistinguishable from random change in the generated permutation and in the cipher's output.

In other words the relations between two permutations or two keystreams generated from keys K and K' differing in one bit or byte are undistinguishable from relations between, respectively, two random permutations or two random data-streams.

The KSA was designed to provide the diffusion without the use of the Initialization Vector and the tests were run without the IV (only steps 1-6 were applied). The Initialization Vector would obviously mix the generated permutation further, which would improve the diffusion effect. The aim of this approach is to ensure that the diffusion effect is provided with a safety margin over the test results.

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.

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.

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.

4 d-level VMPC-MAC Scheme

The d-level VMPC-MAC Scheme is described in Table 3. Steps E1-E12 define the Encryption Phase, steps P1-P10 the Post-Processing Phase and steps M1-M5 the MAC Generation Phase.

Variables:

$Pt[m]$: m-th 8-bit word of plaintext
$Ct[m]$: m-th 8-bit word of ciphertext

$x_1, x_2, \ldots, x_d$: 8-bit variables
T : $(8 \times d)$-element table of 8-bit words. Let $T[n]$ denote n-th element of T

n, m, g, R : temporary integer variables
Let V and z be defined as in Section 3.2 and L as in Section 3.1
Let $+$ denote addition modulo 256

Table 3. d-level VMPC-MAC Scheme

```
E1. Run the VMPC Key Scheduling Algorithm (Table 2)
E2. Set $T$, $x_1, \ldots, x_d, n, m, g$ to 0

E3. $s = P[s\ +\ P[n]]$
E4. $Ct[m] = Pt[m]$ xor $P[P[P[s]]\ +\ 1]$

E5. For $i$ from $d$ down to 2 : $x_i = P[x_i\ +\ x_{i-1}]$
E6. $x_1 = P[x_1\ +\ s\ +\ Ct[m]]$
E7. For $i$ from 1 to $d$ : xor $T[g + i - 1]$ with $x_i$

E8. $Temp = P[n];\ \ P[n] = P[s];\ \ P[s] = Temp$
E9. $g = (g + d)$ modulo $(8 \times d)$
E10. $n = n\ +\ 1$
E11. Increment $m$
E12. If $m < Length\ of\ Plaintext$ : Go to step E3
--------------------------------------------------
P1. Set $R$ to 1
P2. $s = P[s\ +\ P[n]]$

P3. For $i$ from $d$ down to 2 : $x_i = P[x_i\ +\ x_{i-1} + \text{R}]$
P4. $x_1 = P[x_1\ +\ s\ +\ R]$
P5. For $i$ from 1 to $d$ : xor $T[g + i - 1]$ with $x_i$

P6. $Temp = P[n];\ \ P[n] = P[s];\ \ P[s] = Temp$
P7. $g = (g + d)$ modulo $(8 \times d)$
P8. $n = n\ +\ 1$
P9. $R = R + 1$
P10. If $R \leq 24$ : Go to step P2
--------------------------------------------------
M1. Store table $T$ in table $V$
M2. Set $z$ to $(8 \times d)$
M3. Execute step 8 of the VMPC Key Scheduling Algorithm (Table 2)
M4. Set $L$ to 20
M5. Execute steps 1 and 2 of the VMPC Stream Cipher (Table 1) and
      save the 20 generated outputs as the 160-bit $MAC$
```

5 Statistical properties of the scheme

The scheme was tested for diffusion of changes of the input data onto the output data. The input includes the plaintext message Pt, the key K and the Initialization Vector V. The actual output is the keystream generated by the VMPC Stream Cipher in step M5, however to obtain a more rigorous measure of the diffusion effect, we will consider the T table generated only in steps E1-E12 and P1-P10 as the scheme's output for the purpose of the tests. Steps M1-M5 ensure the diffusion properties of the VMPC KSA, discussed in Section 3.3, which provides a safety margin to the diffusion effect of the complete MAC scheme.

A proper diffusion effect (a random-like relation between outputs generated from different inputs) is considered here to occur, when for T_r being an $(8 \times d)$-element table of 8-bit words derived from a truly random source, $T_1 = MAC(Pt_1, K_1, V_1)$ and $T_2 = MAC(Pt_2, K_2, V_2)$, where $Pt_1 \neq Pt_2$ or $K_1 \neq K_2$ or $V_1 \neq V_2$ (the or is not exclusive and the differences can be limited to only one bit-position), probability of determining which of the three values T_r, T_1 or T_2 was generated by the truly random source is 1/3.

The diffusion of changes of the key K or the Initialization Vector V onto the T table can be assumed random-like as a direct consequence of the diffusion properties of the VMPC Key Scheduling Algorithm discussed in Section 3.3.

The question remains whether the proper diffusion occurs when $K_1 = K_2$, $V_1 = V_2$ and $Pt_1 \neq Pt_2$. To evaluate this, statistical tests were run for: (a) Pt_1 and Pt_2 differing only in the last byte and (b) for Pt_1 and Pt_2, where Pt_2 is derived by copying Pt_1 to Pt_2 and appending one byte to Pt_2.

5.1 Pt_1 and Pt_2 differing in the last byte

In this test the length of Pt_1 and Pt_2 was 8 bytes and the last byte of Pt_2 (the one which is input to the scheme in the latest time) was taking on all the possible 255 values except for the value of the last byte of Pt_1. For each value of Pt_2 two tables $T_1 = MAC(Pt_1, K_0, V_0)$ and $T_2 = MAC(Pt_2, K_0, V_0)$ were compared. The values of Pt_1, K_0 and V_0 were changed after each set of the 255 steps. The test was run for $d = 4$ and for $2^{37.2}$ pairs of Pt_1 and Pt_2. Frequencies of occurrence of situations where $T_1[x] = T_2[x]$ for $x \in \{0..31\}$ showed no statistically significant deviations from their expected value of 1/256.

5.2 Pt_2 with an appended last byte

In this test the length of Pt_1 was 8 bytes and the length of Pt_2 was 9 bytes. First 8 bytes of Pt_1 and Pt_2 were equal and the last byte of Pt_2 (the one which is input to the scheme in the latest time) was taking on all the possible 256 values. For each value of Pt_2 two tables $T_1 = MAC(Pt_1, K_0, V_0)$ and $T_2 = MAC(Pt_2, K_0, V_0)$ were compared. The values of Pt_1, K_0 and V_0 were changed after each set of the 256 steps. The test was run for $d = 4$ and for $2^{37.2}$ pairs of Pt_1 and Pt_2. Frequencies of occurrence of situations where $T_1[x] = T_2[x]$ for $x \in \{0..31\}$ showed no statistically significant deviations from their expected value of 1/256.

6 A proof of security

A desirable feature of an authenticated encryption scheme is a proof of security. Proofs are usually obtained for MAC algorithms based on primitives like block ciphers or hash functions. Under the assumption of the desirable properties of the primitives (e.g. that they are PRPs or PRFs), it is possible to build a proof of the adopted notion of security of the MAC algorithm.

Here the MAC algorithm is integrated with the underlying stream cipher, it employs the cipher's internal state to update the internal state of the MAC-scheme. In our view it is rather unlikely that a formal proof of security of a similar scheme can be constructed. Such proof would probably be comparable to a proof of security of a primitive itself, rather than to a proof of security of a given mode of operation of a primitive. From the lack of proofs for the primitives (like stream and block ciphers or hash functions) we accept that building a proof of security of the described scheme could be practically unachievable.

7 Fastest chosen ciphertext attack found

The most efficient chosen ciphertext attack found against the VMPC-MAC Scheme obtains collisions in the T table (and consequently in the MAC tags) with a success probability of 2^{-144}. If a higher level of resistance was desired, in an analogous attack model, it could be obtained by increasing the value of the d parameter.

The attack assumes that the adversary has full passive and active access to the ciphertext and plaintext and can use an unlimited number of verification queries for the new message. The purpose of the adversary is to introduce a new valid ciphertext Ct_2 which was not MACed through an authentication query but which is deemed valid in a verification query. The most efficient attack found assumes that the adversary starts with a valid message Ct_1 (which was intercepted or obtained through an authentication query), copies it to Ct_2 and changes Ct_2 it in such way as to make Ct_2 generate the same MAC as Ct_1 did.

The attack begins with a change of one word of the ciphertext – $Ct_2[m]$ and proceeds with changing 9 other words in such way as to obtain
MAC(Ct_2)=MAC(Ct_1).

The attack is illustrated for $d = 4$.
Let $x_w(m)$ denote the value of x_w in iteration m; $w \in \{1, 2, 3, 4\}$
Let n be a function defined as $n = (m \text{ modulo } 8) \times 4$
Let $(+)$ denote addition modulo 32

A change of $Ct_2[m]$ unconditionally causes a change of $x_1(m)$ in step E6, since P is a permutation.

Because $x_1(m)$ and only $x_1(m)$ directly updates $x_2(m+1)$ and indirectly updates $x_3(m+2)$ and $x_4(m+3)$, the variables $x_2(m+1)$, $x_3(m+2)$ and $x_4(m+3)$ will be unconditionally changed too.
The following elements of table T will be updated and unconditionally changed by those variables: $T[n]$ changed by $x_1(m)$, $T[n(+)5]$ changed by $x_2(m+1)$, $T[n(+)10]$ changed by $x_3(m+2)$ and $T[n(+)15]$ changed by $x_4(m+3)$.

The most efficient method of reverting these changes found requires the following changes of the ciphertext:

1. Change $Ct_2[m+1]$ in such way as to make $x_4(m+4)$ return to its original value. The unavoidable cost of this is a change of $x_1(m+1)$, $x_2(m+2)$ and $x_3(m+3)$. [1] [$x_3(m+3)$ must be changed in such way as to make $x_4(m+4) = (x_4(m+3) + x_3(m+3))$ modulo 256 return to its original value [2]].
As a result $T[n(+)4]$ is changed by $x_1(m+1)$, $T[n(+)9]$ is changed by $x_2(m+2)$ and $T[n(+)14]$ is changed by $x_3(m+3)$. $T[n(+)19]$ remains unchanged because the change of $x_4(m+4)$ was reverted.

2. Change $Ct_2[m+2]$ in such way as to make $x_3(m+4)$ return to its original value. The unavoidable cost of this is a change of $x_1(m+2)$ and $x_2(m+3)$.
As a result $T[n(+)8]$ is changed by $x_1(m+2)$ and $T[n(+)13]$ is changed by $x_2(m+3)$. $T[n(+)18]$ remains unchanged because the change of $x_3(m+4)$ was reverted.

3. Change $Ct_2[m+3]$ in such way as to make $x_2(m+4)$ return to its original value. The unavoidable cost of this is a change of $x_1(m+3)$.
As a result $T[n(+)12]$ is changed by $x_1(m+3)$. $T[n(+)17]$ remains unchanged because the change of $x_2(m+4)$ was reverted.

4. Change $Ct_2[m+4]$ in such way as to make $x_1(m+4)$ return to its original value. As a result $T[n(+)16]$ remains unchanged.

[1] The algorithm can be varied into making some of the variables (e.g. $x_2(m+2)$) remain unchanged, which yields an apparent improvement, however further analysis shows that this actually leads to higher complexity of the complete attack.

[2] The approach by which the first variable to return to its original value is x_4, rather than e.g. x_1 or x_2, in further analysis shows to lead to much lower complexities of the complete attack.

At this moment the avalanche of changes of the elements of T, resulting from a change of $Ct_2[m]$, was stopped by reverting the changes of x_1, x_2, x_3, x_4 in the earliest possible iteration $m + 4$. The cost of this is an unavoidable change of 10 elements of the T table ($T[n$, $n(+)4$, $n(+)5$, $n(+)8$, $n(+)9$, $n(+)10$, $n(+)12$, $n(+)13$, $n(+)14$, $n(+)15]$).

To complete a successful forgery, the adversary needs to revert the changes of these elements of T. Operations analogous to steps 1-4 are needed to refrain x_1, x_2, x_3, x_4 from causing more damage to T and an additional requirement – to revert the already caused changes to T – needs to be satisfied. The most efficient method found achieves that in the following steps 5-9:

5. Change $Ct_2[m + 8]$ in such way as to change $x_1(m + 8)$ in such way as to revert the change of $T[n]$, make $x_2(m + 9)$ change in such way as to revert the change of $T[n(+)5]$, $x_3(m + 10)$ change in such way as to revert the change of $T[n(+)10]$ and $x_4(m + 11)$ change in such way as to revert the change of $T[n(+)15]$.

6. Change $Ct_2[m + 9]$ in such way as to make $x_4(m + 12)$ return to its original value, $x_1(m + 9)$ change in such way as to revert the change of $T[n(+)4]$, $x_2(m+10)$ change in such way as to revert the change of $T[n(+)9]$ and $x_3(m+11)$ change in such way as to revert the change of $T[n(+)14]$. $T[n(+)19]$ remains unchanged because the change of $x_4(m + 12)$ was reverted.

7. Change $Ct_2[m + 10]$ in such way as to make $x_3(m + 12)$ return to its original value, $x_1(m + 10)$ change in such way as to revert the change of $T[n(+)8]$ and $x_2(m + 11)$ change in such way as to revert the change of $T[n(+)13]$. $T[n(+)18]$ remains unchanged because the change of $x_3(m + 12)$ was reverted.

8. Change $Ct_2[m + 11]$ in such way as to make $x_2(m + 12)$ return to its original value and $x_1(m + 11)$ change in such way as to revert the change of $T[n(+)12]$. $T[n(+)17]$ remains unchanged because the change of $x_2(m + 12)$ was reverted.

9. Change $Ct_2[m + 12]$ in such way as to make $x_1(m + 12)$ return to its original value. As a result $T[n(+)16]$ remains unchanged.

At this point MAC(Ct_2)=MAC(Ct_1).

The success probability of the described attack is determined by the total number of changes to variables x_1, x_2, x_3, x_4 and $T[0, 1, \ldots, 31]$, which need to be reverted. Steps 1-9 determine this probability, for the assumed $d = 4$, to 256^{-18} $= 2^{-144}$.

An analogous attack for $d = 5$ would yield a success probability of 256^{-25} =

2^{-200} (which would also imply an increase in the size of the MAC tag to 25 or more bytes), however the implementation of the scheme would not be as natural as for $d = 4$ (while still easily achievable) which, given the fact that 2^{-144} can be considered a comfortable level of security, encourages to propose $d = 4$ as sufficient for possible practical applications of the VMPC-MAC Scheme.

8 Performance of the VMPC-MAC Scheme

Performance of a moderately optimized 32-bit assembler implementation of the VMPC-MAC Scheme measured on an Intel Pentium 4, 2.66 GHz processor, is given in Table 4.

Table 4. Performance rates of the VMPC-MAC Scheme

MBytes/s	MBits/s	cycles/byte
91	728	29

9 Test values of the VMPC-MAC Scheme

Table 5 gives an example 20-byte tag generated by the VMPC-MAC Scheme for a given 16-byte key K, a given 16-byte Initialization Vector V and for a 256-byte plaintext $Message$ consisting of consecutive numbers from 0 to 255 ($Message[x] = x$ for $x \in \{0, 1, \ldots, 255\}$).

Table 5. Test vectors of the VMPC-MAC Scheme

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
$Message$ [dec]	0, 1, 2, 3, ..., 253, 254, 255
MAC [hex]	9B, DA, 16, E2, AD, 0E, 28, 47, 74, A3, AC, BC, 88, 35, A8, 32, 6C, 11, FA, AD

10 Conclusions

An algorithm for computing Message Authentication Codes for a stream cipher, based on the VMPC Stream Cipher and its Key Scheduling Algorithm, was described. Some data required to update the internal state of the scheme is derived from the internal state of the underlying cipher, which allowed the design to remain simple and achieve good performance in software implementations. The diffusion properties of the KSA were employed to provide a safety-margin over the test results to the diffusion effect of the MAC algorithm.

From the analyses performed so far, VMPC-MAC appears to provide a proper diffusion effect, remain practically secure in the fastest found chosen ciphertext attack and allow to increase the security level, in an analogous attack model, by increasing a parameter d.

References

1. Federal Information Processing Standards Publication 198:
The Keyed-Hash Message Authentication Code (HMAC), 2002 http://csrc.nist.gov/publications/fips/fips198/fips-198a.pdf
2. Mihir Bellare, Ran Canetti, Hugo Krawczyk: Message Authentication using Hash Functions the HMAC Construction, CryptoBytes, Vol 2, No. 1, RSA Laboratories, 1996
3. Mihir Bellare, Chanathip Namprempre: Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm, Proceedings of ASIACRYPT 2000, LNCS vol. 1976 Springer-Verlag, 2000
4. Phillip Rogaway, Mihir Bellare, John Black, Ted Krovetz: OCB: A Block-Cipher Mode of Operation for Efficient Authenticated Encryption (2001), Eighth ACM Conference on Computer and Communications Security (CCS-8) (August 2001), ACM Press.
5. Mihir Bellare, Roch Guerin, Philip Rogaway: XOR MACs: New Methods for Message Authentication Using Finite Pseudorandom Functions, Proceedings of CRYPTO 1995, LNCS vol. 963, Springer-Verlag, 1995.
6. V. Gligor, P. Donescu: Fast Encryption and Authentication: XCBC Encryption and XECB Authentication Modes, 2nd NIST Workshop on AES Modes of Operation, Santa Barbara, USA, 2001.
7. T. Iwata, K. Kurosawa: OMAC: One-key CBC MAC, Proceedings of Fast Software Encryption 2003, LNCS vol. 2887, Springer-Verlag 2003.
8. Mihir Bellare, Philip Rogaway, David Wagner: The EAX Mode of Operation Proceedings of Fast Software Encryption 2004, LNCS vol 3017, Springer-Verlag 2004.
9. Tadayoshi Kohno, John Viega, Doug Whiting: CWC: A High-Performance Conventional Authenticated Encryption Mode, Proceedings of Fast Software Encryption 2004, LNCS vol 3017, Springer-Verlag 2004.
10. Dai Watanabe, Soichi Furuya: A MAC forgery attack on SOBER-128, Proceedings of Fast Software Encryption 2004, LNCS vol 3017, Springer-Verlag 2004.
11. Niels Ferguson, Doug Whiting, Bruce Schneier, John Kelsey, Stefan Lucks, Tadayoshi Kohno: Helix: Fast Encryption and Authentication in a Single Cryptographic Primitive, Proceedings of Fast Software Encryption 2003, LNCS vol. 2887, Springer-Verlag 2003.
12. Fredric Muller: Differential Attacks Against the Helix Stream Cipher, Proceedings of Fast Software Encryption 2004, LNCS vol 3017, Springer-Verlag 2004.
13. Bartosz Zoltak: VMPC One-Way Function and Stream Cipher, Proceedings of Fast Software Encryption 2004, LNCS vol 3017, Springer-Verlag 2004.
14. NESSIE consortium: Performance of Optimized Implementations of the NESSIE Primitives, 2003 www.cryptonessie.org

15. 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.
16. Scott R. Fluhrer, David A. McGrew: Statistical Analysis of the Alleged RC4 Keystream Generator. Proceedings of Fast Software Encryption 2000, LNCS, vol. 1978, Springer-Verlag, 2001.
17. Itsik Mantin, Adi Shamir: A Practical Attack on Broadcast RC4. Proceedings of Fast Software Encryption 2001, LNCS, vol. 2355, Springer-Verlag, 2002.
18. Scott Fluhrer, Itsik Mantin, Adi Shamir: Weaknesses in the Key Scheduling Algorithm of RC4. Proceedings of SAC 2001, LNCS, vol. 2259, Springer-Verlag 2001.
19. Jovan Dj. Golic: Linear Statistical Weakness of Alleged RC4 Keystream Generator. Proceedings of EUROCRYPT 1997, LNCS, vol. 1233, Springer-Verlag 1997.