Tail-MAC Scheme for Stream Ciphers and Example Application with VMPC

Bartosz Zoltak[1]

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

Abstract. A simple algorithm for computing Message Authentication Codes for messages encrypted with stream ciphers is described along with the analysis of selected aspects of its security. The proposed algorithm was designed to minimize the cost of additional-to-encryption MAC-related procedures, like the udnerlying hash function in the HMAC scheme, by taking advantage of some of the data already computed by the cipher. The construction of the scheme, assuming a proper implementation with a selected stream cipher, appears to resist chosen plaintext/ciphertext attacks and provide practical unforgeability together with high software-implementation efficiency, reaching a rate of about 20 cycles for a total of encryption and authentication per byte.

Keywords: Authenticated encryption, MAC, HMAC, stream cipher, hash function

1 Introduction

A simple algorithm for computing Message Authentication Code (MAC) of an encrypted message, here termed "Tail-MAC scheme", is described.

Authenticated encryption, enabling verification, with high probability, whether the message output from the decryption process is the actual message which was encrypted, is a significant requirement in practical applications of cryptographic algorithms.

The NIST-approved standard, Keyed Hash Message Authentication Code (HMAC), is broadly analyzed, considered secure and can be proven secure under assumption of security of the underlying hash-function.

The approach here proposed was designed, on the contrary to HMAC, to minimize the computational cost of the additional-to-encryption MAC-related procedures while employing some of the internal-state data, already computed by the cipher. This approach, when implemented carefully, appears to provide a full MAC functionality, high performance in software and a comfortably-acceptable level of resistance to forgery. According to the performed analyses, the Tail-MAC scheme, when integrated with a fast stream cipher, like RC4 or the presented at FSE 2004 VMPC Stream Cipher, can provide securely authenticated encryption

at a performance rate of about 20 clock-cycles per byte on a modern computer processor.

Sections 2-4 discuss general features and definitions of the Tail-MAC scheme, Section 5 outlines the VMPC Stream Cipher, sections 6 and 7 present an example application of the Tail-MAC as it is integrated with VMPC, together with software-implementation performance and Section 8 discusses security of the Tail-MAC in a chosen ciphertext attack.

2 General description of the Tail-MAC scheme

The scheme is based on an internal state which is transformed along with the progress of the encryption process in a manner determined by ciphertext-, key- and Initialization Vector (IV)-derived information.

Implementation of the scheme, as it is integrated with an encryption algorithm, will vary from cipher to cipher. This results from the property that the scheme employs some of the internal-key-data already computed by the cipher. This feature, when applied carefully, allows to eliminate, or at least significantly reduce, any excessive computational work, which is required - besides the work performed by the encryption process - for example by the underlying hash-function in the HMAC scheme.

This allows to simplify the encryption-and-authentication procedure, obtain very high efficiency in software implementations and, according to the further discussed analyses, a reasonable level of security.

3 Definition of a *d*-level Tail-MAC scheme

The definition assumes existence of a *cipher* generating a stream of b-bit words.

The $(8 \times d \times b)$-bit internal state of the scheme consists of the following variables T and x_w:

T: $(8 \times d)$-element table of b-bit words. Let $T[n]$ denote n-th element of T.

$x_1, x_2, \ldots, x_d$: b-bit variables. Let $[x_1, x_2, \ldots, x_d]$ denote a $(b \times d)$-bit word combining $x_1, x_2, \ldots, x_d$

Let f denote a function and $iK(m)$ denote a part of the internal key of the *cipher* in time m; f and iK will be specified differently, depending on what cipher the Tail-MAC scheme is integrated with. Section 6 gives a proposed f function and $iK(m)$ for the published in [6] and outlined in Section 5 VMPC Stream Cipher.

Let h denote a function combining T with the cryptographic key (K), the message-unique Initialization Vector (V) and compressing it into a word of proposed size of 20 bytes (160 bits); h will vary depending on what cipher is chosen

to be used with the Tail-MAC.

Let $g, m; i, j; q, r$ be temporary variables.

Let $Pt[m]$ denote m-th b-bit word of plaintext
Let $Pt0$ denote a b-bit word set to zero ($Pt0 = 0$).
Let $Ct[m]$ denote m-th b-bit word of ciphertext.

Table 1. The Tail-MAC scheme

1. Set g, m, q, r to 0.
2. Encrypt $Pt[m]$ and store output in $Ct[m]$
3. For j from d down to 2 execute step 4: $*^1$ 4. $x_j = (x_j + x_{j-1})$ modulo 256 5. $x_1 = f(x_1, iK(m), Ct[m])$
6. xor $T[g, g+1, \ldots, g+d-1]$ with $[x_1, x_2, \ldots, x_d]$ $*^2$
7. $g = (g + d)$ modulo $(8 \times d)$ 8. $r = r + q$ 9. if $r = (8 \times d)$: Go to step 15
10. Increment m 11. If m is lower than total number of plaintext blocks: Go to step 2
12. Set q to 1 13. Encrypt $Pt0$ and store output in $Ct[m]$ 14. Go to step 3
15. MAC = $h(K, V, T)$
$*^1$ In the following $(d-1)$ steps j will take on values $d, (d-1), (d-2), \ldots, 2$. $*^2$ xor $T[g]$ with x_1; xor $T[g+1]$ with x_2; ...; xor $T[g+d-1]$ with x_d.

4 Design rationale

The Tail-MAC scheme is designed to keep a sufficiently long record of the information derived from ciphertext-, key- and IV-data in the *tail* comprising a set of variables $x_1, x_2, \ldots, x_d$ and mark the T table with the *tail* in an extent sufficient to make it hard to manipulate the ciphertext in a way which would allow to generate an unchanged T table for a changed message.

Selection of iK and the f function should be carried out in such way as to ensure corruption of any predictable patterns of the ciphertext which could be conveyed onto the *tail* and successively onto the T table in a chosen ciphertext attack.

The value of the d parameter, defining the length of the *tail*, indirectly implies the computational effort which would be required to forge the MAC in a chosen ciphertext attack. Further analysis of the scheme suggests that $d = 4$ provides a well sufficient security level (analyzed in detail in Sections 6 and 8) and enables a comfortable implementation of the system (as described in Section 6).

Applying the compression function h at the end of the process (h can be based either on the internals of the employed *cipher* or on one of the widely-analyzed hash-functions) is aimed mostly at preventing a possible leakage of key-information. A dedicated analysis of this aspect is however required separately for a specific implementation of the Tail-MAC with a selected encryption algorithm.

The scheme was designed to make it computationally unfeasible to obtain two identical T tables at any moment when processing two different messages encrypted with the same key and the same Initialization Vector. The extent of this difficulty is partly established by the d parameter determining the length of the *tail*, and magnified by the fact that T keeps record of reasonably many $(8 \times d)$ past values of $x_1, x_2, \ldots, x_d$ in a single run. A forgery attack would need to revert all the changes of the $x_1, \ldots, x_d$ variables and of the T table. The size of T $(8 \times d)$ and a proposed length of the *tail*, $d = 4$, appear to provide a comfortable resistance to the possible to predict attacks aimed at reverting these changes, as discussed in detail in Section 8.

Observation of other messages encrypted with the same key and IV (in practical applications the IV should be message-unique for messages encrypted with the same key) does not lead to a noticeable advantage the attacker might acquire (e.g. by trying to learn about the behavior of the f function by introducing different ciphertexts, observing the resulting MACs and trying to use this knowledge to revert the changes of $x_1, \ldots, x_d$ and T for his new message) mostly because of the use of the h compression function in the final step. Construction of the h function should corrupt any partial regularities of T, from observing which the attacker might benefit.

The post-processing of the T table (the final $(8 \times d)$ steps initialized in step 12) is designed to prevent possible forgery attempts through processing very short messages (one or a few bytes long), through manipulating only one or a few first or last bytes of the message or through appending or prepending attacker-chosen data to the message, which might be aimed at obtaining two messages differing minorly and generating the same T tables. The post-processing extends all the changes of the message or data appended/prepended to the message onto variables $x_1, x_2, \ldots, x_d$ and onto all the $(8 \times d)$ elements of the T table, which should make these changes hard to control. This effect is additionally magnified by the h compression function.

5 Description of the VMPC Stream Cipher and its KSA

VMPC was introduced at FSE 2004 as a simple and software-efficient stream cipher with a specified Key Scheduling Algorithm and Initialization Vector management procedure. The internals of VMPC can be comfortably employed to construct an efficient encrypt-and-authenticate system based on the Tail-MAC scheme. Following [6], the VMPC Stream Cipher, generating a stream of 8-bit words, is specified in Table 2.

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 2. 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

The VMPC Key Scheduling Algorithm (Table 3) 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

Table 3. 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. $Temp = P[n]$
       $P[n] = P[s]$
       $P[s] = Temp$

According to [6] there are no known security problems regarding this cipher and its KSA. The cipher is described to have a number of security advantages over the RC4 keystream generator, its KSA is reported to provide a random-looking diffusion of changes of one bit (or byte) of the cryptographic key of size up to 64 bytes onto the generated P permutation and onto output generated by the cipher. The cipher is also claimed to perform at a rate of about 12.7 clock-cycles per byte on a Pentium 4 processor. These features make the described algorithm a plausible candidate to illustrate a practical application of the Tail-MAC scheme on.

6 d-level Tail-MAC scheme integrated with VMPC Stream Cipher

Let $b = 8$ and $x_1, x_2, \ldots, x_d$, T, Pt, Ct, m, g be defined as in Section 3.

Let P, s, n, L, V, z be defined as in Section 5.

Table 4. Tail-MAC scheme with VMPC Stream Cipher

1. Run the VMPC Key Scheduling Algorithm
2.1. Set T to 0
2.2. Set $x_1, x_2, \ldots, x_d, m, g, n$ to 0
3. Append $(8 \times d)$ zeros to Pt ($Pt[x] = 0$ for $x \in \{L, L+1, \ldots, L+8 \times d-1\}$)
4. $s = P[(s + P[n])$ modulo 256]
5. $Ct[m] = Pt[m]$ xor $P[(P[P[s]] + 1)$ modulo 256]
6. For j from d down to 2: execute step 7:
7. $x_j = (x_j + x_{j-1})$ modulo 256
8. $x_1 = P[(x_1 + s + Ct[m])$ modulo 256]
9. For j from 1 to d: execute step 10:
10. xor $T[g + j - 1]$ with x_j
11. $Temp = P[n]$; $P[n] = P[s]$; $P[s] = Temp$
12. $g = (g + d)$ modulo $(8 \times d)$
13. $n = (n + 1)$ modulo 256
14.1. Increment m
14.2. If $m < (L + 8 \times d)$: Go to step 4
15.1. Store table T in table V.
15.2. Set z to $(8 \times d)$
15.3. Execute step 8 of the VMPC Key Scheduling Algorithm (Table 3)
16.1. Set L to 20
16.2. Execute step 2 of the VMPC Stream Cipher (Table 2) and save the 20 generated outputs as a 160-bit MAC.

The above implementation was designed to conform the general rationale described in Section 4. It makes use of the diffusion effect provided by the VMPC KSA in the construction of the h function (steps 15.1-16.2) to make h corrupt any possible patterns that might occur in the T table. The h function this way magnifies the avalanche effect, already ensured by the post-processing phase in steps 4-14.2 for $m \in \{L, L+1, \ldots, L+8 \times d-1\}$, which yields a hard to control or predict correlation of the ciphertext-, key- and IV-data with the resulting MAC. The same properties of the h function also provide a significant level of resistance against attempts of deducing any information about the internal key, stored in the P permutation, from the resulting MAC. Construction and extent of the post-processing phase ($8 \times d^2$ updates of the T table performed before passing T to the h function) is aimed at thwarting attempts to manipulate the

ciphertext by changing any part of it or by appending or prepending any data to it with the purpose of obtaining an unchanged T table for a changed message.

Construction of the f function and $iK(m)$ (step 8) fulfils its roles described in Section 4 by taking advantage of the pseudo-randomness, key- and IV-dependence and secrecy of the P permutation and the s variable. Any possible pattern an attacker might want to convey from a chosen ciphertext onto x_1 and consecutively onto $x_2, \ldots, x_d$ and T will be corrupted by P and s.

The most efficient forgery attack found against the described scheme is presented in Section 8.

7 Performance of the VMPC Stream Cipher with Tail-MAC scheme

Performance of a moderately optimized 32-bit assembler implementation of the Tail-MAC scheme integrated with the VMPC Stream Cipher, measured on an Intel Pentium 4, 2.66 GHz processor, is given in Table 5. Table 6 gives a performance rate of the bare Tail-MAC scheme, computed as a difference between the speed of the VMPC-with-Tail-MAC and the bare VMPC Stream Cipher.

Table 5. Performance of VMPC Stream Cipher with Tail-MAC scheme

MBytes/s	MBits/s	cycles/byte
127	1016	20.9

Table 6. Performance of bare Tail-MAC scheme

MBytes/s	MBits/s	cycles/byte
324	2592	8.2

The Tail-MAC integrated with RC4 would obtain a total performance rate of less than 20 cycles / byte, however due to several statistical weaknesses of RC4 and some security concerns regarding its KSA, the VMPC Stream Cipher, so far considered free from known weaknesses, was chosen for the example implementation of the Tail-MAC.

8 Chosen ciphertext attack against the Tail-MAC scheme

The most efficient chosen ciphertext attack found against the Tail-MAC scheme is described in this section. The complexity of the attack is 2^{144}, which can be considered a well-sufficient security level in any practical applications in the possible to predict future. In case a higher level of resistance against the discussed attack model was required - it can be obtained by increasing the d parameter.

The attack assumes that the attacker has full passive and active access to the ciphertext and can use an unlimited number of verification attempts for the new message. The purpose of the attacker is to introduce a new ciphertext which conforms the MAC of the original one.

The attack model begins with a random (or intended by the attacker) change of one bit (or byte) of the ciphertext - $Ct[m]$. The purpose of the attacker is to hide this change by manipulating the remaining part of the ciphertext in such way as to leave the resulting MAC unchanged.

The attack is illustrated on an example of the system described in Section 6, for $d = 4$ and $b = 8$, however analogous approach would apply for different d and b parameters and different choice of ciphers.

Let $x_w(m)$ denote the value of the x_w variable of the *tail* in step m;
$w \in \{1, 2, \ldots, d\}$
Let $n = (m \times d)$ modulo $(8 \times d)$).
Let "$(+)$" denote addition modulo $(8 \times d)$.

A change of $Ct[m]$ unconditionally causes a change of $x_1(m)$, 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 forces the attacker to perform the following changes of the ciphertext:

1. Change $Ct[m+1]$ 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 $Ct[m+2]$ 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 $Ct[m+3]$ 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 $Ct[m+4]$ in such way as to make $x_1(m+4)$ return to its original value. As a result $T[n(+)16]$ remains unchanged.

At this moment the attacker succeeded in stopping the avalanche of changes of elements of T, resulting from a change of $Ct[m]$, by reverting the changes of $x_1, x_2, \ldots, x_4$ in the earliest possible step $m+4$. 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 attacker needs to revert the changes of these elements of T, too. Operations similar to steps 1-4 need to be performed to refrain $x_1, x_2, \ldots, x_4$ from causing more damage to T and the additional requirement - to revert the already caused changes to T - needs to be satisfied. The most efficient approach found achieves that in the following steps 5-9:

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

6. Change $Ct[m+9]$ in such way as to make $x_4(m+12)$ return to its original value, make $x_1(m+9)$ change in such way as to revert the change of $T[n(+)4]$,

[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.

make $x_2(m+10)$ change in such way as to revert the change of $T[n(+)9]$, make $x_3(m+11)$ 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 $Ct[m+10]$ in such way as to make $x_3(m+12)$ return to its original value, make $x_1(m+10)$ change in such way as to revert the change of $T[n(+)8]$, make $x_2(m+11)$ 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 $Ct[m+11]$ in such way as to make $x_2(m+12)$ return to its original value, make $x_1(m+11)$ 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 $Ct[m+12]$ in such way as to make $x_1(m+12)$ return to its original value. As a result $T[n(+)16]$ remains unchanged.

A total complexity of the described attack is determined by the total number of changes to variables $x_1, x_2, \ldots, x_d$ and $T[0, 1, \ldots, 8 \times d - 1]$, which need to be reverted. Steps 1-9 determine this complexity, for the assumed $d = 4$ and $b = 8$, to $256^{18} = 2^{144}$.

Extending the length of the *tail* to $d = 5$ would, in an analogous attack, yield a complexity of $256^{25} = 2^{200}$ (which would also imply an increase of the size of the MAC to 25 or more bytes), however the implementation of the scheme would not be as comfortable as for $d = 4$ (while still easily achievable) which, given the fact that 2^{144} is a well out-of reach security level, encourages to propose $d = 4$ as sufficient for possible practical applications of the Tail-MAC scheme.

9 Conclusions

A simple and efficient algorithm for computing Message Authentication Codes for stream ciphers was proposed. Implementation of the scheme will vary depending on what cipher it is applied with, however the performed security analyses were intended to be cipher-independent and provide a general view of the resistance of the scheme against forgery in a chosen ciphertext attack.

An example application of the scheme as it is integrated with the VMPC Stream Cipher was given together with performance rates of its software implementation and some discussion of the security of this particular system.

The proposed Tail-MAC scheme appears to be a simple to implement and analyze, efficient in software implementations and, according to analyses performed so far, secure approach to provide authenticated encryption for stream ciphers.

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. T. Bellare, J. Guerin, and P. Rogaway: XOR MACs: New methods for message authentication using finite pseudorandom functions, Proceedings of CRYPTO 1995, LNCS vol. 963, Springer-Verlag, 1994
6. Bartosz Zoltak: VMPC One-Way Function and Stream Cipher, FSE 2004 Conference, proceedings to appear in LNCS, Springer-Verlag
7. NESSIE consortium Portfolio of recommended cryptographic primitives, 2003, www.cryptonessie.org
8. NESSIE consortium: Performance of Optimized Implementations of the NESSIE Primitives, 2003 www.cryptonessie.org
9. 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.
10. Scott R. Fluhrer, David A. McGrew: Statistical Analysis of the Alleged RC4 Keystream Generator. Proceedings of FSE 2000, LNCS, vol. 1978, Springer-Verlag, 2001.
11. Itsik Mantin, Adi Shamir: A Practical Attack on Broadcast RC4. Proceedings of FSE 2001, LNCS, vol. 2355, Springer-Verlag, 2002.
12. Scott Fluhrer, Itsik Mantin, Adi Shamir: Weaknesses in the Key Scheduling Algorithm of RC4. Proceedings of SAC 2001, LNCS, vol. 2259, Springer-Verlag 2001.
13. Jovan Dj. Golic: Linear Statistical Weakness of Alleged RC4 Keystream Generator. Proceedings of EUROCRYPT 1997, LNCS, vol. 1233, Springer-Verlag 1997.