A MAC forgery attack on SOBER-128

Dai Watanabe[1] and Soichi Furuya[1]

Systems Development Laboratory, Hitachi, Ltd.,
292 Yoshida-cho, Totsuka-ku, Yokohama, 244-0817, Japan
{daidai, soichi}@sdl.hitachi.co.jp

Abstract. SOBER-128 is a stream cipher designed by Rose and Hawkes in 2003. It can be also uses for generating Message Authentication Codes (MACs). The developers claimed that it is difficult to forge the MAC generated by SOBER-128, though, the security model defined in the proposal paper is not realistic. In this paper, we examine the security of the MAC generation function of SOBER-128 under the security notion given by Bellare and Namprempre. As a result, we show the MAC generation function of SOBER-128 is vulnerable against differential cryptanalysis. The success probability of this attack is estimated at 2^{-6}.

Keywords. Stream cipher, Message Authentication Code, Differential cryptanalysis, SOBER

1 Introduction

The desire to use a known cryptographic module for various applications exists from long ago. The first trial is realized as modes of operation of a block cipher. The OFB-mode and the counter-mode are the usages of a block cipher for a random number generation and CBC-MAC provides a message authentication mechanism. On the other hand, Anderson and Biham presented the construction of a block cipher from a pseudo-random number generator (PRNG) and a hash function [1]. The security of these modes of operation are provable under the assumption that the underlying primitives are ideal cryptographic functions.

Daemen considered the construction of elemental cryptographic functions, a block cipher, a PRNG , and a hash function, from unreliably weak functions, e.g., a round function of a block cipher [6]. The security of these constructions are not certain, but their processing speeds are often significantly faster than that of modes of operation. Daemen and Clapp proposed a cryptographic module Panama in 1998 [7]. Panama can be used as a PRNG and as a hash function. Ferguson *et al.* proposed an authenticated encryption algorithm Helix in 2003 [11]. Helix can be also used as a PRNG and for a MAC generation.

SOBER [18] is a stream cipher developed by Rose in 1998. SOBER adopts a linear feedback shift register (LFSR) defined over GF(2^8) as the update function of the internal state. This construction enables an efficient software implementation in 8-bit processors so that LFSRs defined over an extension field are employed not only by SOBER, but also by SSC2 [20], SNOW [8, 10], and so on.

Several variants of SOBER have been developed to strengthen its security or to be suitable for 16-bit and 32-bit processors. SOBER-t16 [12], t32 [13], Turing [14], SOBER-128 [15] are the published algorithms. SOBER-t16 and t32 were submitted to NESSIE project, but were rejected because some security flaws are reported [4, 5, 9]. A weakness of the initialization of Turing has been also reported [16].

SOBER-128 is the latest algorithm in SOBER family, and is the modified version of SOBER-t32. In addition, SOBER-128 can be used not only as a stream cipher, but also as a MAC generation. However, the security of MAC generation algorithm of SOBER-128 has not been evaluated, so is still unclear.

In this paper we examine the security of the MAC generation function of SOBER-128. We show that MAC generation algorithm of SOBER-128 is vulnerable against differential cryptanalysis and the forgery of the MAC generated by SOBER-128 is successful with high probability, about 2^{-4}.

The attack that we present in this paper is out of the expectation of the designers. Our attack follows the security notion defined by Bellare and Namprempre [2].

The rest of this paper is organized as follows. Firstly we define the supposed attacker in this paper in Sect. 2. Secondly we briefly describe the MAC generation algorithm of SOBER-128 in Sect. 3. Then we show how to forge MAC generated by SOBER-128 in Sect. 4 and discuss the weakness of SOBER-128 in Sect. 5. At last we summarize the result in Sect. 6.

2 The model of the attack

The designers mentioned that it is necessary for the recovery of the internal state (or the secret key) to forge a MAC generated by SOBER-128 and resolved the difficulty of a MAC forgery into that of the recovery of the internal state. In the attack to recover the internal state, the attacker can generate a MAC for a message with a secret key and an IV only once. Under the security notional description, the attacker can make queries only to a (probabilistic) MAC generation oracle.

In this paper, we consider a MAC forgery attack without any knowledge about the secret internal state. The attacker we supposed is a malicious person in a public communication channel. In our attack, there are a sender of messages, a receiver, and the attacker. The sender sends pairs of messages and the associated tags $(M, MAC(M, K))$ to the receiver. The attacker intercepts them, changes only the messages, then sends $(M', MAC(M, K))$ to the receiver. The attack is successful if the receiver does not detect the manipulation of a message.

The security notion of this attack was given by Bellare and Namprempre [2]. Under the notion, the attacker can make queries not only to the MAC generation oracle, but also to the MAC verification oracle. For each query, the MAC verification oracle returns 1 if the attached tag is correct, and return 0 otherwise. In real communication, the MAC verification oracle is the receiver and he outputs a requirement of retransmission to the sender if he detects the manipulation.

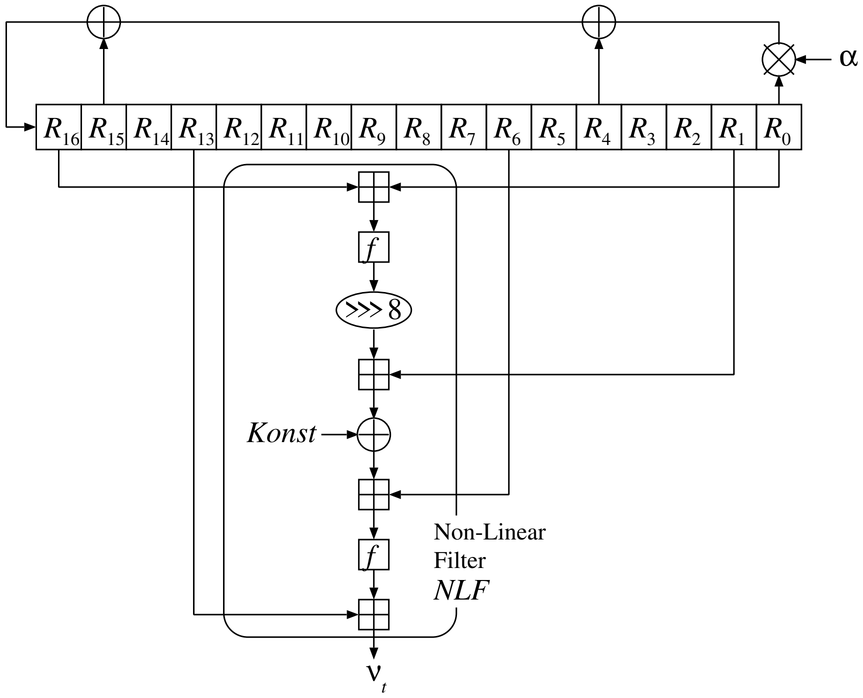

Fig. 1. The structure of non-linear filter NLF

There is no difference between these two attacks if the oracles are deterministic as a MAC verification oracle can be constructed from a MAC generation oracle. However, they are essentially different if the oracle is probabilistic (i.e. the oracle has a nonce as an input). In our model, the attacker can make a query for each oracle. On the other hand, the attacker whom the designers supposed can make only a query to a MAC generation oracle. The difference is critical to apply our attack given in Sect. 4.

3 The algorithm of SOBER-128

SOBER-128 is a filtering generator which takes a secret key and an initial vector (IV) as inputs. The lengths of inputs are less than 128 bits. The update function of the internal state is an LFSR whose feedback polynomial is defined over GF(2^{32}). The non-linear filtering function consists of additions modulo 2^{32}, XORing, circular shift, and 8×32-bit substitution box (S-box) S (See Figure 1 for detail).

3.1 The linear feedback shift register

Before giving the definition of the LFSR, we firstly define the bit representation of elements of a finite field GF(2^w). The extension field over GF(2) is defined by the residue field GF(2)$[z]/(\varphi(z))$, where $\varphi(z)$ is the irreducible polynomial. An element of GF(2^w) is a polynomial whose degree is less than w. The bit representation of a polynomial $a_{w-1}z^{w-1} + a_{w-2}z^{w-2} + \cdots + a_0$ is given by $a_{w-1}z^{w-1}||a_{w-2}z^{w-2}||\cdots||a_0$, where the notation $||$ means a bit-concatenation.

In the proposal of SOBER-128, the finite field GF(2^{32}) is defined by an extension field of a subfield GF(2^8). Firstly the subfield GF(2^8) is given by the residue field GF(2)$[z]/(x^8 + z^6 + z^3 + z^2 + 1)$. Next, GF($2^{32}$) is defined by the residue field GF(2^8)$[y]/(g(y))$, where $g(y) = \texttt{0xd0} \cdot y^3 + \texttt{0x2b} \cdot y^2 + \texttt{0x43} \cdot y + \texttt{0x67}$. The coefficients in the irreducible polynomial g are the hexadecimal expressions of the elements of GF(2^8).

Now we can define the feedback polynomial $p(x)$ of the LFSR of SOBER-128:

$$p(x) = x^{17} + x^{15} + x^4 + \alpha, \tag{1}$$

where $\alpha = \texttt{0x00000100}$ (hexadecimal expression) $= y$.

The registers of SOBER-128 is denoted by $R = (R[0], R[1], \ldots, R[16])$, where each $R[i]$ is a register of 32-bit length. We use subscript, like R_t, if the time t should be clarified. The update function defined by above feedback polynomial is calculated as follows:

$$\begin{aligned} \texttt{tmp} &= R[0], \\ R[i] &= R[i+1], \quad 0 \le i < 16, \\ R[16] &= R[14] \oplus R[4] \oplus \alpha \cdot \texttt{tmp}. \end{aligned} \tag{2}$$

Furthermore, we can reduce the calculation of the multiplication with a constant in GF(2^{32}) on 32-bit processor. The multiplication consists of only two operations, referring a pre-computed table Multab indexed by the most significant byte and a shift operation:

$$\alpha \cdot x = (x << 8) \ \hat{}\ \text{Multab}[(x >> 24) \& \texttt{0xFF}], \tag{3}$$

where ˆ is the XORing operation and $<<$, $>>$ are the left and right shift operations respectively.

3.2 Non-linear filtering NLF

The description of the non-linear filtering function NLF is not necessary for explaining our attack, but we give the brief description.

NLF is defined by the following equation (See Figure 1.):

$$NLF(R) = f(((R[0] \boxplus R[16])) \ggg 8 \boxplus R[1] \oplus Konst) \boxplus R[6]) \boxplus R[13],$$

where $\boxplus$ is the addition modulo 2^{32} and $\ggg$ is the rotation to right in a 32-bit register. 8×32-bit non-linear substitution f is defined by

$$f(a) = S_1[a_H]||(S_2[a_H] \oplus a_L),$$

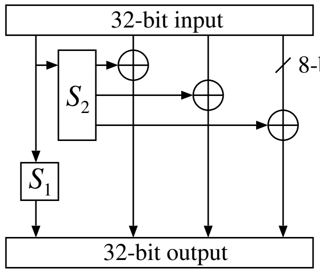

Fig. 2. The structure of function f

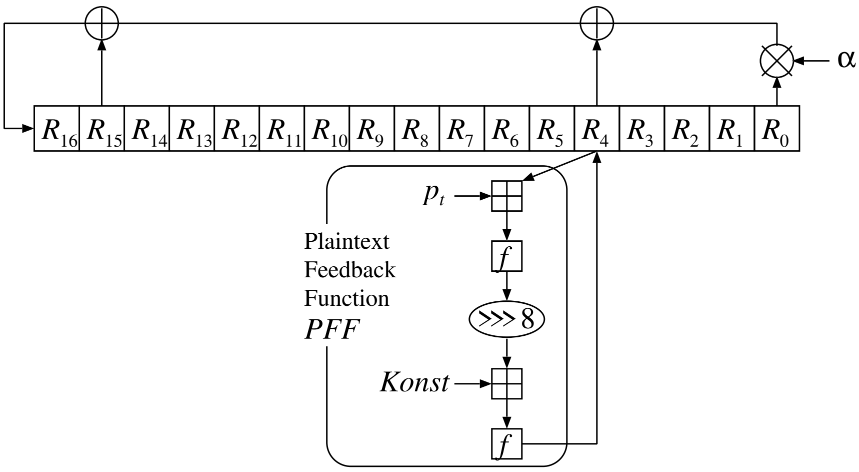

Fig. 3. The structure of plaintext feedback function PFF

where a_H and a_L are the most significant 8 bits and the least significant 24 bits of the input a respectively (See Figure 2.). S_1 and S_2 in Figure 2 are the substitution boxes that output 8 bits and 24 bits values respectively. Please refer the proposal document of SOBER-128 [15] for more detail about the substitution table. $Konst$ is dependent on the initial values, and does not change after the initialization. Hence we can consider it is a key-dependent constant.

3.3 The message feedback function PFF

The message feedback function PFF is used for injecting a message to the LFSR in MAC generation. PFF is given by the following equation:

$$PFF(R[4], p, Konst) = f((f(R[4] \boxplus p) \ggg 8) \boxplus Konst).$$

The value of the register $R[4]$ is replaced by the output of PFF.

3.4 MAC generation

The MAC generation of SOBER-128 is divided into two phases. The first phase is called MAC accumulation and the second phase is called MAC finalization. Before starting MAC generation, the internal state is initialized with a secret key and an IV.

In the MAC accumulation phase, the message words are injected into the LFSR by using the message feedback function PFF:

$$R_t[4] \leftarrow PFF(R_t[4], p_t, Konst) = f((f(R_t[4] \boxplus p_t) \ggg 8) \boxplus Konst). \qquad (4)$$

In the MAC finalization phase, the internal state is mixed by XORing the constant $INITKONST$ to $R[15]$, applying the non-linear update function *Diffuse* 18 times. The non-linear function *Diffuse* is defined as follows:

Step 1. Updating the LFSR.
Step 2. Applying NLF to the register and replacing $R[4]$ with the value $R[4] \oplus \nu$, where ν is the output of NLF.

After the mixing, SOBER-128 outputs MAC of arbitrary length. The last step is same as the random number generation using NLF.

4 The MAC forgery

Our attack aims to find a collision of messages with a same secret key and a same IV. For that, we apply differential cryptanalysis developed by Biham and Shamir [3]. Differential cryptanalysis observes the differential propagation of the target encryption function. In differential cryptanalysis, the attacker firstly encrypts amount of plaintext pairs with a fixed differential. and observes the distribution of the differentials of ciphertext. If the output differentials do not distribute uniformly, the attacker can recover some information of the secret key from the deviation of the distribution.

The basic idea of our attack is finding a pair of message sequence P and P', which yields same internal state value at a certain time T with a secret key and an IV. In a practical sense, the purpose of the attack is finding the differential sequence $\{\Delta p_t\}_t$ with which any message sequence pair $P = \{p_t\}_t$ and $P' = \{p_t \oplus \Delta p_t\}_t$ make a collision with high probability.

The search can be divided into two part. The search for the differential propagation of PFF with high probability and the construction of the differential elimination equation in the LFSR. In the following two subsection, each topic is discussed in detail.

4.1 The differential propagation in the message feedback function PFF

Firstly, we discuss about the differential propagation in the message feedback function PFF.

We denote the differential propagation in the function f by $\varDelta \xrightarrow{f} \varDelta'$. The differential characteristic of non-linear permutation f is very large because the transformation is not uniform. Only the most significant byte is transformed non-linearly. so that any differential $\varDelta$ whose most significant byte is zero does not influenced by f, i.e., $\varDelta \xrightarrow{f} \varDelta$ holds with probability 1. Hence, if the byte-wise expression of the differential of the register $R[4]$ is given by $(0, \delta_1, \delta_2, 0)$ and the differential of the message injection p is given by $(0, \delta'_1, \delta'_2, 0)$, the following differential propagation of PFF holds with high probability:

$$\begin{aligned}((0, \delta_1, \delta_2, 0), (0, \delta'_1, \delta'_2, 0)) &\xrightarrow{\boxplus} (0, \delta''_1, \delta''_2, 0)\\ &\xrightarrow{f} (0, \delta''_1, \delta''_2, 0)\\ &\xrightarrow{\ggg} (0, 0, \delta''_1, \delta''_2)\\ &\xrightarrow{\boxplus} (0, 0, \delta''_1, \delta''_2)\\ &\xrightarrow{f} (0, 0, \delta''_1, \delta''_2). \end{aligned} \tag{5}$$

4.2 How to eliminate differentials in the LFSR

Next we discuss how to eliminate differentials in the LFSR. To simplify the discussion, we replace the message feedback function PFF by XORing the message directly to $R_t[4]$. In fact, Eq. 5 indicates that any differential of $R_t[4]$ can be entirely overwritten by addition to a message block. Hence the discussion with the simplified message injection is useful. To avoid confusion, we denote the message block at time t by q_t and the differential by $\varDelta q_t$ respectively.

The internal state initialized by a secret key K and an initial vector I is denoted by R_0. We denote by $R_0(K, I)$ if the initial values should be clarified.

We denote the expression matrix of the LFSR of SOBER-128 defined over GF(2^{32}) by A. In the following discussion, we treat the internal state (registers) of the LFSR as a vector space over GF(2^{32}). Eq. 5 shows that the injection of the differential to the LFSR is identified by addition of the vector D_t given by

$$D_t = {}^t(0, 0, 0, 0, \varDelta q_t, 0, \ldots, 0).$$

The corresponding internal state to message sequence $q_t \oplus \varDelta q_t$ is given by

$$\begin{aligned} R'_T \oplus D_T &= A \cdot (R'_{T-1} \oplus D_{T-1}) \oplus D_T\\ &= A^2 \cdot (R'_{T-2} \oplus D_{T-2}) \oplus A \cdot D_{T-1} \oplus D_T\\ &\quad \vdots\\ &= A^T \cdot R_0 \oplus \sum_{t=0}^{T} A^t \cdot D_{T-t}. \end{aligned} \tag{6}$$

Eq. 6 indicates that the differential vector of the internal state becomes zero at time T iff the following equation is satisfied.

$$\sum_{t=0}^{T} A^t \cdot D_{T-t} = 0. \tag{7}$$

The differential vector $D_t = {}^t(0,0,0,0,\Delta q_t,0,\dots,0)$ is divided into the elemental vector $e_4 = {}^t(0,0,0,0,1,0,\dots,0)$ and the coefficient Δq_i:

$$D_t = \Delta q_t \cdot e_4.$$

Multiplying a constant is commutative with any matrix, hence we can transform Eq. 7 as follows:

$$\begin{aligned} 0 &= \sum_{t=0}^{T} A^t \cdot D_{T-t} \\ &= \sum_{t=0}^{T} A^t(\Delta q_{T-t} \cdot e_4) \\ &= (\sum_{t=0}^{T} \Delta q_{T-t} A^t) e_4. \end{aligned} \tag{8}$$

A differential sequence which satisfies this equation is given by Cayley-Hamilton relation. The characteristic polynomial of matrix A is given by Eq. 1 so that A satisfies the following relation:

$$A^{17} + A^{15} + A^4 + \alpha E = 0, \tag{9}$$

where E is the identity matrix of dimension 17. Therefore the following differential sequence Δq_t satisfies Eq. 8:

$$\Delta q_0 = \Delta q_2 = \Delta q_{13} = a, \quad \Delta q_{17} = \alpha \cdot a. \tag{10}$$

On the other hand, we examined exhaustive search for bite-wise truncated differential of message sequences $\{\Delta q_t\}_t$ which satisfy Eq. 9 and $T \leq 20$ on PC. The experiment concludes that Eq. 10 gives the best differential set, i.e. both the time T and the number of non-zero differentials are smallest.

Now we get back to the message injection by PFF. There are two different purpose to inject a message differential Δp_t into the LFSR. One is for injecting a certain differential value into a register, And the other is for eliminating a differential in $R_t[4]$. For each case, $\Delta p_t = \Delta q_t \lll 8$ and $\Delta p_t = \Delta q_t$ yields the same result as in the discussion of the simplified message injection.

Table 1 shows the propagation of the differential given in Eq. 10 in the internal state, where $b = \alpha \cdot a$.

Eq. 3 implies a significant characteristic of α. If the byte-wise expression of the differential Δq_0 is given by $(0,x,y,z)$, then $\alpha \cdot \Delta q_0 = (x,y,z,0)$ holds for every x,y,z. For example, the 32-bit differentials corresponding to the truncated differentials a and b in Table 1 are given by `0x00000001` and 0x00000100 respectively.

4.3 The success probability of the MAC forgery

In the differential propagation of the message feedback function PFF given by Eq. 5, the differential changes probabilistically only at the addition over GF(2^{32}).

Table 1. The differential propagation in the LFSR

Time	Δp_t	Δq_t	Differential in the LFSR
00	$a \lll 8$	a	$0000a000000000000$
01	0	0	$000a000000000000a$
02	$a \lll 8$	a	$00a0a0000000000a0$
03	0	0	$0a0a0000000000a00$
04	0	0	$a0a0000000000a000$
05	0	0	$0a0000000000a000b$
06	0	0	$a0000000000a000b0$
07	0	0	$0000000000a000b00$
08	0	0	$000000000a000b000$
09	0	0	$00000000a000b0000$
10	0	0	$0000000a000b00000$
11	0	0	$000000a000b000000$
12	0	0	$00000a000b0000000$
13	a	a	$00000000b00000000$
14	0	0	$0000000b000000000$
15	0	0	$000000b0000000000$
16	0	0	$00000b00000000000$
17	b	b	00000000000000000

Lipmaa and Moriai presented the efficient algorithm for calculating the differential probability of an addition modulo 2^n for arbitrary n [17]. For example, if the differential $a = \texttt{0x00000001}$ in Eq. 10 is chosen, the differential probabilities of PFF at each time $t = 0, 2, 13, 17$ are 2^{-2}, 2^{-2}, 2^{-1}, 2^{-1} respectively. Therefore the success probability of the MAC forgery of SOBER-128 is about 2^{-6}.

We examined the attack with a sample code provided by the designers. The success probability of the experimental attack is about $2^{-5.5}$.

5 Discussion

Generally, the supposed attack in MAC forgery is adaptive chosen plaintext attack. However, the attack applied in the previous section is known plaintext attack. Any information about IVs and the secret key is not necessary as the attack is applicable even if the internal state is perfectly random. Besides, though SOBER-128 has authenticated encryption mechanism, encrypting messages does not strengthen the security of message authentication function because altering ciphertext is equal to altering plaintext for the encryption by a stream cipher.

In this section, we examine some idea to improve the security of SOBER-128 and consider its efficiency.

The vulnerability of MAC generation algorithm of SOBER-128 is mainly derived from the fact that the substitution f is not uniformly non-linear. From the viewpoint of differential propagation, only the transformation of the most significant byte is non-linear. Besides, a rotation has no diffusion function. Hence at least 4 times iterations of these functions is necessary to apply the non-linear transformation to all bytes. This reduces the performance of SOBER-128.

Another idea to improve the security is injecting the output of PFF to plural registers for increasing the complexity. However, the transformation in Eq. 8 holds for any vector whose elements are defined by $v_i = x_i \cdot v \quad (\mathrm{GF}(2^{32}))$. Hence this change of algorithm does not improve the security.

6 Conclusion

In this paper, we show that the MAC generation algorithm of SOBER-128 is vulnerable under a certain practical assumption. We assume that the attacker intercepts the message and change some bits of the message, and sends it to the receiver. Under this assumption, the attacker can forge a message with probability 2^{-6}.

References

1. R. Anderson and E. Biham, "The Practical and Provably Secure Block Ciphers: BEAR and LION," *Fast Software Encryption, FSE'96*, Springer-Verlag, LNCS 1039, pp. 113–120, 1996.
2. M. Bellare and C. Namprempre, "Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm," *Advances in Cryptology, Asiacrypto 2000*, Springer-Verlag, LNCS 1976, pp. 531–545, 2000.
3. E. Biham and A. Shamir, *Differential Cryptanalysis of the Data Encryption Standard*, Springer-Verlag, 1993.
4. S. Babbage and J. Lano, "Probabilistic Factors in the Sober-t Stream Ciphers," *Third Open NESSIE Workshop*, proceedings, 2002.
5. C. De Cannière, J. Lano, B. Preneel, and J. Vandewalle, "Distinguishing Attacks on Sober-t32," *Third Open NESSIE Workshop*, proceedings, 2002.
6. J. Daemen, "Cipher and hash function design strategies based on linear and differential cryptanalysis," *Doctoral Dissertation*, K.U.Leuven, 1995.
7. J. Daemen and C. Clapp, "Fast Hashing and Stream Encryption with PANAMA," *Fast Software Encryption, FSE'98*, Springer-Verlag, LNCS 1372, pp. 60–74, 1998.
8. P. Ekdahl and T. Johansson, "SNOW – a new stream cipher, " *NESSIE project submission*, 2000, available at `http://www.cryptonessie.org/`.
9. P. Ekdahl and T. Johansson, "Distinguishing Attacks on SOBER-t16 and t32," *Fast Software Encryption, FSE 2002*, Springer-Verlag, LNCS 2365, pp. 210–224, 2002.
10. P. Ekdahl and T. Johansson, "A new version of the stream cipher SNOW, " *Selected Areas in Cryptography, SAC 2002*, Springer-Verlag, LNCS 2595, pp. 47–61, 2002.
11. N. Ferguson, D. Whiting, B. Schneier, J. Kelsey, S. Lucks, and T. Kohno, "Helix, Fast Encryption and Authentication in a Single Cryptographic Primitive," *Fast Software Encryption, FSE 2003*, pre-proceedings, pp. 345–362, 2003.
12. P. Hawkes and G. Rose, "Primitive Specification and Supporting Documentation for SOBER-t16 Submission to NESSIE," *First Open NESSIE Workshop*, proceedings, 2000.

13. P. Hawkes and G. Rose, "Primitive Specification and Supporting Documentation for SOBER-t32 Submission to NESSIE," *First Open NESSIE Workshop*, proceedings, 2000.
14. P. Hawkes and G. Rose, "Turing, A Fast Stream Cipher," *Fast Software Encryption, FSE 2003*, pre-proceedings, pp. 307–324, 2003.
15. P. Hawkes and G. Rose, "Primitive Specification for SOBER-128," IACR ePrint Archive, `http://eprint.iacr.org/2003/81/`, 2003.
16. A. Joux and F. Muller, "A Chosen IV Attack against Turing," *Selected Areas in Cryptography, SAC 2003*, pre-proceedings, 2003.
17. H. Lipmaa and S. Moriai, "Efficient Algorithms for Computing Differential Properties of Addition," *Fast Software Encryption, FSE 2001*, Springer-Verlag, LNCS 2355, pp. 336–350, 2001.
18. G. Rose, "A Stream Cipher based on Linear Feedback over GF(2^8)," "Proc. Australian Conference on Information Security and Privacy," Springer-Verlag, 1998.
19. R. Rueppel, *Analysis and Design of Stream Ciphers*, Springer-Verlag, 1986.
20. "The Software-Oriented Stream Cipher SSC2," *Fast Software Encryption, FSE 2000*, Springer-Verlag, LNCS 1978, pp. 31–48, 2000.