SOBER-t-16 and SOBER-t-32

P. Topark-Ngarm, P. Kanivichaporn

Abstract—
SOBER-t-16 and SOBER-t-32 are discussed in this paper. The t-class ciphers are based on the same principles as the original SOBER family: SOBER [1], SOBER-II [2], S16 and S32 [3], utilizing the structure SOBER-II and S16 are based. The t-class ciphers [4] [5] [6] are software stream ciphers designed for software implementation. Changes between the t-class and the original SOBER family are centered around constructing a stronger non-linear filter and more secure key loading.

I. Introduction

SOBER-t16 and SOBER-t-32 are synchronous stream ciphers designed for a secret key that is up to 128 bits in and 256 bits. The cipher outputs the key stream in 16-bit and 32 bits blocks.The t-class contains three ciphers based on 8-bit, 16-bit and 32-bit operations. SOBER-t32 is one of these ciphers.

The SOBER ciphers were primarily designed for use within mobile telephony. In an application such as mobile telephony, any loss of synchronization with such a stream cipher is disastrous. To counter the loss of synchronization, information is sent in small portions: *frames*. One possible solution to avoid loss of stream cipher synchronization is to encrypt each frame using a different secret key. An alternative solution (used, for example, in the GSM system) is to use a secret key for multiple frames. However, each encrypted frame is implicitly numbered with a value called a *frame key* or *hook*. SOBER-t16 and SOBER-t-32 can be employed with or without a frame key.

The t-class stream ciphers have four components: *key loading*, an LFSR, a *nonlinear filter (NLF)* and *stuttering*, as shown in Figure 1. The key loading sets the 17 words in the register of the LFSR to an *initial* state derived from the key. In some cases a re-synchronization key or frame key is used during key loading. The LFSR uses a *linear feedback function* to construct a stream of words from the *initial* state; this stream of words is the *LFSR stream* (s_n). The process of producing a new word in the LFSR stream is called a cycle of the LFSR. The purpose of the NLF is to disguise the linearity in the LFSR stream. After every cycle of the LFSR, the NLF combines words from the register in a non-linear function; the outputs form the *NLF stream* (v_n). The stuttering uses occasional NLF stream words to select NLF stream words to be used in the key stream (z_j) [1].

The t-class cipher differs from the original SOBER family in the following areas:

- t32 now uses the same structure as the rest of the t-class, while S32 had a different structure to SOBER-II and S16.
- The NLF has been significantly strengthened with the use of a nonlinear Sbox and the inclusions of a key-dependent constant as an additional variable.
- The key loading has changed in two ways.
- The frame is now loaded as if it were a 4-octet key.
- The key loading has been strengthened to eliminate algebraic relationships that existed between words of the initial state of the LFSR in SOBER-II.

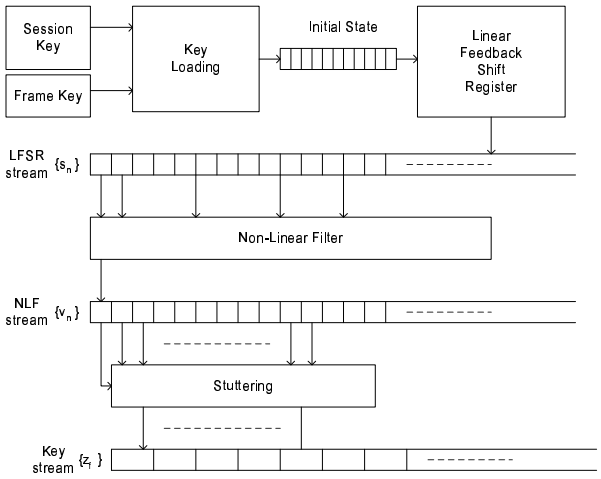

Fig. 1. The components of the t-class SOBER stream ciphers.

II. The Linear Feedback Shift Register

A linear feedback shift register (LFSR) is typically based on a recurrence relation over the Galois Field of order 2 ($GF(2)$). The output sequence (of bits) is defined by:

$$s_{n+k} = c_{k-1}s_{n+k-1} + c_{k-2}s_{n+k-2} + ... + c_1 s_{n+1} + c_0 s_n \quad (1)$$

where s_n is the n-th output of the sequence, the constant coefficients c_i, $0 \leq i \leq k-1$, are elements of $GF(2)$, that is, single bits, and k is called the *order* of the recurrence relation.

The LFSR is typically represented by the polynomial:

$$p_1(x) = x^k + c_{k-1}x^{k-1} + c_{k-2}x^{k-2} + ... + c_1 x + c_0 \quad (2)$$

where $p_1(x)$ indicates that multiplication and addition are over $GF(2)$. The operations involved, namely shifting and bit extraction, are efficient in hardware but inefficient in software, especially if the length of the shift register exceeds the length of the registers of the processor in question. In addition, only one bit of output is generated for each set of operations, which again is inefficient use of the general purpose CPU.

2.1 LFSR over $GF(2^w)$

LFSRs can operate over any finite field, and can be made more efficient in software by utilizing a finite field more

suited to the processor. Particularly good choices for such a field are the Galois Field with elements ($GF(2^w)$), where w is related to the size of items in the underlying processor, usually bytes or 16- or 32-bit words. The elements of this field and the coefficients of the recurrence relation occupy exactly one unit of storage and can be efficiently manipulated in software. In the meantime, the order k of the recurrence relation that encodes the same amount of state is reduced by a factor of w.

The field $GF(2^w)$ can be represented (the *standard representation*) as the modulo 2 coefficients of all polynomials with degree less than w. The choice of an irreducible degree w polynomial alters the way elements of the group are mapped into encoded words on the computer, but does not otherwise affect the actual group operations. The original SOBER family and the SOBER t-class ciphers use the irreducible polynomials shown in Table I.

w	Irreducible Polynomial used in tw	Hexadecimal
8	$x^8 + x^6 + x^3 + x^2 + 1$	0x14D
16	$x^{16} + x^{14} + x^{12} + x^7 + x^6 + x^4 + x^2 + x + 1$	0x150D7
32	$x^{32} + (x^{24} + x^{16} + x^8 + 1)(x^6 + x^5 + x^2 + 1)$	0x165656565

TABLE I

The irreducible polynomials used in tw, $w \in \{8, 16, 32\}$

Now that there is a known representation for the elements of the underlying field which can be stored in a single computer unit, the LFSR can be specified in terms of bytes or words instead of bits, and successive output values will also be those units rather than bits. The feedback function is still of the form of equation (2), however the values s_n and the coefficients c_i are elements of $GF(2^w)$, rather than bits. Cycling the LFSR requires a number of constant multiplication operations followed by XOR of these terms. Multiplication with any other constants i c is implemented using pre-calculated tables stored in read-only memory.

The t-class ciphers use LFSRs of the form:

$$s_{n+17} = \alpha s_{n+15} \oplus s_{n+4} \oplus \beta s_n \tag{3}$$

where $\oplus$ denotes addition over $GF(2^w)$ (equivalent to w-bit XOR), multiplication is performed over $GF(2^w)$, and are non-zero. This is the same form as used in SOBER-II and S16, indeed the feedback function for t16 is identical to that used in SOBER-II (S16). The feedback functions are shown in Table 2. The LFSR is represented by a polynomial over $GF(2^w)$:

$$p_w(x) = x^{17} \oplus \alpha x^{15} \oplus x^4 \oplus \beta \tag{4}$$

where the subscript w indicates that the addition and multiplication is over $GF(2^w)$.

The LFSR over the field $GF(2^w)$ is mathematically equivalent to w parallel shift registers over $GF(2)$ of length equivalent to the total state 13w, each with the same recurrence relation but different initial state. Let the polynomial p1(x) represent the LFSR over $GF(2)$. The multiplication

w	Feedback Function for tw
8	$s_{n+17} = 0\text{xCE}s_{n+15} \oplus s_{n+14} \oplus 0\text{x63}s_n$
16	$s_{n+17} = 0\text{xE382}s_{n+15} \oplus s_{n+4} \oplus 0\text{x67C3}s_n$
32	$s_{n+17} = s_{n+15} \oplus s_{n+4} \oplus 0\text{xC2DB2AA3}s_n$

TABLE II

The feedback functions for the t-class, where $\oplus$ denotes w-bit XOR and $\otimes$ denotes multiplication over $GF(2^w)$.

constants chosen minimize the number of coefficient not equal to one, such that the following properties are satisfied.

• **The LFSR has maximum length period.** The period has a maximum length of (2^{13w-1}) when p_1(x) is a *primitive* polynomial of degree 13w, that is, it divides x^d+1, for d = 2^{13w}-1, but not for any d that divides (2^{13w}-1).

• **Approximately half of the coefficients of p_1(x) are 1.** This condition is ideal for maximum diffusion and strength against cryptanalysis.

III. The Non-linear Filter

Much of the cryptographic security of the t-class SOBER family resides in the nonlinear filter (NLF) used to defend against attacks on the linear feedback or stuttering phase. It is therefore important to make this function as strong as possible without compromising the performance of the cipher.

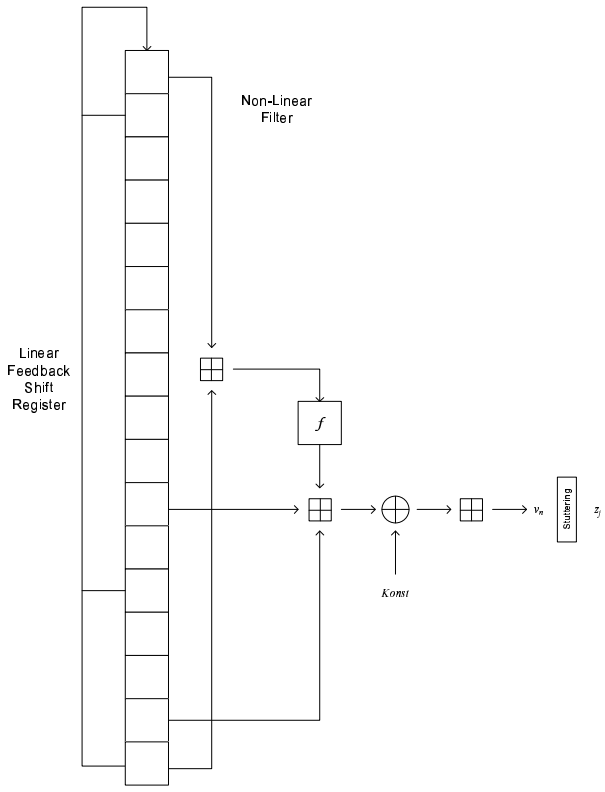

Fig. 2. The feedback functions for the t-class, where $\oplus$ denotes w-bit XOR and $\otimes$ denotes multiplication over $GF(2^w)$.

The NLF takes the values from certain positions in the register as inputs; as in SOBER-II and S16, the t-class ciphers use the register elements $r_0, r_1, r_6, r_{13}, r_{16}$ as inputs. As the LFSR is cycled before the NLF is applied, the NLF stream word vn depends on LFSR stream words $s_{n+1}, s_{n+2}, s_{n+7}, s_{n+14}, ands_{n+17}$. The t-class NLF also uses a key-dependent constant word called *Konst* as an input value. This value is derived immediately after the secret session key is loaded, and remains the same even if the frame number changes.

In designing the NLF we gave ourselves certain restrictions. The processors for which the t-class ciphers are designed are likely to have restrictions on the amount of ROM available. Thus these ciphers can afford to use a S-box in the NLF, although it would be preferable for the S-box to use only a small amount of memory; we restrict the S-box to containing only 256 entries where each entry is a w-bit word. The NLF should also be balanced (that is, every output word occurs with equal probability). Furthermore, we place the requirement if any five of the six inputs (including *Konst*) are fixed then every value for the remaining input corresponds to a unique output.

The NLF chosen for the t-class is of the form:

$$v_n = ((f_w(r_0 + r_{16}) + r_1 + r_6) \oplus Konst) + r_{13} \qquad (5)$$

as shown in Figure 2, where addition is modulo 2^w , and the function f_w changes for each value of w. In terms of LFSR stream words,

$$v_n = ((f_w(s_{n+1} + s_{n+17}) + s_{n+2} + s_{n+7}) \oplus Konst) + s_{n+14} \qquad (6)$$

The function f_w serves three purposes. Firstly, it removes the linearity in the least significant bit and adds significantly non-linearity to the remaining bits. Secondly, it ensures that the addition of r_0 and r_{16} does not commute with the addition of r_1 and r_6. Thirdly it ensures that every bit of the output of the NLF depends on every bit of r_0 and r_{16}.

XORing the value *Konst* into the NLF has two purposes. Firstly, it increasing the complexity of any attack (excluding exhaustive key search) as there are now 2^{16} possible NLF functions. Secondly, the *Konst* is XORed (rather than added) so as to lower the probability of the addition of r13 commuting with the addition of r_1 and r_6. There is still a small probability that the operations will commute, but this probability is low and relies on the value of *Konst*. This issue requires further analysis.

3.1 The Function f_w used in t_w, $w \in \{16,32\}$

In t16 and t32, the fw function has three parts, as shown in Fig. 3.

The input to the S-box should depend on every bit of r_0 and r_{16}, so the eight most significant bits (MSBs) of $(r_0 + r_{16})$ are extracted to be the input (reference) to the 256 entry S-box. We call this operation *most significant byte extraction*, and denote the operation by *MSBE*.

The S-boxes for t16 and t32, denoted SB16 and SB32 respectively, are a combination of the Skipjack S-box and an S-box tailor-designed by the Information Security Research Centre (ISRC) at the Queensland University of Technology. The eight MSBs of the output of SB16 and SB32 are defined by the Skipjack S-box. The remaining (w-8) least significant bits (LSBs) of the output of 16 SB and 32 SB are defined by the S-boxes constructed by the ISRC. These S-boxes were constructed as (w-8) mutually uncorrelated, balanced and highly non-linear single bit functions.

The output $f_w(x), w \in \{16, 32\}$ is determined as follows. Following MSBE, the most significant byte of X becomes the input to SBw . The most significant byte is then removed from X, and this value is XORed with the w-bit output of the S-box. Thus, the most significant byte of the output of $f_w(x), w \in \{16, 32\}$ is the output of the Skipjack S-box, while the least significant (w-8) bits are obtained by XORing the (w-8) bits of the output of SBw with the (w-8) least significant bits of the input. The function fw is defined this way to ensure that it is a highly non-linear permutation, while using only a single, small S-box. The function 8 f can be considered of the same form, noting that w-8 = 0.

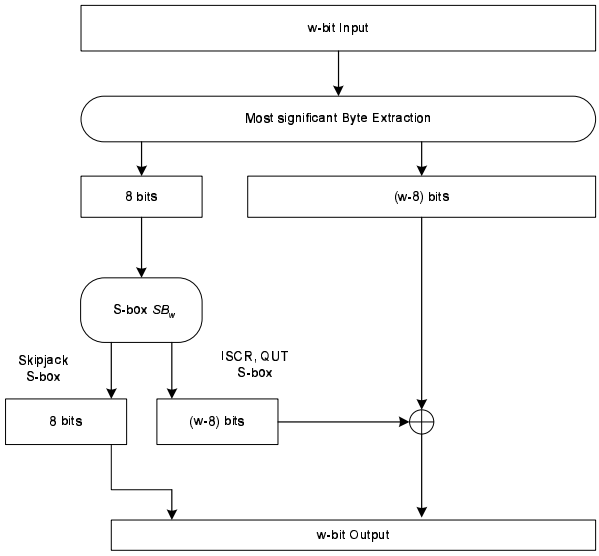

Fig. 3. The function f_w, used in t_w, $w \in$ {16,32}

IV. Stuttering the Non-Linear Output

It is easily conceivable that the state of the LFSR could be used to efficiently reconstruct the state, particularly by a fast correlation attack. The task is made much more difficult if some of the states are not represented in the output, in a way that is difficult to predict (irregular decimation). This is the role of the stuttering in the t-class ciphers. While the stuttering formed the most ad-hoc part of the design of SOBER and SOBER-II, the stuttering has also been the source of least trouble in the security analyses done. In updating the SOBER family to the t-class, various other forms of decimation were considered. However, the original stuttering still appears to offer the best increase in security. Thus, the stuttering employed in the t-class is derived from the stuttering used in SOBER and SOBER-II. Stuttering is based on occasional words of nonlinear out-

put being used to determine the inclusion of other words in the output stream. When the generator is started, the first NLF output (NLF stream word) is taken to be used as a *stutter control word* (SCW). Each SCW is broken into pairs of bits called *dibits*, with the least significant dibit being used first. The dibits provide the cipher with instructions regarding how many times to cycle the LFSR, whether to output an NLF output, and how this value is included in the key stream. When the instruction from all the dibits have been performed, the LFSR is cycled and the NLF output from the register forms the next SCW.

V. Key Loading

The t-class ciphers are designed (primarily) for applications in wireless telephony. In such applications, packets may be lost due to noise, synchronisation between the Mobile Station (cellphone) and the Base Station may be lost due to signal reflection, or a particular call might be handed off to a different base station as the phone zooms along a freeway. Any loss of synchronisation with a stream cipher is disastrous. One solution, used in the GSM system, is to have each encrypted frame implicitly numbered with a frame key, and the stream cipher re-keyed for each frame with the secret session key and the frame key. The frame key is public and consists of a 4-octet unsigned integer. SOBER, SOBER-II and S16 were designed to support such a two-tier keying structure; S32 was not. All t-class ciphers have been designed to support the two-tiered keying structure in addition to the standard mode of operation. The cipher is keyed and re-keyed using two operations:

- Include(X); adds the word X to 15 r modulo 2w.
- Diffuse(); cycles the register and XORs the output of the NLF with 4 r.

A t-class cipher is initially keyed using a secret, t-byte session key [0], , [1] K K t - K as follows:

1. The 17 words of state information are initialised to the first 17 Fibonacci numbers. There is no particular significance to these numbers being used, except for the ease of generating them. The value of Konst is set to the all zero word.
2. The cipher applies Loadkey(K[],t) which includes the session key bytes and session key length into the register, and diffuses the information throughout the register.
3. The LFSR is cycled and Konst is set to the value of the NLF output.

If the cipher is going to be re-keyed or the variable S-box constructed, then the 17 word state of the register, 0 16 , , r r K , (which we call the initial key state) can be saved at this point for later use, and the session key discarded. However, for shorter session keys, the session key could be saved and this procedure repeated as necessary, trading additional computation time for some extra memory.

If the cipher does not use the two-tiered keying structure, then the cipher produces a key stream with the register starting in the initial key state. That is, the initial key state is used as the initial state. However, if the cipher uses a frame key, the cipher first resets the register state to the saved initial key state, and then loads the 4- byte frame key frame[0],, frame[3] using Loadkey(frame[4]). The state of the register following the re-keying is taken as the initial state, and the cipher produces a key stream with the register starting in this state.

VI. Attacks

In [7-11] there are a lot of attacks have been investigated: Theoretical Attacks, Correlation Attacks, Distinguishing Attacks, Inversion Attacks, Guess-and-Determine Attacks, Extended Guess-and-Determine Attacks, etc.

Theoretical attacks

A necessary security requirement of a filter generator is that the LFSR length n and the algebraic order k of the non-linear filter function should be large enough so that is much bigger than the expected key stream length. For a very naive attack, let's consider SOBER-t16 without irregular decimation. The least significant bit of the output of the NLF can be written as boolean function with 35 binary input variables of algebraic order at most 32. So each least significant bit of the NLF is a known linear combination of all possible products of up to 32 of the 35 input bits. There are exactly

$$\sum_{i=0}^{32} = 34359737737 \tag{7}$$

of such products. With roughly twice of that many observed key stream bytes (65 giga-bytes), we can solve a linear equation system in that many variables, and we will get get a time complexity of O(2101). With the solution of these linear equations we get at most 35 bits of the length 17 linear feedback shift register over GF(216), i.e. 6 bits of the least significant bit LFSR and the bits of two full register cells of the LFSR over GF(216).

In order to get more bits we can proceed as follows. We drop the first one of the 34359737737 least significant output bits of the NLF, clock the LFSR once more, get the next least significant output bit and solve the simultaneous equations again. So, after only 11 additional clocks, and by exploiting the linear feedback recurrence, we can restore 17 consecutive LFSR output words.

Of course, we can extend this idea to all remaining output bit functions of the NLF. But due to the s-box and the carry propagation of the modulo addition the number of products increases dramatically. In order to avoid the gigantic requirements of the theoretical attack described above, we need a boolean function with n input bits and an algebraic order of k, so that solving a binary system

$$\sum_{i=0}^{k} \tag{8}$$

linear equations is still feasible. For instance, the function is a linear approximation of the least significant bit f1(x) of the f-function, which holds with probability p = 0.5313. Therefore we get a boolean approximation of the least significant bit of the NLF with 22 input bits and with

algebraic order of 11.

$$\sum_{i=0}^{11} = 2449867 \tag{9}$$

With time complexity of O(263) and roughly 4 megabyte key stream we can solve these linear equations, which hold only with a certain probability.

Correlation Attacks

The least significant output bit of a modulo addition is not connected to any carry bit, so this property could be a good starting point for an attack. If we can find a correlation of output bits of the NLF with a linear combination of least siginificat bit taps of the LFSR, we can launch a correlation attack to the least significant bit LFSR. But unfortunately, the least significant bit input of the f-function, is correlated only with probability 1/2 to the least significant bit output of f, which is completely useless.

Distinguishing Attacks

The attack is successful if one can distinguish the generated pseudo-random sequences from truly random sequences. The NESSIE-Tools were applied to SOBER-t16 and SOBER-t32 with 128 bit key size. However, the results didn't indicated a deviation from random behavior.

Inversion Attacks

Inversion attacks are so named because they "invert" the operations of the NLF: rather than using the inputs to determine the NLF output, an input is determined from the NLF output and the remaining inputs. The attacks were initially conceived as attacks on bit-wise LFSR-based ciphers with an NLF. The NLF had to be a linear function in the first or last input; for example, vt = NLF(t) = g(st, , st+x-1) + st+x (mod 2). In this case, the L-words are single bits, with st+x denoting the last input to the NLF. Suppose that the values ut, , ut+x-1, are candidates for the L-words/bits st, , st+x-1. If these candidates were correct then this would imply that:

$$u_{t+x} = v_t + g(u_t + ... + u_{t+x-1})(mod 2) \tag{10}$$

$$u_{t+x+1} = v_{t+1} + g(u_{t+1} + ... + u_{t+x})(mod 2) \tag{11}$$

$$u_{t+x+2} = v_t + g(u_{t+2} + ... + u_{t+x+1})(mod 2) \tag{12}$$

and so-forth. We say that the candidates ut+x, ut+x+1 and so forth have been determined. In an inversion attack, the attacker guesses the values for the candidates ut, , ut+x-1, and "inverts" the NLF to determine further candidates until a full candidate state t has been determined. This state is then tested. Provided x is less than the register length, such an approach can offer a significantly lower complexity than an exhaustive state search.

The inversion attack can be easily extended to word-oriented stream ciphers. The attacker guesses the values of ut, , ut+x-1, (these values are words) and inverts the NLF to determine ut+x, ut+x+1 and so forth. In the case of SOBER-t16, the attacker would guess values for the candidates u1, , u16 and Konst. Then the attacker determines the candidate u17 from the N-word v1 by inverting the NLF:

where "+" denotes addition modulo 216 and "-" denotes subtraction modulo 216. The attacker now has a full candidate state 1. This state is then tested.

Unfortunately for the attacker, this attack requires guessing 272 bits of information. There is no advantage to performing such an attack on SOBER-t16: an exhaustive key search has a considerably lower complexity. Hence, SOBER-t16 is resistant to the inversion attack.

Guess-and-Determine Attacks

Guess-and-Determine (GD) attacks are based on the assumption that the attacker knows part of a sequence of NLF outputs, and the attacker knows how many times the LFSR has been cycled between the NLF outputs. The stuttering destroys knowledge about the number of LFSR cycles; to account for this an attacker must assume the value of the dibits (two bit blocks) used to control the stuttering. The resulting attack is called an assume-guess-and-determine (AGD) attack.

Extended Guess-and-Determine Attacks

Extended guess-and-determine (EGD) attacks use the same approach as GD attacks, however EGD attacks use additional linear relationships between LFSR stream words that result from the linear feedback function. While GD attacks have been known since late 1998, EGD attacks have only been noticed since August 1999, so less is known about these attacks. To date, these are the EGD basis for the t-class ciphers:

$$r_1(x) = p_w(x) = x^{17} + \alpha x^{15} + x^4 + \beta \tag{13}$$

$$r_2(x) = p_w^2(x) = x^{34} + \alpha x^{30} + x^8 + \beta^2 \tag{14}$$

$$r_3(x) = p_w(x)(x^2 + \alpha)(x^4 + \beta) \tag{15}$$

$$r_3 = x^{23} + (\alpha^2 + \beta)x^{19} + \alpha\beta x^{15} + x^{10} + x^8 + \beta^2 x^2 + \alpha\beta^2 \tag{16}$$

Every possible EGD attack, using this EGD basis, has been tested. None of these EGD attacks has a complexity less than that of the best known GD attacks which have a complexity of 210w (when stuttering is ignored). This is not concrete proof of the resistance to EGD attacks, as it is still uncertain whether there are further linear relationships that are unresolvable with respect to this basis. We will continue to look for further linear relationships. We estimate that we have checked approximately half of the possible products of pw(x) that have degree less than or equal to 34; all have thus far proven to be resolvable with respect to the EGD basis above. We do not expect to find products of degree greater than 34 that are useful in an EGD attack.

VII. Conclusion

The t-class ciphers are conservatively designed stream ciphers with a very small software footprint, designed primarily for embedded applications in wireless telephony. Software implementations of LFSRs over $GF(2^w)$ can be extremely efficient, allowing well-tried design principles to be brought to bear in software ciphers. The t-class improves on the original SOBER family by offering a stronger non-linear filter and a more secure key loading process.

References

[1] G. Rose, "A stream cipher based on linear feedback over $GF(2^8)$," in *Australian Conference on Information Security and Privacy*, C. Boyd, Ed. 1998, Springer-Verlag.

[2] G. Rose, "Sober: A stream cipher based on linear feedback over $GF(2^8)$," 1998.

[3] G. Rose, "S16 & s32: A fast stream cipher based on linear feedback over gf(232)," 1998.

[4] P. Hawkes G. Rose, "The t-class of sober stream ciphers," 1999.

[5] P. Hawkes G. Rose, "Primitive specification and supporting documentation for sober-t16 submission to nessie," 2003.

[6] P. Hawkes G. Rose, "Primitive specification and supporting documentation for sober-t32 submission to nessie," 2003.