Exploiting Multiples of the Connection Polynomial in Word-Oriented Stream Ciphers

Philip Hawkes[1] and Gregory G. Rose[1]

Qualcomm Australia, Suite 410 Birkenhead Point, Drummoyne NSW 2047 Australia,
{phawkes, ggr}@qualcomm.com

Abstract. This paper describes some attacks on word-oriented stream ciphers that use a linear feedback shift register (LFSR) and a non-linear filter. These attacks rely on exploiting linear relationships corresponding to multiples of the connection polynomial that define the LFSR.
Keywords: stream ciphers, cryptanalysis, SOBER, t-class, SSC-II.

1 Introduction

This paper presents new attacks on word-oriented stream ciphers constructed from a *linear feedback shift register* (LFSR) and a *non-linear filter* (NLF). These ciphers are constructed from operations on blocks of bits called *words*, where the length of a word is denoted by w. In particular this paper analyses what we call SOBER-like ciphers (based on the SOBER family of ciphers [8, 12–14]) and SSC-like ciphers (as used in SSC [15], and SSC-II [16]).

The LFSR of a SOBER-like cipher produces a stream $\{s_t\}$ of w-bit words using operations over the Galois field of order 2^w, which is denoted $GF(2^w)$. The words s_t are called *L-words* and the stream is called the *L-stream*. The L-words $(s_0, \ldots, s_{r-1})$ are initialised from the secret key (some ciphers also initialise using a resynchronisation value). The remaining words are produced by iterating a linear recurrence $s_{t+r} = \sum_{i=0}^{r-1} \alpha_i s_{t+i}$, where $\alpha_i \in GF(2^w)$ are constant, and multiplication and addition are performed over $GF(2^w)$. Addition over $GF(2^w)$ is equivalent to bit-wise exclusive-OR (XOR). The LFSR is represented by the *connection polynomial*: $p(x) = x^r + \sum_{i=0}^{r-1} \alpha_i x^i$, where, once more, multiplication and addition are performed over $GF(2^w)$. The set of exponents of $p(x)$ with nonzero coefficients is called the *LFSR tapset*, denoted T. The LFSR of an SSC-like cipher differs in that it uses bit rotations rather than field multiplications and is based on a bit-wise LFSR (more details are given in Sect. 2). The vector $\sigma_t = (s_t, \ldots, s_{t+r-1})$ in either cipher is known as the *state* of the LFSR at time t.

The L-stream is fed through an NLF to produce the *N-stream* $\{v_t = F(\sigma_t)\}$. The words v_t are called *N-words*. SOBER-like ciphers use an LFSR with a large state σ_t, and the NLF relies on a small, fixed subset of the words in σ_t. That is, we can write $v_t = F(s_{t+\gamma_1}, \ldots, s_{t+\gamma_a})$, where $\Gamma = \{\gamma_1, \ldots, \gamma_a\} \subset \{0, \ldots, r-1\}$, is the *NLF tapset*. SSC-like ciphers, on the other hand, use an LFSR with a small state, and the NLF relies on the entire state.

SOBER-like ciphers use an LFSR, an NLF and a form of decimation called stuttering (described in Sect. 3). The resulting stream, denoted $\{z_n\}$, is the *key stream*. The stuttering chooses which N-words will be output to the key stream. The stuttering is intended to, and appears to, defeat attacks requiring large amounts of output, such as correlation attacks [4, 10]. However, the stuttering merely adds an almost constant factor to the complexity of the attacks described below.

In the analysis of stream ciphers based on bit-wise LFSRs, cryptanalysts found that attacks could be improved by exploiting linear relationships in the L-stream other than that expressed by the linear recurrence (see for example [4, 6, 10]). Such linear relationships correspond to multiples of the connection polynomial: the polynomial $r(x) = p(x) \cdot q(x) = \sum_{i=0}^{a} \epsilon_i x^i$, corresponds to a linear relationship of the form $\sum_{i=0}^{a} \epsilon_i s_{t+i} = 0$. For the remainder of the paper, a *multiple* refers to either a multiple of the connection polynomial or the linear relationship corresponding to that multiple. The main purpose of this paper is to provide examples of word-oriented stream ciphers for which the multiples can lead to low complexity attacks.

The first example is a component of the word-oriented stream cipher SSC-II [16]. SSC-II consists of two *half-ciphers* producing streams that are XORed to form the output. One of these half-ciphers is based on a 4-word LFSR (each word consists of 32 bits), with an NLF and no stuttering. The LFSR is based on a simple 127-bit, bit-wise linear recurrence that appears difficult to exploit due to the word-oriented structure of the NLF. However, a power of the bit-wise connection polynomial results in a linear relationship between corresponding bits of s_t, s_{t+63} and s_{t+127}. This paper describes how this relationship can be exploited in an attack of complexity $c(2^{41.7})$ against the LFSR-half cipher, where $c(N)$ indicates that the complexity is expected to be a small multiple of N. The authors would like to emphasise that this attack on the half-cipher does not defeat the entire SSC-II cipher.

The attack on the SSC-II half-cipher is due to the bit-wise connection polynomial of the LFSR having extremely low weight (that is, a low number of terms). If the LFSR was based on a higher-weight connection polynomial, but there was some low-weight, low-degree multiple $r(x)$, then a similar attack could be applied using this multiple. The linear recursion over $GF(2^w)$ in a SOBER-like cipher can be shown to be equivalent to implementing w parallel bit-wise LFSRs of length wr over $GF(2)$, see [9]. The constants α_i are chosen so that the bit-wise LFSR has many terms (high weight). This property defeats attacks similar to the above attack, as well as defeating other attacks designed for stream ciphers employing bit-wise LFSRs. The most successful attacks against SOBER-like ciphers have been what we call *guess-and-determine* (GD) attacks [1–3, 7, 8, 12, 13]. These GD attacks are based on exploiting two relationships: the linear relationship between L-words described by the LFSR; and the relationship between L-words and the key stream defined by the NLF. However, previous attacks have not exploited any further linear relationships.

The latest edition SOBER ciphers, the t-class [8], contains three ciphers: t8, t16 and t32. The cipher t16 is currently being assessed for use in "third generation" mobile communication systems, while t32 is being implemented for encryption in mail transfer sessions between e-mail servers. Thus far, our research into the t-class has not found any GD attacks exploiting further linear relationships that can decrease the complexity below that of previously known GD attacks. However, we have observed that multiples can lead to low-complexity GD attacks on other SOBER-like ciphers. This is demonstrated by a dummy SOBER-like cipher, TIPSY, for which the best GD attacks exploiting only the LFSR and NLF have complexity $c(2^{150})$. Our search method found a GD attack exploiting further linear relationships for which the complexity is reduced to $c(2^{117})$.

The paper is arranged as follows. Section 1.1 introduces some definitions. Section 2 describes the analysis of the LFSR half-cipher in SSC-II. Section 3 introduces GD attacks and the cipher TIPSY is analysed. Section 4 describes our method for finding GD attacks. Conclusions and areas for further research are discussed in Sect. 5.

1.1 Definitions

For any $t \geq 0$, we define a *candidate L-word* u_t to be a guess for the value of the L-word s_t, and define a *candidate state* $\mu_t = (u_t, \ldots, u_{t+r-1})$ to be a guess for the value of σ_t. We consider that an LFSR-based stream cipher is broken once the initial state of the LFSR has been determined. One method by which a stream cipher can be attacked is to search through every candidate μ_t until the value of σ_t is found (this process is commonly known as *guessing*). A candidate state μ_t is *tested* (to see if it is correct) by constructing a key stream using this value μ_t, and comparing the resulting key stream with the observed key stream. If the two streams match then the candidate is correct. In general, the large size of the register and the corresponding large number of possible candidate states make any such attack prohibitive.

2 Analysis of SSC-II

SSC-II [16] was proposed by Zhan, Carroll and Chan, and is based on $w = 32$-bit operations and w-bit words. The cipher consists of two *half-ciphers*: each half cipher produces a stream of 32-bit words and these streams are XORed to form the output. One half-cipher uses a lagged Fibonacci Generator which is based on addition modulo 2^{32} and is not considered here. The other half-cipher is based on a four-word LFSR. This LFSR produces an L-stream of 32-bit L-words $\{s_t\}$ by iterating the linear recurrence: $s_{t+4} = s_{t+2} \oplus (s_{t+1} << 31) \oplus (s_t >> 1)$, where $a << b$ ($a >> b$) denotes left (right) shifting of a by b bits. The bit-shifts are not cyclic: the remaining values are filled with zero bits. We denote the corresponding bit-stream by $\{b_i\}$ where $b_{32t+j} = s_t[j]$, the j-th bit of s_t, $0 \leq j \leq 31$, $t \geq 0$. The bit stream $\{b_i\}$ can be produced by a bit-wise LFSR with linear recurrence $b_{i+127} = b_{i+63} + b_i \pmod 2$. The LFSR in SSC-II calculates

32 bits of the L-stream simultaneously. SSC [15] employs an LFSR based on a similar principle.

The LFSR half-cipher has an NLF containing: addition modulo 2^{32}, denoted by $\boxplus$; 32-bit XOR; swapping the higher and lower order halves of the 32-bit word, denoted by *SWAP*; and including the carry resulting from adding words where $\rightarrow$ denotes outputting this carry. Let s_t^* denote the word s_t with the least significant bit (LSB) set to one. The N-word v_t is determined from the state $\sigma_t = (s_t, \ldots, s_{t+3})$ as follows:

$$A = s_t^* \boxplus s_{t+3} \rightarrow c_1, \qquad B = SWAP(A),$$
$$C = B \boxplus (s_{t+2} \oplus (c_1 \cdot s_t^*)) \rightarrow c_2, \; v_t = c_2 \boxplus (s_{t+1} \oplus s_{t+2}) \boxplus C,$$

where $c_1 \cdot s_t^* = 0$ if $c_1 = 0$ and $c_1 \cdot s_t^* = s_t^*$ if $c_1 = 1$.

Note 1. Let $\widehat{p}(x) = x^{127} + x^{63} + 1$ denote the connection polynomial for the bit stream $\{b_i\}$. Due to cancellation of terms, $\widehat{p}^2(x) = x^{127 \cdot 2} + x^{63 \cdot 2} + 1$, $\widehat{p}^4(x) = x^{127 \cdot 4} + x^{63 \cdot 4} + 1$ and so forth. Thus, $\widehat{p}^{32}(x) = x^{127 \cdot 32} + x^{63 \cdot 32} + 1$, indicating that $b_{i+127 \cdot 32} = b_{i+63 \cdot 32} + b_i$. This implies that $s_{t+127}[m] = s_{t+63}[m] + s_t[m]$, for each m, $0 \leq m \leq 31$, and thus $s_{t+127} = s_{t+63} \oplus s_t$.

This linear relationship is likely to lend the LFSR half-cipher to a fast correlation attack. The authors are currently analysing SSC-II to assess the complexity of such an attack. The following attack illustrates an alternative method of exploiting this linear relationship. The 32-bit words are first divided into two 16-bit half-words: for example, $s_{t+i} = sH_{t+i} \| sL_{t+i}$ and $v_{t+j} = vH_{t+j} \| vL_{t+j}$. Note that the half-word N-words vH_t and vL_t are functions of the half-words sH_{t+i} and sL_{t+i}, $0 \leq i \leq 3$, using addition modulo 2^{16} (denoted by $\boxplus$), 16-bit XOR and carries d_i from the addition of the lower half-words:

$$AL = sL_t^* \boxplus sL_{t+3} \rightarrow d_1 \;, \qquad AH \;=\; sH_t \boxplus sH_{t+3} \boxplus d_1 \rightarrow c_1 \;,$$
$$CL = AH \boxplus (sL_{t+2} \oplus (c_1 \cdot sL_t^*)) \rightarrow d_2 \;,$$
$$CH = AL \boxplus (sH_{t+2} \oplus (c_1 \cdot sH_t)) \boxplus d_2 \rightarrow c_2 \;,$$
$$vL_t = c_2 \boxplus (sL_{t+1} \oplus sL_{t+2}) \boxplus CL \rightarrow d_3 \;,$$
$$vH_t = (sH_{t+1} \oplus sH_{t+2}) \boxplus CH \boxplus d_3 \;.$$

(The *SWAP* step is integrated into the evaluation of CL and CH). If the values of $c_1 \in \{0, 1\}$, $(c_2 \boxplus d_1) \in \{0, 1, 2\}$ and $(d_2 \boxplus d_3) \in \{0, 1, 2\}$ are known, then the NLF half-word outputs can be written as:

$$vL_t = sH_t \boxplus sH_{t+3} \boxplus (sL_{t+2} \oplus (c_1 \cdot sL_t^*)) \boxplus (sL_{t+1} \oplus sL_{t+2}) \boxplus (c_2 \boxplus d_1) \;,$$
$$vH_t = sL_t^* \boxplus sL_{t+3} \boxplus (sH_{t+2} \oplus (c_1 \cdot sH_t)) \boxplus (sH_{t+1} \oplus sH_{t+2}) \boxplus (d_2 \boxplus d_3) \;.$$

For fixed values of c_1, $(c_2 \boxplus d_1)$ and $(d_2 \boxplus d_3)$, the expression for the LSB of vL_t provides a linear relationship between the LSBs of sL_t^*, sH_t, sL_{t+1} and sH_{t+3}. Similarly, the expression for the LSB of vH_t provides a linear relationship between the LSBs of sL_t^*, sH_t, sH_{t+1} and sL_{t+3}. The LSB of sL_t^* is one, so this can be ignored.

Consider the sets $X = \{0, 1, 2, 3, 63, 64, 65, 66, 126, 189\}$,

$Y = \{0, 63, 126, 127, 189, 190, 253, 254, 317, 381\}$, and

$Z = \{0, 1, 2, 3, 63, 64, 65, 66, 126, 127, 128, 129, 130, 189, 190, 191, 192, 193,$
$253, 254, 255, 256, 257, 317, 318, 319, 320, 381, 382, 383, 384\}$.

The values of L-words s_{t+j}, $j \in Z$, can be derived from values of the L-words s_{t+i}, $i \in X$, by applying the equation in Note 1. For example, $s_{t+127} = s_{t+63} \oplus s_t$, and $s_{t+191} = s_{t+127} \oplus s_{t+64}$. Thus, each L-word s_{t+j}, $j \in Z$, can be expressed as $s_{t+j} = \bigoplus_{i \in X} \beta_{j,i} s_{t+i}$, where $\beta_{j,i} \in \{0, 1\}$ for $i \in X$. Furthermore, these equations relate bits of the L-words s_{t+j}, $j \in Z$, to corresponding bits of the L-words s_{t+i}, $i \in X$: $s_{t+j}[m] = \bigoplus_{i \in X} \beta_{j,i} s_{t+i}[m]$, for each m, $0 \leq m \leq 31$. Note that the values of the 10 N-words v_{t+j}, $j \in Y$ rely on the set L-words s_{t+j}, $j \in Z$. Each bit of these L-words s_{t+j}, $j \in Z$ is, in turn, a linear function of the corresponding bits in 10 L-words s_{t+i}, $i \in X$. Candidates u_{t+i}, $i \in X$, for the L-words s_{t+i}, $i \in X$, are determined as follows.

From the expressions for the 20 half-word outputs vL_{t+j} and vH_{t+j}, $j \in Y$, we get 20 linear equations in the LSBs of uH_{t+j}, uL_{t+j+1}, uH_{t+j+1}, uL_{t+j+3} and uH_{t+j+3}, $j \in Y$. The attacker guesses the values of of c_1, $(c_2 \boxplus d_1)$ and $(d_2 \boxplus d_3)$ in the expression for each N-word v_{t+j}, $j \in Y$. For each of the 10 N-words there are 2 possible values for c_1, and 3 possible values each for $(c_2 \boxplus d_1)$ and $(d_2 \boxplus d_3)$. Therefore, the total number of guesses is $(2 \cdot 3^2)^{10} = 2^{41.7}$. These values are subtracted from the expressions for the 20 half-word outputs vL_{t+j} and vH_{t+j}, $j \in Y$, to get 20 linear equations in the LSBs of uH_{t+j}, uL_{t+j+1}, uH_{t+j+1}, uL_{t+j+3} and uH_{t+j+3}, $j \in Y$. As noted above, each of these LSBs is, in turn, a linear equation in the LSBs of uL_{t+i} and uH_{t+i}, $i \in X$. Thus the attacker obtains 20 linear equations in the LSBs of uL_{t+i} and uH_{t+i}, $i \in X$ (these LSBs represent a total of 20 bits). These equations are solved to obtain the LSBs of uL_{t+i} and uH_{t+i}, $i \in X$. From these LSBs, the attacker determines uL_{t+j} and uH_{t+j}, $j \in Z$, which enables the attacker to determine the carries up to the second LSBs of vL_{t+j} and vH_{t+j}, $j \in Y$. After subtracting these carries, the attacker now has 20 linear equations in the second LSBs of uH_{t+j}, uL_{t+j+1}, uH_{t+j+1}, uL_{t+j+3} and uH_{t+j+3}, $j \in Y$. Once again, each of these bits is a linear equation in the second LSBs of uL_{t+i} and uH_{t+i}, $i \in X$. The attacker obtains the system of 20 linear equations in the second LSBs of uL_{t+i} and uH_{t+i}, $i \in X$ (20 in total), and solves this system to obtain these values. This process is repeated to obtain all of the bits in uL_{t+i} and uH_{t+i}, $i \in X$. These candidates (uL_{t+j} and uH_{t+j}, $j \in X$) combine to form several full states, any of which may be tested (by producing some of the N-stream and comparing it with the observed key stream).

As mentioned above, the total number of guesses is $2^{41.7}$, so the process complexity of the attack is $c(2^{41.7})$. The data complexity of the attack is small: the attacker requires v_{t+j}, $j \in Y$, for a single t, which will require observing 382 consecutive key-stream words. This attack is feasible for one primary reason: the bit-wise connection polynomial has a small number of terms. The attack would also have been feasible if there was a low-weight, low-degree multiples of

the bit-wise connection polynomial. However, the attack cannot be applied if the weight of the multiple is sufficiently high, or the degree is sufficiently large, for the following reasons. A high-weight multiple of the bit-wise connection polynomial would require more equations in the N-words before system of bit-wise linear equations was solvable. Consequently, more values of $c_1, (c_2 \boxplus d_1), (d_2 \boxplus d_3)$ would be guessed, increasing the complexity and rendering the attack infeasible. On the other hand, if the degree of the multiple exceeds the maximum number of key-stream words produced from a single initial state, then this relationship cannot be exploited, regardless of weight.

3 Guess-and-determine Attacks

The LFSRs of SOBER-like ciphers correspond to bit-wise connection polynomials with extremely large numbers of terms. For example, the LFSR of t16 has a corresponding bit-wise connection polynomial with approximately 136 terms. This property helps SOBER-like ciphers resist the kind of attack described in the previous section. The most successful attacks [1–3, 7, 14, 13, 12] against SOBER-like ciphers have been GD attacks (there is no common name for these attacks). The following example describes a dummy SOBER-like cipher which is used to demonstrate how GD attacks are performed, and how GD attacks can, in some cases, be improved by exploiting multiples.

Example 1. TIPSY is a SOBER-like cipher designed for $w = 16$-bit processors, so the words are 16-bits long and all operations are 16-bit operations. TIPSY uses the LFSR tapset $T = \{0, 1, 4, 13\}$ and the NLF tapset $\Gamma = \{0, 5, 10, 11\}$. The linear recursion is of the form $s_{t+13} = s_{t+4} + s_{t+1} + \alpha s_t$, where $\alpha =$ 0xEDED, and addition and multiplication are performed over $GF(2^{16})$. The corresponding connection polynomial is $p(x) = x^{13} + x^4 + x + \alpha$. The NLF is of the form: $v_t = F(s_t, s_{t+5}, s_{t+10}, s_{t+11}) = f(s_t \boxplus s_{t+11}) \boxplus s_{t+5} \boxplus s_{t+10}$, where $\boxplus$ denotes addition modulo 2^{16} and f is a fixed, nonlinear, one-to-one 16-bit S-box. TIPSY decimates the N-stream to form the key stream using the same stuttering as t16 (the stuttering is described in Sect. 3.1).

As mentioned in Sect. 1.1, a stream cipher can be broken by guessing the value of any state σ_t, but the large size of the register and the corresponding large number of possible candidate states make any such attack prohibitive. GD attacks guess only a small set of candidate L-words, rather than an entire state. These attacks then use some observed N-stream words, and the relationships resulting from the LFSR and the NLF, to determine an entire state from this smaller set of L-words.

Example 2. In attacking TIPSY, if u_t, u_{t+1} and u_{t+13} are guessed, then u_{t+4} can be determined as $u_{t+4} = u_{t+13} + u_{t+1} + \alpha u_t$. Alternatively, if u_{t+5}, u_{t+10} and u_{t+11} are guessed then u_t can be determined from v_t; if $\boxminus$ denotes subtraction modulo 2^{16}, then $u_t = f^{-1}(v_t \boxminus (u_{t+5} \boxplus u_{t+10})) \boxminus u_{t+11}$.

These two processes of determining L-words are called *D-exploiting* the LFSR and NLF respectively (the 'D' is for 'determine'). Note that, for TIPSY, D-exploiting the LFSR and NLF is computationally equivalent to $c(1)$ encryption. The same applies to the t-class ciphers. D-exploiting the NLF is not a new concept: inversion attacks [6] and the generalised inversion attacks [5] are based on a similar approach.

Given a suitable portion of the N-stream,[1] previous GD attacks were based on **guessing** candidates for a small set of L-words, D-exploiting the LFSR and NLF to **determine** a full candidate state, and then **testing** this candidate state. These analyses of SOBER-like ciphers examined only those GD attacks that exploit the relationships explicitly defined by the LFSR and NLF. This paper extends the range of GD attacks by D-exploiting further multiples. There are simply too many multiples to begin searching for all attacks exploiting all possible multiples. Consequently, a method has been developed for reducing the amount of work by considering multiples that are more likely to lead to improved attacks: the rationale behind the authors' approach is described in Sect. 4. Using this method, the authors conducted a search for attacks exploiting polynomials of degree $2r$ (twice the degree of $p(x)$) or less and with 10 or less terms. This method cannot be guaranteed to find the best attack, as there may be some other high-weight or high-degree polynomial which can be exploited in a low complexity attack. However, the existence of such an attack becomes more unlikely as the weight and degree of the polynomials increases.

When applied to the t-class ciphers, the analysis described in Sect. 4 revealed that the additional linear relationships did not provide an attack of lower complexity than was already known. However, the analysis of TIPSY did find improvements by exploiting further multiples. The lowest complexity GD-attack D-exploiting only the LFSR and NLF of TIPSY has complexity $c(2^{128})$, given a suitable portion of the N-stream. Using the method described in Sect. 4, the authors found the following attack of complexity $c(2^{96})$, given a suitable portion of N-stream, a significant improvement.

Example 3. Table 1 describes an GD attack on TIPSY that D-exploits the LFSR, the NLF and the following multiples:

$$\begin{aligned}
p^2(x) &= x^{26} + x^8 + x^2 + \alpha^2 \ , \\
r_1(x) &= (x^9 + x^6 + x^3 + 1) \cdot p(x) \\
&= x^{22} + x^{19} + x^{16} + \alpha x^9 + \alpha x^6 + \alpha x^3 + x + \alpha \ , \\
r_2(x) &= (x^{12} + \alpha x^{11} + \alpha^2 x^{10} + x^6 + x^3 + \alpha x^2 + \alpha^2 x + 1) \cdot p(x) \\
&= x^{25} + \alpha x^{24} + \alpha^2 x^{23} + x^{19} + (\alpha^3 + 1)x^{10} + \alpha^2 x^5 + (\alpha^3 + 1)x + \alpha \ .
\end{aligned}$$

To perform the attack, a portion of the N-stream must be observed, including v_{t+i}, $i \in \{4, 7, 11, 12, 17, 18, 22, 23\}$ for some value of t. Let ϕ_t denote the six-word candidate vector $\phi_t = (u_{t+12}, u_{t+14}, u_{t+15}, u_{t+17}, u_{t+22}, u_{t+27})$. For a given

[1] The problem of obtaining a suitable portion of N-stream from the key stream is addressed in Sect. 3.1.

value of ϕ_t, Steps 2 to 18 in Table 1 determine candidates for the 17 L-words:

$$s_{t+i},\ i \in \{4,5,6,7,8,9,18,21,23,25,28,29,30,32,33,34,41\}\ .$$

For example, in Step 2, the value of u_{t+23} is determined from the values of v_{t+12}, u_{t+12} u_{t+17}, and u_{t+22} by D-exploiting the NLF:

$$u_{t+23} = f^{-1}(v_{t+12} \boxminus (u_{t+17} \boxplus u_{t+22})) \boxminus u_{t+12}.$$

Table 1. A GD attack on TIPSY exploiting the LFSR, the NLF, $p^2(x)$, $r_1(x)$ and $r_2(x)$, given v_{t+i}, $i \in \{4,7,11,12,17,18,22,23\}$. "Action" indicates the following actions: C, perform an NLF check; G, guess values; L, D-exploit the LFSR; N, D-exploit the NLF; r_1, D-exploit the multiple $r_1(x)$; r_2, D-exploit the multiple $r_2(x)$; S, D-exploit the square of the connection polynomial ($p^2(x)$); T, test the given candidate state. In the next two columns a candidate L-word u_{t+i} is indicated using the value of i. "Used" indicates those values used to determine or check the value indicated in the "Det." column.

Step	Act.	Values Used	Value Det.	Step	Act.	Values Used	Value Det.
1	G		12,14,15,	13	L	4,5,17	8
			17,22,27	14	L	8,9,12	21
2	N	$v_{t+12}, 12, 17, 22$	23	15	L	17,18,21	30
3	N	$v_{t+17}, 17, 22, 27$	28	16	S	4,12,30	6
4	S	15,17,23	41	17	r_1	6,7,9,12,15,22,28	25
5	L	14,15,27	18	18	L	21,22,25	34
6	N	$v_{t+7}, 12, 17, 18$	7	19	C	23,28,33,34	v_{t+23}
7	N	$v_{t+18}, 18, 23, 28$	29	20	G		11
8	L	28,29,41	32	21	N	$v_{t+11}, 11, 21, 22$	16
9	N	$v_{t+22}, 22, 27, 32$	33	22	L	12,16,25	13
10	S	7,15,33	9	23	S	8,16,34	10
11	N	$v_{t+4}, 9, 14, 15$	4	24	T	μ_{t+4}	
12	r_2	4,9,14,23,27,28,29	5				

Note that the L-words s_{t+i}, $i \in \{23, 28, 33, 34\}$, are the inputs to the NLF producing v_{t+23}, and candidates for all these inputs are known after Step 18 is performed. However, v_{t+23} has not been used to determine any of these values when exploiting the NLF, so these candidates are independent of the value of v_{t+23}. Clearly, if the candidates in ϕ_t are correct, then $F(u_{t+23}, u_{t+28}, u_{t+33}, u_{t+34}) = v_{t+23}$. If $F(u_{t+23}, u_{t+28}, u_{t+33}, u_{t+34}) \neq v_{t+23}$, then at least one of the candidates in ϕ_t is incorrect, and there is no use in completing any further steps. This information can be used to eliminate incorrect values of ϕ_t using a process called an *NLF check*. If $F(u_{t+23}, u_{t+23}, u_{t+33}, u_{t+34}) = v_{t+23}$, then the vector ϕ_t, is said to *pass* the NLF check, otherwise it *fails*. If ϕ_t fails the NLF check in Step 19,

then the attack returns to Step 1 and tries another guess for ϕ_t, otherwise the attack proceeds to Step 20.

At Step 20, a candidate u_{t+11} for s_{t+11} is guessed. Steps 21, 22 and 23 determine candidates u_{t+16}, u_{t+13} and u_{t+10}. Thus after Step 23, a candidate state $\mu_{t+4} = (u_{t+4}, \ldots, u_{t+16})$ for the state σ_{t+4} has been determined. This candidate state μ_{t+4} is then tested in Step 24. If μ_{t+4} is incorrect, then the attack returns to Step 20 and guesses another value for u_{t+11}, unless all values for u_{t+11} have been tested for a given value of ϕ_t, in which case the attack returns to Step 1 and guesses another value for ϕ_t.

There are $2^{6w} = 2^{96}$ possible values for ϕ_t, so performing Steps 1 to 19 is computationally equivalent to $c(2^{96})$ encryptions. As the NLF is balanced, only one in $2^w = 2^{16}$ values of ϕ_t will pass the NLF check. Thus, only 2^{80} values of ϕ_t will proceed to Step 20. There are 2^{16} values for u_{t+11}, so Steps 20 to 24 are performed $2^{80} \cdot 2^{16} = 2^{96}$ times: equivalent to $c(2^{96})$ encryptions. Therefore, the total complexity of the attack is equivalent to only $c(2^{96}) + c(2^{96}) = c(2^{96})$ encryptions.

Note 2. This attack clearly exploits the property that TIPSY has two pairs of NLF taps which are 5 words apart, contravening criteria suggested by Golic [6] and Löhlien [11].

3.1 Accounting for the Stuttering

The stuttering decimates the N-stream $\{v_t\}$ as follows. The first output of the NLF (v_1) is the first *stutter control word* (SCW). Each SCW is partitioned into eight pairs of bits (each pair is called a *dibit*). Beginning with the least significant dibit, the stuttering reads the value of the dibit and performs one of four actions according to the value of the dibit. The actions corresponding to the dibits are shown in Table 2. When all the dibits have been read, the LFSR is cycled, and the output of the NLF becomes the next SCW. The resulting stream, denoted $\{z_n\}$, is the key stream.

The stuttering decimates the N-stream in a random manner, so that consecutive key-stream words may or may not be consecutive N-stream words. This results in some uncertainty in relating the position of N-words to position of key-stream words. Furthermore, this uncertainty increases with the distance (in words) between key-stream words. This helps defeat attacks which require large amounts of key stream, such as correlation attacks. However, the stuttering does not add much resistance against GD attacks.

Example 4. Consider the attack in Example 3. This attack requires the attacker to know the values of v_{t+i}, $i \in \{4, 7, 11, 12, 17, 18, 22, 23\}$. To perform this attack, the attacker must assume that at a certain point in the key stream, one or more SCWs have a particular value or values which allow the appropriate N-words to be obtained from the key stream. Given a suitably large amount of key stream, an attacker can assume that for some values of t, $v_{t+3} = (01, 10, ab, 01, 10, cd, 10, 01)$

Table 2. The actions of the stuttering corresponding to the four possible values of the dibits.

00:	Cycle the LFSR, but do not output anything.
01:	Cycle the LFSR, output the NLF output XOREd with 0x6996, then cycle the LFSR again (without producing another output).
10:	Cycle the LFSR once (without producing any output), then cycle the LFSR again and output the NLF output.
11:	Cycle the LFSR and output the NLF output XORed with the bit-wise complement of 0x9669.

where $ab, cd \in \{01, 10\}$, and v_{t+3} is an SCW. The key stream output by this SCWs will be:

$$\begin{aligned} z_n &= v_{t+4} \oplus \text{0x6996}, & z_{n+1} &= v_{t+7}, \\ z_{n+2} &= v_{t+8} \oplus \text{0x6996 OR } z_{n+2} = v_{t+9}, & & \\ z_{n+3} &= v_{t+11}, & z_{n+4} &= v_{t+12} \oplus \text{0x6996}, \\ z_{n+5} &= v_{t+14} \oplus \text{0x6996 OR } z_{n+5} = v_{t+15}, & & \\ z_{n+6} &= v_{t+17}, & z_{n+7} &= v_{t+18} \oplus \text{0x6996}, \end{aligned}$$

The next SCW will v_{t+20}. The attacker can assume that for some value of t, not only is v_{t+3} of the above form, but v_{t+20} is also of the form $v_{t+20} = (\ldots, 01, 10)$, If this is the case, then the next key-stream words are $z_{n+8} = v_{t+22}$ and $z_{n+9} = v_{t+23} \oplus \text{0x6996}$.

Thus, assuming that the values of the SCWs are correct, the attacker is able to determine the N-words from the key stream, and perform the attack in Example 3. There are two obstacles. First, the attacker does not know when the SCWs have these values, and second, the attacker does not even know where in the key stream the SCWs occur. As a result, the attacker proceeds through the key stream assuming that each sequence of 10 key-stream words was derived from the N-stream using the SCWs in Example 4, and performs the steps in Example 3 until the correct state is found. Let N denote the data complexity, equal to the number of times that the process in Example 3 is repeated. The expected value of N is the inverse of the probability that a random portion of key stream was obtained from the N-stream using the SCWs in Example 4. This probability is determined as follows. Firstly, consider the probability that the first key-stream word is the first word output by an SCW. There are an average of 6 key-stream words output for every SCW, so this is 1/6. Secondly, ignoring the requirement that v_{t+4} be an SCW, the values of v_{t+4} and v_{t+20} are of the correct form (in this example) with probability 2^{-18}. The combined probability is $\frac{1}{6} \cdot 2^{-18} \approx 2^{-20.6}$. Consequently, $N = 2^{20.6}$ is the expected data complexity and the expected process complexity of the attack is $c(2^{20.6} \cdot 2^{96}) = c(2^{116.6})$. The GD attack on TIPSY exploiting only the LFSR and NLF (of process complexity

$c(2^{128})$, given the N-stream) would correspond to an attack of process complexity $c(2^{150})$, when considering the stuttering.

4 Searching for GD Attacks

This section provides a brief description of the authors' method of searching for GD attacks. In this section, the *tapset* of any polynomial $r(x) = \sum_{i=0}^{r+k} \epsilon_i x^i$, is defined to be $T[r(x)] = \{i : \epsilon_i \neq 0\}$, and the number of non-zero coefficients of $r(x)$ (equal to $|T[r(x)]|$) is called the *weight* of $r(x)$.[2] A GD attack is defined by a set of steps where the LFSR, the NLF and other multiples are D-exploited to determine a candidate state from a small set of candidate L-words. It is the tapsets (of the LFSR, NLF and multiples) that determine which candidate L-words can be determined from a given set of candidate L-words. Thus, the existence of a GD attack is determined by the tapsets of the LFSR, NLF and multiples, and not other details of the relationship such as the coefficients. In the case of a bit-wise LFSR, finding the tapsets for the multiples is simple because the tapsets of the the factors $p(x)$ and $q(x)$ define the polynomials and hence define the tapset of the product $r(x) = p(x) \cdot q(x)$. However, in a word-oriented LFSR, there can be many factors $q(x)$ with the same tapsets (but different coefficients) for which the products $p(x) \cdot q(x)$ have different tapsets. This adds significant complication to the search for GD attacks. In addition to this complication, there is a very large set of multiples (and their tapsets). Consequently, the task of searching for the optimal GD attack (the GD attack of lowest complexity) is still an open problem.

The search for GD attacks can be approached from two directions. One approach is to have a growing set of multiples to exploit, where the search program constantly tests for all multiples that can be D-exploited given the set of L-words that are currently known. This approach has not yet been implemented, although the authors are in the process of developing such a program.

The second approach divides the search into two parts: a *polynomial search*, that determines a set of multiples B to exploit; and a *B-attack search*, that examines the GD attacks exploiting the NLF and the polynomials in B. The set B is called an *GD basis* and is always assumed to contain $p(x)$.

4.1 The B-attack search.

The B-attack search finds a GD attack which minimises the complexity of the GD attacks exploiting the NLF and the polynomials in B. The B-attack search chooses a subset of L-words to guess, and finds the position of all L-words that could be determined by exploiting the NLF and the polynomials in B. If these L-words do not comprise a full state, then an additional L-word is guessed, and the process repeated. This continues until all L-words in an entire state are

[2] Note that D-exploiting $r(x)$ is computationally equivalent to at most $c(|T[r(x)]|)$ encryptions.

determined. Alternatively, if guessing an additional word will result in an attack with complexity larger than that of the best known attack, then the B-attack search tries another subset of L-words. To ensure that the B-attack search does not proceed indefinitely, the authors bounded the distance between the first L-word guessed and any determined L-words to a maximum of four register lengths.

4.2 The Polynomial Search.

The speed of the B-attack search decreases as the size of B increases, so the aim of the polynomial search is to find a small set of multiples that are likely to find the best attack. Intuition suggests that a multiple $r(x)$ is more likely to be D-exploited if the corresponding linear relationship is between a small number of L-words. That is, $r(x)$ is more likely to be exploited if it has low weight. Consequently, the first criterion used for selecting multiples for the set B is that they have low weight. Now, suppose that the polynomial search is considering adding a multiple $r(x)$ to B. Suppose that whenever $r(x)$ is D-exploited, some combination of multiples can be D-exploited to determine the same L-word. Such multiples are *redundant* and should not be added to B. Hence, the polynomial search looks for a set of low-weight, non-redundant multiples of $p(x)$. The polynomial search takes a polynomial $p(x)$, and two bounds D and W on the degree and weight of the polynomials to be added to the GD basis. The polynomial search looks through the multiples of degree $\leq D$ and with weight $\leq W$: any non-redundant multiples are added to the GD basis. The polynomial search fixes a tapset T' and considers the tapsets of $r(x) = p(x) \cdot q(x)$ when $T[q(x)] = T'$. Note that for a given T', all these multiples $r(x)$ will share some similar characteristics. There will be some coefficients of $r(x)$ which will be certain to be zero (in the *zero* positions), there will be some coefficients which will be certain to be nonzero (in the *nonzero* positions), and the remaining coefficients could be either zero or nonzero, depending on the cancellation of terms in the expansion of $p(x) \cdot q(x)$, (the *zero-or-nonzero* positions). From these sets of coefficients we can determine a superset of the possible tapsets for multiples $p(x) \cdot q(x)$ with $T[q(x)] = T'$, by considering all possible combinations of the nonzero positions and the zero-or-nonzero positions. The polynomial search only considers those tapsets with weight less than the bound W. For each resulting tapset, the polynomial search conducts tests for redundancy, and then confirm that the tapset corresponds to a multiple $p(x) \cdot q(x)$ with $T[q(x)] = T'$. This requires less processing than determining if the tapset corresponds to a multiple and then conducting the tests for redundancy.

The greatest restriction on the authors' polynomial search is the weight of the tested multiples. Our fastest algorithm employed fixed arrays containing the subsets of b elements from a set of a elements. This method worked best for us. As a and b increases, the necessary storage requirements increase significantly, placing constraints on a and b. The authors restricted the polynomial search to finding multiples of degree less than $2r$ (twice the degree of $p(x)$) and weight 10 or less. The tests for redundancy then reduced this set of multiples. Given these

restrictions, the polynomial search and B-attack search require less than a day of processing each.

4.3 Results

The polynomial search on the LFSR of TIPSY found 123 multiples within the above constraints (maximum degree $D = 26 = 2r$ and maximum weight $W = 10$). Using this basis, the B-attack search found an attack on TIPSY of complexity $c(2^{96})$ (ignoring stuttering): this is the attack described in Example 3. Given the improved attack on TIPSY, the authors considered that t-class might also weaker than first claimed. A polynomial search on the LFSR of t16 was conducted to find the GD basis B within the aforementioned constraints (maximum degree $D = 34 = 2r$ and maximum weight $W = 10$). This search revealed a GD basis of 63 multiples. The B-attack search using this basis found only GD-attacks of complexity $c(2^{160})$ (ignoring stuttering). Such attacks offer no improvement over previous GD attacks (such attacks are simple variants of the attacks in [2, 7], discussed in [8]). A similar analysis of t8 and t32 revealed that the additional linear relationships did not provide an attack of lower complexity than was already known.

5 Conclusion

This paper provides two examples of how multiples can be exploited in attacks against various word-oriented ciphers. In the first example, powers of the bitwise connection polynomial reveal a weakness in SSC-II. This supports the well-known criteria that stream ciphers (even word-oriented stream ciphers) should avoid using connection polynomials for which there exists low-degree, low-weight multiples. In the second example, multiples of the connection polynomial over $GF(2^w)$ are used in a low complexity GD attack against a dummy SOBER-like cipher, TIPSY. However, the t-class ciphers appear to resist attacks exploiting multiples. The authors continue to examine how multiples can be exploited against SOBER-like ciphers, and consider how SOBER-like ciphers resist such attacks. It is hoped that this will lead to a method of determining the best possible GD attack on a given SOBER-like cipher.

References

1. S. Blackburn, S. Murphy, F. Piper, and P. Wild. A SOBERing remark. Technical report, Information Security Group, Royal Holloway University of London, Egham, Surrey TW20 0EX, U.K., 1998.
2. D. Bleichenbacher, W. Meier, and S Patel. Analysis of the SOBER stream cipher. Technical Report TR45.AHAG.08.30.12, TR45 Ad Hoc Authentication Group, 1999.
3. D. Bleichenbacher and S Patel. SOBER cryptanalysis. *Fast Software Encryption, FSE'99 Lecture Notes in Computer Science, vol. 1636, L. Knudsen ed., Springer-Verlag*, pages 305–316, 1999.

4. V. Chepyzhov and B. Smeets. On a fast correlation attack on certain stream ciphers. *Advances in Cryptology, EUROCRYPT'91, Lecture Notes in Computer Science, vol. 547, D. W. Davies ed., Springer-Verlag*, pages 176–185, 1991.
5. J. Golić, A. Clark, and E. Dawson. Inversion attack and branching. *Information Security and Privacy, Fourth Australasian Conference, ACISP'99, Lecture Notes in Computer Science, vol. 1587, J. Pieprzyk, R Safavi-Naini, J. Seberry eds., Springer-Verlag*, pages 88–102, 1999.
6. J. Dj. Golić. On the security of nonlinear filter generators. *Fast Software Encryption, Lecture Notes in Computer Science, vol. 1039, D. Gollmann ed., Springer*, pages 173–188, 1995.
7. P. Hawkes. An attack on SOBER-II. Technical report, QUALCOMM Australia, Suite 410, Birkenhead Point, Drummoyne NSW 2137, Australia, 1999.
8. P. Hawkes and G. Rose. The t-class of SOBER stream ciphers. Technical report, QUALCOMM Australia, Suite 410, Birkenhead Point, Drummoyne NSW 2137, Australia, 1999. See `http://www.home.aone.net.au/qualcomm`.
9. T. Herlestam. On functions of Linear Shift Register Sequences. *Advances in Cryptology, EUROCRYPT'85, Lecture Notes in Computer Science, vol. 219, F. Pichler ed., Springer-Verlag*, 1986.
10. T. Johansson and F Jönsson. Improved fast correlation attacks on stream ciphers via convolutional codes. *Advances in Cryptology, EUROCRYPT'99, Lecture Notes in Computer Science, vol. 1592, J. Stern ed., Springer-Verlag*, pages 347–362, 1999.
11. B. Löhlein. Analysis and modifications of the conditional correlation attack. 1999. Accepted at 3rd IEEE/ITG Conference on Source and Channel Coding, 17-19 Jan. 2000, Munich.
12. G. Rose. S32: A fast stream cipher based on linear feedback over $GF(2^{32})$. Technical report, QUALCOMM Australia, Suite 410, Birkenhead Point, Drummoyne NSW 2137, Australia, 1998.
13. G. Rose. SOBER: A stream cipher based on linear feedback over $GF(2^8)$. Technical report, QUALCOMM Australia, Suite 410, Birkenhead Point, Drummoyne NSW 2137, Australia, 1998. See `http://www.home.aone.net.au/qualcomm`.
14. G. Rose. A stream cipher based on linear feedback over $GF(2^8)$. *Information Security and Privacy, Third Australasian Conference, ACISP'98, Lecture Notes in Computer Science, vol. 1438, C. Boyd, E. Dawson eds., Springer-Verlag*, pages 135–146, 1998.
15. M. Zhang, C. Carroll, and A. Chan. SSC. Technical Report TR45.AHAG.99.02.09.15, TR45 Ad Hoc Authentication Group, 1999.
16. M. Zhang, C. Carroll, and A. Chan. The software-oriented stream cipher SSC-II. In *Proceedings of Fast Software Encryption Workshop 2000*, pages 39–56, 2000.