

# An Attack on the Stream Cipher Whiteout

Lillian Kråkmo

Department of Mathematical Sciences  
Norwegian University of Science and Technology  
Trondheim, Norway  
krakmo@math.ntnu.no

Marie Elisabeth Gaup Moe

Centre for Quantifiable Quality of Service in Communication Systems  
Norwegian University of Science and Technology  
Trondheim, Norway  
marieeli@q2s.ntnu.no

**Abstract**—The stream cipher Whiteout is presented and an attack on Whiteout is proposed. This attack involves an active adversary, and recovers 74 bits of the initial states of all LFSRs involved, by using in worst case 30000 ciphertexts of length 2. The attack relies on the fact that the initialization procedure of Whiteout is linear. Whiteout has a secret key of 120 bits and our attack recovers this key by an exhaustive search of only 49 bits.

**Index Terms**—Stream ciphers, linear keyloading, Whiteout, keystream generator.

## I. INTRODUCTION

Linear feedback shift registers (LFSRs) are popular building blocks in pseudorandom bit generators, because of the nice statistical properties of the generated sequences and the practical implementation in hardware. Such generators can be used to produce keystreams for streamciphers. The keystream generator is typically loaded with a secret key which is used for several encryptions, and an initialization vector that is different for each encryption. It is important that the loading procedure of the LFSRs is implemented with care, so that attacks are prevented. In particular the secret key and the initialization vector should not be linearly combined together in the initial state of the generator. In this paper we will demonstrate this by proposing an attack on the keystream generator Whiteout exploiting this weakness. Other attacks exploiting the similarly weak initialization procedure of the stream cipher used in Bluetooth are for example found in [1] and [2].

## II. THE STREAM CIPHER WHITEOUT

In this section a description of the stream cipher encryption algorithm Whiteout will be given. This algorithm was originally designed for implementation in a dedicated cryptographic chip that could be exported outside NATO countries. However the cipher was never used, discarded, and handed over to us by Thales Communications AS so that we could use it in this study. It should be noted that the cipher is intentionally weak. All details about Whiteout given in this section are found in the up to now unpublished specification of Whiteout.

### A. Output mode

The key generator of Whiteout shown in Fig. 1 consists of two parts of irregularly clocked, maximal length LFSRs, the block of 3 X-LFSRs and the network of 8 R-LFSRs defined in

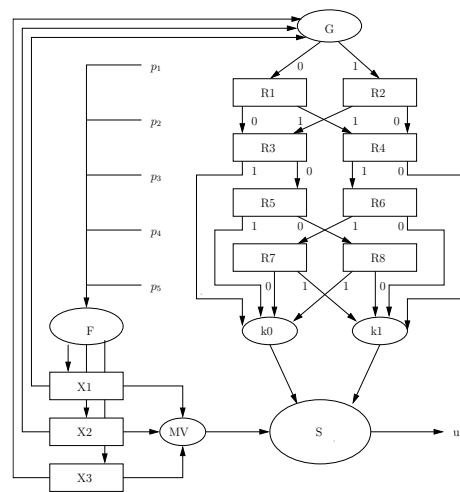


Fig. 1. Whiteout key generator in output mode

TABLE I  
DEFINITION OF THE LFSRS

LFSR	Characteristic polynomial
X1	$1 + x^3 + x^7 + x^{13} + x^{17}$
X2	$1 + x^5 + x^7 + x^{14} + x^{19}$
X3	$1 + x^3 + x^7 + x^{16} + x^{23}$
R1	$1 + x^2 + x^5 + x^8 + x^{11}$
R2	$1 + x^2 + x^6 + x^8 + x^{11}$
R3	$1 + x^2 + x^7 + x^9 + x^{11}$
R4	$1 + x^2 + x^7 + x^{10} + x^{11}$
R5	$1 + x^2 + x^9 + x^{10} + x^{11}$
R6	$1 + x^3 + x^4 + x^{10} + x^{11}$
R7	$1 + x^3 + x^5 + x^7 + x^{11}$
R8	$1 + x^3 + x^5 + x^8 + x^{11}$

Table I. These two parts mutually control the clocking of each other via the functions  $G$  and  $F$ . The  $G$  function takes 6 input bits from the X-LFSRs and outputs one bit that decides the start of the path through the network of R-LFSRs that starts in the “node”  $G$  and ends up in one of the “leaves”  $k0$  or  $k1$ . Let  $x_{i,j}$  denote the bit in position  $j$  in the  $X_i$ -LFSR. The input bits to the  $G$  function are:

$$\begin{aligned} b_1 &= x_{1,8} & b_3 &= x_{2,9} & b_5 &= x_{3,11} \\ b_2 &= x_{1,16} & b_4 &= x_{2,18} & b_6 &= x_{3,22}. \end{aligned}$$

TABLE II  
H FUNCTION TAPPING POSITIONS

R-LFSR	x	y	z
R1	2	5	7
R2	10	8	3
R3	6	9	4
R4	3	7	8
R5	8	4	2
R6	4	10	6
R7	5	2	9
R8	7	3	5

TABLE III  
F PERMUTATION

$v$	0	1	2	3	4	5	6	7
$u$	20	19	17	31	11	8	29	14
$v$	8	9	10	11	12	13	14	15
$u$	24	30	15	22	28	25	2	18
$v$	16	17	18	19	20	21	22	23
$u$	21	6	13	26	3	23	7	1
$v$	24	25	26	27	28	29	30	31
$u$	9	27	12	0	5	4	16	10

TABLE IV  
F CLOCKING OF X-LFSRS

$u$	0	1	2	3	4	5	6	7
$t1$	1	0	1	1	0	0	1	2
$t2$	2	1	0	0	2	1	2	1
$t3$	0	2	2	2	1	2	0	0
$u$	8	9	10	11	12	13	14	15
$t1$	2	2	2	2	2	1	1	2
$t2$	1	0	1	1	0	2	2	0
$t3$	0	1	0	0	1	0	0	1
$u$	16	17	18	19	20	21	22	23
$t1$	0	1	1	1	0	2	0	1
$t2$	2	0	2	0	1	1	2	2
$t3$	1	2	0	2	2	0	1	0
$u$	24	25	26	27	28	29	30	31
$t1$	1	2	2	2	1	0	2	0
$t2$	2	1	1	0	2	1	1	2
$t3$	0	0	0	1	0	2	0	1

The output of  $G$  is given by:

$$\begin{aligned}
 G(b_1, b_2, b_3, b_4, b_5, b_6) &= b_2 \oplus b_6 \oplus b_4 b_5 \oplus \\
 &b_5 b_6 \oplus b_3 b_4 \oplus b_2 b_5 \oplus b_1 b_5 \oplus b_1 b_2 \oplus b_4 b_5 b_6 \oplus \\
 &b_3 b_5 b_6 \oplus b_3 b_4 b_5 \oplus b_2 b_5 b_6 \oplus b_2 b_4 b_6 \oplus b_2 b_4 b_5 \oplus \\
 &b_2 b_3 b_6 \oplus b_1 b_3 b_5 \oplus b_1 b_3 b_4 \oplus b_1 b_2 b_5 \oplus b_1 b_2 b_3 b_4 \oplus \\
 &b_1 b_2 b_4 b_5 \oplus b_1 b_2 b_4 b_6 \oplus b_1 b_3 b_4 b_6 \oplus b_2 b_3 b_4 b_6.
 \end{aligned}$$

The output of  $G$  determines which of the LFSRs  $R1$  or  $R2$  that will be enabled and defines the start of a path through the network of R-LFSRs that will be calculated during each cycle in output mode. The path through the network of R-LFSRs is defined as a binary 5-tuple  $(p_1, p_2, p_3, p_4, p_5)$ , where  $p_4$  and  $p_5$  may be undefined,  $p_1$  is the output of  $G$ ,  $p_2$  is the output of  $R1$  or  $R2$  dependent on which LFSRs that was enabled by  $p_1$ ,  $p_3$  is the output of  $R3$  or  $R4$ , and so on. The path gives input to the clocking function  $F$  and decides which R-LFSRs that will be clocked during the cycle. The path also determines the value of the leaves  $k0$  and  $k1$ .  $k0$  and  $k1$  are initially set to 0 in the start of each cycle, when the path ends in one of them, the value of the terminating leaf is changed to 1.

The output of the R-LFSRs is given by the function  $h$  that takes 3 input bits  $x$ ,  $y$  and  $z$  from the R-LFSRs and outputs one bit that determines which R-LFSR next to enable according to Fig. 1. The tapping positions for the input to  $h$  for the respective R-LFSRs are given in Table II, the output of  $h$  is given by:

$$h(x, y, z) = x \oplus z \oplus xy.$$

The function  $F$  decides the clocking of the X-LFSRs and is computed after the path through the R-LFSRs is calculated.  $F$  takes 5 input bits and outputs a triple  $(t_1, t_2, t_3)$ ,  $t_i \in \{0, 1, 2\}$  where  $t_i$  counts the number of steps the LFSR  $X_i$  is going to be clocked. Let  $r_{i,j}$  denote the bit in position  $j$  in the  $R_i$ -LFSR. The input bits to  $F$  are:

$$\begin{aligned}
 c_1 &= p_1 & c_4 &= r_{5,10} \oplus r_{7,8} \\
 c_2 &= p_2 & c_5 &= r_{6,10} \oplus r_{8,8}, \\
 c_3 &= p_3
 \end{aligned}$$

where  $p_1$ ,  $p_2$  and  $p_3$  are the first, second and third bit in the calculated path through the R-LFSRs. From these input bits

a corresponding integer value  $v \in \{0, 1, \dots, 31\}$  is calculated as:

$$v = 16c_1 + 8c_2 + 4c_3 + 2c_4 + c_5.$$

$F$  is defined by a permutation of the possible values for  $v$  given in Table III, the  $u$ -value is then the input to the look-up table given in Table IV, which gives the stepping numbers.

The function  $MV$  (majority vote function) outputs the selector bit  $s$ :

$$s = MV(x, y, z) = xy \oplus xz \oplus yz,$$

where  $x = x_{1,0}$ ,  $y = x_{2,0}$  and  $z = x_{3,0}$ . If  $s = 0$  the produced key bit  $u = k0$ , else  $u = k1$ . Since  $k0$  and  $k1$  always will be the inverse of each other at the time when the key bit is calculated, the produced key bit can be written as

$$u = s \oplus k0.$$

To sum up, one cycle of the key generator of Whiteout in output mode is defined by the following steps:

- 1) Compute the output of  $G$
- 2) Compute the path through the network of R-LFSRs
- 3) Compute  $MV$  to find  $s$
- 4) Compute the keystream bit  $u$

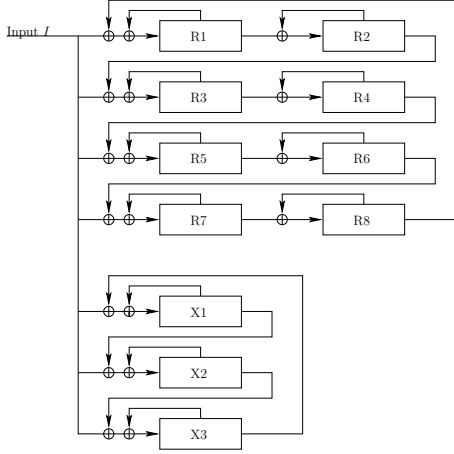


Fig. 2. Whiteout key generator in input mode

- 5) Compute  $F$  to get the stepping of the X-LFSRs
- 6) Step the X-LFSRs according to computed data
- 7) Step the R-LFSRs that were enabled in the path.

### B. Input mode

In input mode all the LFSRs are connected together as shown in Fig. 2 for loading of the input data consisting of the 120 bit secret key variable (KV) and the 80 bit message indicator (MI). The input mode of Whiteout can also be used for generating a message authentication code (MAC).

Let  $I$  be the data input signal, let  $f_{ri}/f_{xi}$  be the internal feedback of the LFSRs  $R_i/X_i$  as defined by the characteristic polynomials and let  $FR_i/FX_i$  denote the full feedback of the LFSRs in input mode. The feedback bits in input mode are then given by:

$$\begin{aligned}
 FR1 &= I \oplus fr1 \oplus r_{8,0} & FR2 &= fr2 \oplus r_{1,0} \\
 FR3 &= I \oplus fr3 \oplus r_{2,0} & FR4 &= fr4 \oplus r_{3,0} \\
 FR5 &= I \oplus fr5 \oplus r_{4,0} & FR6 &= fr6 \oplus r_{5,0} \\
 FR7 &= I \oplus fr7 \oplus r_{6,0} & FR8 &= fr8 \oplus r_{7,0} \\
 FX1 &= I \oplus fx1 \oplus x_{3,0} \\
 FX2 &= I \oplus fx2 \oplus x_{1,0} \\
 FX3 &= I \oplus fx3 \oplus x_{2,0}
 \end{aligned}$$

The loading of the LFSRs is done in input mode in the following way:

- 1) All the LFSRs are initially filled with ones
- 2) Load the 120 bits of KV into the LFSRs starting with the first bit
- 3) Load 8 zero bits
- 4) Load the 80 bits of MI starting with the first bit
- 5) Load 102 zero bits

After the loading procedure the key generator switches to output mode and generates one key bit during each cycle of output mode, that can be used for encryption and decryption. If an LFSR enters output mode in an all-zero state, the feedback bit of this LFSR is forced to be 1 until the LFSR is clocked and this feedback bit enters the register.

The generation of a MAC of a given length  $n$  on a given binary message is done in the following way:

- 1) Load the KV and MI according to the procedure described above
- 2) Continue loading the message into the LFSRs using input mode
- 3) Load 102 zeros
- 4) Switch to output mode
- 5) The  $n$  first produced bits in output mode is the MAC on the message.

### III. THE INITIALIZATION PROCEDURE

According to the description of Whiteout, the key and the message indicator are loaded into the LFSRs in a linear manner during input mode. In a scenario where only the key is kept secret, this property implies vulnerability to the generator. If we let  $(s_0, \dots, s_{146})$  denote the initial state of the LFSRs as the generator enters output mode, and view the key bits as indeterminants, we can construct a linear system of equations:

$$\begin{aligned}
 f_0(x_1, \dots, x_{120}) &= s_0 \\
 &\vdots \\
 f_{146}(x_1, \dots, x_{120}) &= s_{146}.
 \end{aligned}$$

We note that the  $f_i$  depend on the message indicator bits. This system can be represented in matrix form by

$$\mathbf{Ax} = \mathbf{b},$$

where  $A$  is a  $147 \times 120$  coefficient matrix,  $\mathbf{x} = (x_1, \dots, x_{120})^T$ ,  $\mathbf{b} = (b_0, \dots, b_{146})^T$ , and  $b_i = c_i + s_i$ , where  $c_i$  is the constant term of equation  $f_i$ . Consequently, if the adversary succeeds in recovering the initial state of the LFSRs, the key bits can be found by Gaussian elimination of the above system.

We may also consider the situation where parts of the initial state are recovered. More precisely, the adversary has knowledge of  $s_I = \{s_i \mid i \in I\}$ , where  $I$  is a set of indices and  $|I| \leq 147$ . Let  $\text{row}(A, i)$  denote the  $i$ th row of  $A$ , and let  $A_I$  be the matrix formed by only including the rows  $\{\text{row}(A, i) \mid i \in I\}$ . Furthermore, let  $\mathbf{b}_I$  be the corresponding vector of constant terms. In this case, a linear system given by

$$A_I \mathbf{x} = \mathbf{b}_I \quad (1)$$

can be obtained. As usual,  $\text{rank}(A_I)$  denotes the number of linearly independent rows of  $A_I$ . Clearly, since  $\mathbf{x}$  has length 120,  $\text{rank}(A_I) \leq 120$ . If  $\text{rank}(A_I) = 120$ , then all of the key bits can be recovered by Gaussian elimination as in the above situation. Generally,  $A_I$  has a null space of dimension  $120 - \text{rank}(A_I)$ , which means that the adversary can recover the key by exhaustive search of  $2^{120 - \text{rank}(A_I)}$  candidates.

Next we consider a scenario where the adversary has the ability to manipulate the message indicator, i.e. to complement a chosen set of the involved bits. This may be a realistic scenario if the encryption device is physically available to the adversary, for instance as a smart card.

By viewing both the key bits  $x_1, \dots, x_{120}$  and the message indicator bits  $m_1, \dots, m_{80}$  as indeterminants, we can construct the following linear system:

$$\begin{aligned} f'_0(x_1, \dots, x_{120}, m_1, \dots, m_{80}) &= s_0 \\ &\vdots \\ f'_{146}(x_1, \dots, x_{120}, m_1, \dots, m_{80}) &= s_{146} \end{aligned}$$

Because of the properties of linear functions, this system can be represented in matrix form by

$$A\mathbf{x} + B\mathbf{m} = \mathbf{b}',$$

where  $B$  is a  $147 \times 80$  coefficient matrix,  $\mathbf{m} = (m_1, \dots, m_{80})$ ,  $\mathbf{b}' = (b'_0, \dots, b'_{146})$  and  $b'_i = c'_i + s_i$ , where  $c'_i$  is the constant term of the function  $f'_i$ . In the assumed scenario, the adversary may add an arbitrary vector  $\mathbf{m}' \in GF(2)^{80}$  to  $\mathbf{m}$ , with the effect of complementing the bits of  $\mathbf{m}$  corresponding to the elements of  $\mathbf{m}'$  equal to 1. The resulting equation is

$$A\mathbf{x} + B(\mathbf{m} + \mathbf{m}') = \mathbf{b}' + B\mathbf{m}' = \mathbf{c}' + (\mathbf{s} + B\mathbf{m}'), \quad (2)$$

where the last equality comes from expressing  $\mathbf{b}'$  as  $\mathbf{c}' + \mathbf{s}$ . Consequently, the adversary may calculate the effect on the initial state bits  $s_i$  from adding  $\mathbf{m}'$  to  $\mathbf{m}$ . An interesting question is whether the adversary is able to manipulate the initial state bits in a predefined manner. With this in mind, we consider the equation

$$B\mathbf{m}_i = \mathbf{e}_i, \quad (3)$$

where  $\mathbf{e}_i$  is the unit vector  $(e_{i,0}, \dots, e_{i,146})$  with  $e_{i,i} = 1$  and  $e_{i,j} = 0$  for all  $j$  such that  $i \neq j$ . If such an  $\mathbf{m}_i$  is found, replacing  $\mathbf{m}'$  by  $\mathbf{m}_i$  in (2) gives

$$A\mathbf{x} + B(\mathbf{m} + \mathbf{m}_i) = \mathbf{c}' + (\mathbf{s} + \mathbf{e}_i),$$

meaning that the bit  $s_i$  is complemented, while the rest of the initial state bits are unaltered. To predict whether such an  $\mathbf{m}_i$  exists for a given  $i$ , we simply have to solve the corresponding system of equations. A solution of the equation (3) exists if and only if the vector  $\mathbf{e}_i$  belongs to the column space of the matrix  $B$ , denoted by  $\text{Col}(B)$ . The dimension of  $\text{Col}(B)$  is upper bounded by 80 so (3) does not have a solution for every  $i$ ,  $0 \leq i \leq 146$ . From this we can conclude that the described technique of complementing bits of the message indicator does not allow the adversary to manipulate every bit of the initial state independently of each other; this can only be done for the  $s_i$  such that (3) is consistent. An alternative approach is to look for a subset  $s_I$  of initial state bits such that all of the bits in  $s_I$  may be complemented independently of each other, without regard to the bits outside the subset. One way of doing this is to search for an  $s_I$  such that the corresponding matrix  $B_I$  formed by only including the rows  $\{\text{row}(B, i) \mid i \in I\}$  has full rank. Let  $\mathbf{e}_{Ii}$  be the vector formed from  $\mathbf{e}_i$  in an analogous manner, and consider the equation

$$B_I \mathbf{m}_{Ii} = \mathbf{e}_{Ii}. \quad (4)$$

We note that  $\mathbf{m}_{Ii}$  has the same number of elements as  $\mathbf{m}_i$ , with values characteristic for the subset  $s_I$ . The advantage of

this method is that equation (4) is consistent for every  $i \in I$ , since  $\text{Col}(B_I)$  has full dimension and thus includes  $\mathbf{e}_i$  for every  $i$ . In other words, the ability to complement the bits in  $s_I$  independently of each other is guaranteed without solving equation (4) for every  $i$ . Of course, solving the equations is still necessary in order to obtain the vectors  $\mathbf{m}_{Ii}$ , by which the complementing is performed.

#### IV. AN ATTACK ON WHITEOUT

In this section we sketch an attack on Whiteout, under the assumption that we are able to manipulate the message indicator, i.e. to complement a chosen subset of the involved bits. As suggested by the previous section, the power of the attack depends on our ability to complement the initial state bits in a predefined way. In fact, each procedure of the proposed attack requires that bits can be complemented independently within a specific subset  $s_I$  of the initial state bits. This is possible if and only if the corresponding matrix  $B_I$  has full rank, which must be verified for all matrices used. Due to this, the procedures of the attack are constructed in an ad hoc manner, based on whether the tested subsets  $s_I$  were found to have the required property or not. We have verified that the matrix  $B$  has rank 80. This implies that the maximum number of initial state bits which can be complemented independently is 80.

In the following,  $f^{(t)}$  is the output function of Whiteout, i.e. the function calculating the keystream bits  $u^{(t)}$ , given the state of the generator after  $t$  time steps. The knowledge of  $f^{(t)}$  is critical to the attack. Its algebraic normal form is derived from the description of Whiteout in Section II. For simplicity, we denote the output of the function  $G$  at the  $t$ th clocking by  $g^{(t)}$ . The outputs at time  $t$  of the function  $h$  for the respective LFSRs R1, ..., R8 are denoted by  $r_1^{(t)}, \dots, r_8^{(t)}$ , the output of the function  $MV$  is denoted by  $s^{(t)}$ , and the value of  $k0$  is written as  $k0^{(t)}$ . For a general binary variable  $v$ ,  $\bar{v} = v + 1$ , i.e. the complemented  $v$ . The keystream bit  $u^{(t)}$  is consequently given by

$$\begin{aligned} u^{(t)} &= f^{(t)}(g^{(t)}, r_1^{(t)}, \dots, r_8^{(t)}, s^{(t)}) \\ &= s^{(t)} \oplus k0^{(t)} \\ &= s^{(t)} \oplus \bar{g}^{(t)} \bar{r}_1^{(t)} r_3^{(t)} \oplus g^{(t)} r_2^{(t)} r_3^{(t)} \\ &\quad \oplus \bar{g}^{(t)} \bar{r}_1^{(t)} \bar{r}_3^{(t)} r_5^{(t)} \oplus g^{(t)} r_2^{(t)} \bar{r}_3^{(t)} r_5^{(t)} \\ &\quad \oplus \bar{g}^{(t)} r_1^{(t)} r_4^{(t)} r_6^{(t)} \bar{r}_7^{(t)} \oplus g^{(t)} \bar{r}_2^{(t)} r_4^{(t)} r_6^{(t)} \bar{r}_7^{(t)} \\ &\quad \oplus \bar{g}^{(t)} \bar{r}_1^{(t)} \bar{r}_3^{(t)} \bar{r}_5^{(t)} r_8^{(t)} \oplus g^{(t)} r_2^{(t)} \bar{r}_3^{(t)} \bar{r}_5^{(t)} r_8^{(t)}. \end{aligned} \quad (5)$$

We note that according to Fig. 1, the 8 terms representing  $k0^{(t)}$  correspond to the 8 respective paths leading to  $k0$  in the network. In the attack,  $f^{(0)}$  is represented by the initial state bits,  $s_1^{(0)}, \dots, s_{146}^{(0)}$ , while  $f^{(1)}$  is represented by the state bits after the first clocking, denoted by  $s_1^{(1)}, \dots, s_{146}^{(1)}$ .

The attack is divided into two stages. During the first stage, we focus on the initial state bits that determine the first output,  $u^{(0)}$ , which is computed before the LFSRs are clocked for the first time. The inputs to  $f^{(0)}$  constitute a relatively small subset of the initial state bits, in which all

the bits can be complemented independently of each other. By complementing these bits in a systematic manner, and observing how  $u^{(0)}$  is affected, we are able to recover all the inputs to  $f^{(0)}$ .

During the second stage, we focus on the initial state bits that are involved in the second output,  $u^{(1)}$ , which is computed after the first clocking. In principle, the procedure for recovering these bits is the same as for the first stage, but several modifications are required.

We note that since only some of the involved LFSRs are stepped during the first clocking, many of the input bits to  $f^{(1)}$  are exactly the same as the inputs to  $f^{(0)}$ , so the number of new bits to be obtained is strictly limited. In order to recover more of the initial state bits, we may repeat the second stage several times, with an intentionally modified first clocking.

Assume that we are able to recover a subset  $s_I$  of the initial state bits. As described in Section III, we may perform Gaussian elimination on the corresponding matrix  $A_I$ , and then recover the key by exhaustive search of  $2^{120-\text{rank}(A_I)}$  candidates. This constitutes the final part of the attack.

#### A. Stage 1

We proceed by describing the overall strategy for the first stage of the attack, where we concentrate on the calculation of the first output  $u^{(0)}$ . The construction of the involved algorithms will be explained in detail later on.

- 1) Find  $g^{(0)}$  by the algorithm Findg.
- 2) Find the inputs to  $G$ ,  $b_1^{(0)}, \dots, b_6^{(0)}$  by one of the search trees GTree0 or GTree1 together with Findg.
- 3) Find the path through the network of R-LFSRs by the search tree FindPath. If the path satisfies a certain criterion, then continue. If not, then repeat the steps 1-3 with a modified message indicator.
- 4) Find  $r_1^{(0)}, \dots, r_8^{(0)}$  by the search tree RTree.
- 5) Find the inputs to  $h$  for the respective R-LFSRs,  $x_{R1}^{(0)}, y_{R1}^{(0)}, z_{R1}^{(0)}, \dots, x_{R8}^{(0)}, y_{R8}^{(0)}, z_{R8}^{(0)}$ , by the search tree hTree together with RTree.
- 6) Find the inputs to  $MV$ , denoted as  $x_{MV}^{(0)}, y_{MV}^{(0)}$  and  $z_{MV}^{(0)}$ , by the search tree MVTree.

We have verified that all the initial state bits involved in  $u^{(0)}$  can be complemented independently. More precisely, we have constructed the matrix  $B_I$  corresponding to the subset  $s_I$  of all the involved bits, and found  $B_I$  to have full rank by Gaussian elimination. The corresponding vectors  $\mathbf{m}_{Ii}$ , which upon addition to the message indicator  $m$  complements the initial state bit  $s_i$ , are found by solving equations of the form

$$B_I \mathbf{m}_{Ii} = \mathbf{e}_{Ii},$$

as described in the previous section.

The algorithm Findg exploits that, due to the construction of  $h$ , the respective variables  $r_1^{(0)}, \dots, r_8^{(0)}$  can be complemented independently by simply complementing  $z_{R1}^{(0)}, \dots, z_{R8}^{(0)}$ . This allows us to manipulate  $r_1^{(0)}, \dots, r_8^{(0)}$  in a systematic manner. By observing  $u^{(0)}$  after each manipulation, we can determine whether  $k0^{(0)}$  has been complemented or not. Findg executes

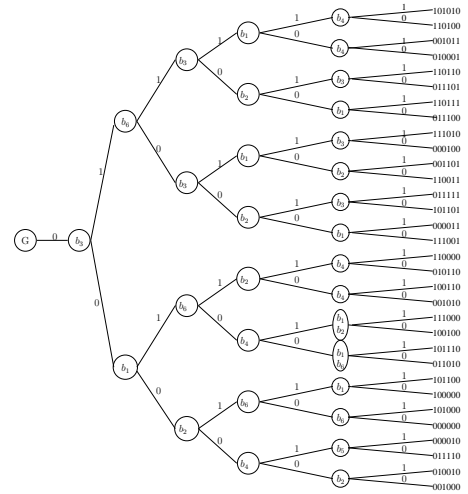


Fig. 3. Searchtree for finding  $b_1, b_2, b_3, b_4, b_5$  and  $b_6$  when the output of  $G$  is 0

an adapted series of manipulations, such that the value of  $g^{(0)}$  can be recovered within a relatively small number of steps.

When  $g^{(0)}$  is recovered, the next step is to determine  $b_1^{(0)}, \dots, b_6^{(0)}$ . This is done by searching GTree0 or GTree1, according to the value of  $g^{(0)}$ . The search tree GTree0 is given as Fig. 3. Each node is characterized by one or several of the  $b_i$ . During the search, the following procedure is to take place at each node:

- 1) Complement the  $b_i^{(0)}$  corresponding to the  $b_i$  of the node.
- 2) Use Findg to find the value of  $g^{(0)}$  resulting from the manipulation of the message indicator.
- 3) Choose the edge corresponding to the new value of  $g^{(0)}$ , and move on to the next node. The  $b_i$  complemented in step 1 should not be complemented back.

The numbers given at each end node are the respective values of the inputs  $b_1, \dots, b_6$ .

The search tree FindPath uses a similar strategy as Findg, but it involves more steps, as it recovers the entire path through the network of R-LFSRs. Due to its size FindPath is not displayed in this paper. We note that since FindPath detects the value of  $k0^{(0)}$ , it also reveals the value of  $s^{(0)}$ . The criterion introduced on the path is that it should not involve any of the LFSRs  $R5, R6, R7$  and  $R8$ . The purpose of this criterion is explained in our discussion of the second stage of the attack. By the assumed attack scenario, we are able to repeat the steps 1-3, with a different MI each time, until the path satisfies the criterion.

The next step is to recover all of the  $r_1^{(0)}, \dots, r_8^{(0)}$ , by searching RTree. This tree is based on the output function  $f^{(0)}$ . To ensure a fairly balanced tree when constructing it, we introduced criteria on the partitions, which were modified in an ad hoc manner, until a complete search tree was obtained. Due to its large size, RTree is not displayed in this paper.

Once every output from  $h$  is recovered, the respective inputs  $x_{R1}^{(0)}, y_{R1}^{(0)}, z_{R1}^{(0)}, \dots, x_{R8}^{(0)}, y_{R8}^{(0)}, z_{R8}^{(0)}$  are obtained by searching hTree, given as Fig. 4. During the search, the edges are chosen

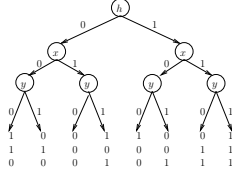


Fig. 4. Searchtree for finding the inputs to the  $h$  function

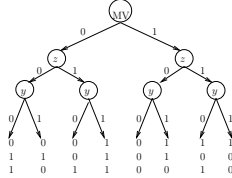


Fig. 5. Searchtree for finding the inputs to the  $MV$  function

according to the resulting value of  $r_i^{(0)}$ , which is recovered by RTree. The numbers given at each end node are the respective values of the inputs  $x^{(0)}, y^{(0)}, z^{(0)}$ .

Finally, the inputs to  $MV$  are found by searching MVTree, given as Figure 5. The edges are chosen according to the value of  $s^{(0)}$ , which is recovered by observing whether  $u^{(0)}$  has been complemented or not. The numbers given at each end node are the respective values of the inputs  $x_{MV}^{(0)}, y_{MV}^{(0)}, z_{MV}^{(0)}$ .

### B. Stage 2

We now consider the second stage of the attack, where we assume to have recovered all the inputs to  $f^{(0)}$ , and move on to the inputs to  $f^{(1)}$ . As mentioned earlier, the procedure is basically the same as for the first stage. However, some difficulties are encountered:

- 1) In order to complement the inputs to  $f^{(1)}$ , we need to be fully aware of the first clocking, since we need to know which initial state bits the inputs to  $f^{(1)}$  are dependent on. However, by studying the function  $F$ , we observe that the clocking of the X-LFSRs depends on 3 bits that we have not yet recovered, namely  $r_{5,10}, r_{7,8}$  and  $r_{8,8}$ . This difficulty can be avoided by guessing the values of these 3 bits.
- 2)  $u^{(1)}$  depends on the first clocking, in addition to the inputs to  $f^{(1)}$ , so we need to control a much larger number of bits. Whether this is possible, depends on the nature of the first clocking of the R-LFSRs. This problem can, to some extent, be overcome by introducing a criterion on this clocking.
- 3) Several of the inputs to  $f^{(1)}$  are also involved in the first clocking. Such bits cannot be complemented without potentially affecting the first clocking, which in turn may affect  $u_1$  in an unintended, possibly unpredictable manner. However, if we guess the values of  $r_{5,10}, r_{7,8}$  and  $r_{8,8}$ , we assume to know all the bits involved in the first clocking. We are thus able to calculate the unintended effect on the clocking, and if it changes, we may restore the original clocking, by complementing certain other, independent bits.

We proceed by a detailed description of each of the above problems, with the purpose of justifying the suggested solutions.

Due to the nature of the problems 1 and 3, we choose to guess the values of  $r_{5,10}, r_{7,8}$  and  $r_{8,8}$ . In the following, we thus assume to be fully aware of the nature of the first clocking, hence the first problem is solved.

The second problem addresses the possibility of dependence between the initial state bits that need to be controlled. In order for the procedure of the first stage to be useful also during the second stage, we need to control the following bits:

- the inputs to  $f^{(0)}$  that may affect the clocking, i.e. every input except the inputs to  $MV$ ,
- the inputs to  $F$  not involved in  $f^{(0)}$ , i.e.  $r_{5,10}, r_{7,8}$  and  $r_{8,8}$ ,
- the inputs to  $f^{(1)}$ .

It is clear that how many and which input bits we need to control during the second stage, is completely determined by the clocking. We have run a series of tests in order to decide whether the desired control is attainable for different clockings. In particular, we have observed that if  $R8$  is stepped during the first clocking, then there is some linear dependency in the subset of initial state bits corresponding to the following variables:

- the inputs to  $f^{(1)}$ :  $x_{R8}^{(1)}, y_{R8}^{(1)}$  and  $z_{R8}^{(1)}$ ,
- the inputs to  $f^{(0)}$ :  $x_{R1}^{(0)}, y_{R1}^{(0)}, z_{R1}^{(0)}, \dots, x_{R8}^{(0)}, y_{R8}^{(0)}, z_{R8}^{(0)}, r_{5,10}, r_{7,8}$  and  $r_{8,8}$ .

This may cause problems for several of the procedures involved in the attack, since this dependency prevents us from complementing each of the above bits independently of each other. Due to this, we will prevent that  $R8$  is stepped throughout the attack.

Moreover, tests showed that if we allow any combination of  $R1, R2, R3$  and  $R4$  to be stepped, and let the X-LFSRs be stepped in an arbitrary way, then the desired control can be obtained. In line with this, we have introduced the criterion that neither one of  $R5, R6, R7$  or  $R8$  should be stepped during the first clocking, which is observed to be true with probability  $\frac{1}{2}$ . Referring to Section III, we thus consider the subset  $s_I$  containing all of the bits potentially involved under this criterion, and assume that the vectors  $m_{Ii}$  which complements the respective  $s_i$  when added to  $MI$ , are given by the equation

$$B_I \mathbf{m}_{Ii} = \mathbf{e}_{Ii}. \quad (6)$$

As for the third problem, this needs to be solved separately for each of the relevant bits.

We proceed by giving the overall strategy for the second stage. The starred algorithms are modified versions of the original ones.

- 1) Find  $g^{(1)}$  by the algorithm Find $g^*$ .
- 2) Find the inputs to  $G, b_1^{(1)}, \dots, b_6^{(1)}$  by one of the search trees GTree0\* or GTree1\* together with Find $g^*$ .
- 3) Find the path through the network of R-LFSRs by the algorithm FindPath\*.

- 4) Find  $r_1^{(1)}, \dots, r_8^{(1)}$  by the search tree RTree\*.
- 5) Find the inputs to  $h$  for the respective R-LFSRs,  $x_{R1}^{(1)}, y_{R1}^{(1)}, z_{R1}^{(1)}, \dots, x_{R8}^{(1)}, y_{R8}^{(1)}, z_{R8}^{(1)}$ , by the search tree hTree\* together with RTree\*.
- 6) Find the inputs to  $MV$ , written as  $x_{MV}^{(1)}, y_{MV}^{(1)}$  and  $z_{MV}^{(1)}$ , by the search tree MVTree\*.

In Findg\*, the bits that may need to be complemented are  $z_{R1}^{(1)}, \dots, z_{R8}^{(1)}$ . We note that the  $z_{Ri}^{(1)}$  of the R-LFSRs that have not been stepped, have not been involved in the clocking, so they can be complemented during the second stage without causing problems. Furthermore, we observe that neither of the  $z_{Ri}^{(1)}$  in the case where  $Ri$  has been stepped, has been involved in the clocking. In other words, Findg\* works exactly as Findg, when manipulating  $z_{R1}^{(1)}, \dots, z_{R8}^{(1)}$  in the same systematic manner and observing how  $u^{(1)}$  is affected.

When searching GTree0\* or GTree1\*, it may be necessary to complement bits that are also involved in the first clocking, since one of the X-LFSRs has not been stepped. For instance, if  $X1$  is not stepped, then complementing  $b_1^{(1)}$  or  $b_2^{(1)}$  means complementing  $b_1^{(0)}$  or  $b_2^{(0)}$ , respectively. However, since both  $X2$  and  $X3$  have been stepped, we know that  $b_3^{(0)}, b_4^{(0)}, b_5^{(0)}$  and  $b_6^{(0)}$  may be complemented, without affecting anything other than  $g^{(0)}$ . Hence we may eliminate the effect on  $g^{(0)}$  of complementing  $b_1^{(1)}$  or  $b_2^{(1)}$ , by complementing a suitable subset of  $\{b_3^{(0)}, b_4^{(0)}, b_5^{(0)}, b_6^{(0)}\}$ . We have verified that whenever keeping one of the subsets  $\{b_1, b_2\}$ ,  $\{b_3, b_4\}$  and  $\{b_5, b_6\}$  fixed, we may always complement  $g$  by complementing a suitable subset of the remaining inputs. It is observed that the bit  $x_{1,8}$  does not cause any problems, because there is no need to complement it during the second stage; in order to complement  $b_2^{(1)}$  if  $X1$  is stepped twice, we simply complement one of the other feedback bits.

The algorithm FindPath\* complements the same subset of bits as Findg\*, so the only modification needed is to change  $z_{R1}^{(0)}, \dots, z_{R8}^{(0)}$  into  $z_{R1}^{(1)}, \dots, z_{R8}^{(1)}$  and  $u^{(0)}$  into  $u^{(1)}$ . The same holds for RTree\*. However, when complementing  $g^{(1)}$ , the  $b_i^{(0)}$  should be treated with the same precautions as in Findg\*.

When considering hTree\*, there are a few bits that must be treated carefully, as it may be necessary to complement all of the bits  $x_{R1}^{(1)}, y_{R1}^{(1)}, z_{R1}^{(1)}, \dots, x_{R8}^{(1)}, y_{R8}^{(1)}, z_{R8}^{(1)}$ . We note that again, we only need to consider the bits of the R-LFSRs that have been stepped, since the other ones have not been involved in the clocking. However, there is one exception: The bit  $y_{R6}^{(1)}$  involves the initial state bit  $r_{6,10}$ , which is also involved in the clocking. To deal with this the respective bits may be treated as follows. We have included scenarios where  $R5, R6$  and  $R7$  are stepped, as this will happen during the repetitions of the second stage.

- Assume that  $R2$  has been stepped. If we need to complement  $x_{R2}^{(1)}$ , we simply complement one of the other feedback bits.
- Assume that  $R4$  has been stepped, and that we need to complement  $y_{R4}^{(1)}$ . Then we also complement  $z_{R4}^{(0)}$ , from which we know that  $r_4^{(0)}$  is complemented. By studying

the function  $h$ , we observe that whenever  $z$  is fixed, it is always possible to complement the output by complement  $x$  or  $y$  or both. Since neither of  $x_{R4}^{(1)}$  or  $y_{R4}^{(1)}$  are involved in the clocking, we may eliminate the unintended effect on  $r_4^{(0)}$ , by complementing these bits in a suitable way.

- Assume that  $R6$  has not been stepped. If we complement  $y_{R6}^{(1)}$ , i.e the initial state bit  $r_{6,10}$ , we also complement the input  $c_5^{(0)}$  to the function  $F$ , and the clocking may change. This effect may be eliminated by also complementing the other input to  $c_5^{(0)}$ , the initial state bit  $r_{8,8}$ . If  $R8$  had been stepped, this could be a problem, since  $r_{8,8} = x_{R8}^{(1)}$  in this case. However, since  $R8$  is never stepped, there is no need to consider this scenario.

As for MVTree\*, neither of the inputs  $x_{MV}^{(1)}, y_{MV}^{(1)}$  and  $z_{MV}^{(1)}$  are involved in the clocking, so we only have to change  $x_{MV}^{(0)}, y_{MV}^{(0)}$  and  $z_{MV}^{(0)}$  into  $x_{MV}^{(1)}, y_{MV}^{(1)}$  and  $z_{MV}^{(1)}$ , and  $s^{(0)}$  into  $s^{(1)}$ .

Next we consider the repetitions of the second stage, where the clocking is manipulated intentionally, with the purpose of involving more of the initial state bits in the output function.

We have verified that for each of the following clockings, the desired control of the initial state bits can be obtained:

- 1) Any combination of  $R1, R2, R3$  and  $R4$  are stepped, and  $X1, X2$  and  $X3$  are stepped in an arbitrary way,
- 2)  $R1, R3$ , and  $R5$  are stepped, and  $X1, X2$  and  $X3$  are stepped in an arbitrary way,
- 3)  $R2, R3$ , and  $R5$  are stepped, and  $X1, X2$  and  $X3$  are stepped in an arbitrary way,
- 4)  $R2, R4$  and  $R6$  are stepped, and  $X1, X2$  and  $X3$  are stepped in an arbitrary way.
- 5)  $R1, R4, R6$  and  $R7$  are stepped, and  $X1, X2$  and  $X3$  are stepped in an arbitrary way.

Our knowledge of all the bits determining the clocking and our ability to complement them independently, allows us to manipulate the path through the network of R-LFSRs, in such a way that either one of the cases 1-5 is obtained. We may then repeat the second stage of the attack, and as we know which one of the cases will occur, we may choose in advance the suitable subset  $s_I$  of initial state bits to control. We observe that after two suitable repetitions, each of the LFSRs  $R1, \dots, R7$  has been stepped.

The same strategy can be applied to the clocking of the X-LFSRs. By studying the function  $F$  and the cases 1-5 above, we observe that we may complement all of the inputs  $c_1^{(0)}, \dots, c_5^{(0)}$ , such that all possible clockings of the X-LFSRs are obtainable. As there are 6 different clockings, 5 repetitions are needed, so all in all we need to repeat the second stage 5 – 7 times.

### C. Time complexity of the attack

The time complexity of the first part of the attack, i.e stage 1 and stage 2, including repetitions, will be evaluated in terms of the number of performed bit complementations. To determine this number, we need to investigate the involved algorithms and search trees. We note that when complementing several

bits at a time, this has the same complexity as complementing one bit. We have observed that Findg performs a maximum number of 16 bit complementations and that GTree0 and GTree1 both have depth 16. The actual depth of RTree has not been examined, but the criteria introduced on the partitions imply that it is 17 in the worst case. FindPath has depth 16 and hTree and MVTree both have depth 2. From this we deduce the following worst-case numbers of bit complementations for each of the steps of stage 1:

- 1) 16
- 2)  $5 \cdot 16$
- 3) 16
- 4) 17
- 5)  $8 \cdot 2 \cdot 17$
- 6) 2

Hence, for stage 1, the total number of bit complementations needed is 403 in the worst case.

As for stage 2, we observe that the worst-case number is the same as for stage 1, since the number of bit complementations in each procedure is essentially the same.

In the worst case, stage 2 must be repeated 7 times. Since we are guessing the values of  $r_{5,10}$ ,  $r_{7,8}$  and  $r_{8,8}$ , we must repeat the first part of the attack at most 8 times. We thus conclude that the worst-case number of necessary bit complementations is totally  $403 \cdot 9 \cdot 8 \approx 30000$ .

As mentioned introductorily in this section, the final part of the attack involves exhaustive search of the remaining candidate keys. The time complexity of this process clearly depends on the number of initial state bits recovered at this stage.

We note that some of the obtained bits are actually feedback bits. For simplicity, the feedback bit of an LFSR at the  $t$ th clocking is marked by the subscript  $fbt$ . From the description of the attack, we deduce that the following initial state bits are recovered:

- $r_{1,2}, r_{1,3}, r_{1,5}, r_{1,6}, r_{1,7}, r_{1,8},$
- $r_{2,3}, r_{2,4}, r_{2,8}, r_{2,9}, r_{2,10}, r_{2,fb0},$
- $r_{3,4}, r_{3,5}, r_{3,6}, r_{3,7}, r_{3,9}, r_{3,10},$
- $r_{4,3}, r_{4,4}, r_{4,7}, r_{4,8}, r_{4,9},$
- $r_{5,2}, r_{5,3}, r_{5,4}, r_{5,5}, r_{5,8}, r_{5,9}, r_{5,10},$
- $r_{6,4}, r_{6,5}, r_{6,6}, r_{6,7}, r_{6,10}, r_{6,fb0},$
- $r_{7,2}, r_{7,3}, r_{7,5}, r_{7,6}, r_{7,8}, r_{7,9}, r_{7,10},$
- $r_{8,3}, r_{8,5}, r_{8,7}, r_{8,8},$
- $x_{1,0}, x_{1,1}, x_{1,2}, x_{1,8}, x_{1,9}, x_{1,10}, x_{1,16}, x_{1,fb0}, x_{1,fb1},$
- $x_{2,0}, x_{2,1}, x_{2,2}, x_{2,9}, x_{2,10}, x_{2,11}, x_{2,18}, x_{2,fb0}, x_{2,fb1},$
- $x_{3,0}, x_{3,1}, x_{3,2}, x_{3,11}, x_{3,12}, x_{3,13}, x_{3,22}, x_{3,fb0}, x_{3,fb1}.$

Although the obtained feedback bits can not be directly identified with initial state bits, they are equally valuable to the attacker. As an example, consider the feedback bit  $r_{6,fb0}$ , and assume that it has the value 0. This gives

$$r_{6,0} \oplus r_{6,3} \oplus r_{6,4} \oplus r_{6,10} = 0,$$

which allows us to eliminate one of the unknown bits. Hence the recovery of a feedback bit may be regarded as recovering an initial state bit, as long as the feedback bit depends on at

least one unknown bit. In our case, this is observed to hold for each of the relevant feedback bits.

From the above we conclude that we have recovered 74 initial state bits. Let  $s_I$  denote the subset constituted by these bits. By performing Gaussian elimination on the corresponding matrix  $A_I$ , the secret key is obtained by exhaustive search of  $2^{120 - \text{rank}(A_I)}$  candidates. We have verified that  $A_I$  has full rank, so the number of candidates to be tested is  $2^{46}$ . Comparing this number to the number of bit complementations performed in the first part, it is clear that the exhaustive search dominates the total time complexity of the attack.

It must be taken under consideration that we guess the values of the bits  $r_{5,10}$ ,  $r_{7,8}$  and  $r_{8,8}$ . In most cases, guessing the wrong values may be detected at an early stage of the attack. But, in worst case, the attack must be repeated  $2^3$  times in order to be successful. This gives a time complexity corresponding to exhaustive search of  $2^{49}$  candidates.

For a more detailed description of the attack and all the involved algorithms we refer to our master thesis, which can be obtained upon request to the authors. In our master thesis we also point out some correlations in Whiteout, which could be exploited in a divide and conquer correlation attack [3] on a modified version of Whiteout. It would be interesting to see if these correlations could be used in an optimization of the exhaustive search involved in the attack. In our thesis we have also discussed why the MAC algorithm of Whiteout does not have the computation-resistance property, which is required for a MAC to resist forgery attacks [4].

## V. CONCLUSION

We have studied the stream cipher Whiteout and proposed an attack, that may be carried out by an active adversary, on the cipher which recovers 74 bits of the initial states of all LFSRs involved. In worst case 30000 ciphertexts of length 2 are needed to find these bits. The attack relies on the fact that the initialization procedure of Whiteout is linear. This allows us to construct linear equations relating the initial state with the initialization vector variables and the secret key. Whiteout has a secret key of 120 bits and our attack recovers this key by an exhaustive search of only 49 bits.

## ACKNOWLEDGMENT

We would like to thank Leif Nilsen and Thales Communications AS for allowing us to study Whiteout, we also thank our supervisor Kristian Gjøsteen for his help and advice when working with our master thesis.

## REFERENCES

- [1] J. D. Golić, V. Bagini, and G. Morgari, "Linear Cryptanalysis of Bluetooth Stream Cipher," in *EUROCRYPT 2002*, ser. Lecture Notes in Computer Science, L. R. Knudsen, Ed. Springer, 2002, pp. 238–255.
- [2] S. Fluhrer and S. Lucks, "Analysis of the  $E_0$  Encryption System," in *SAC 2001*, ser. Lecture Notes in Computer Science, S. Vadenay and A. Youssef, Eds., vol. 2259. Springer, 2001, pp. 38–48.
- [3] T. Siegenthaler, "Decrypting a Class of Stream Ciphers Using Ciphertext Only," *IEEE Transactions on Computers*, vol. c-34, january 1985.
- [4] A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 1996.