Cryptanalysis of the Loiss Stream Cipher

Alex Biryukov[1], Aleksandar Kircanski[2] and Amr M. Youssef[2]

[1] University of Luxembourg
Laboratory of Algorithmics, Cryptology and Security (LACS)
Rue Richard Coudenhove-Kalergi 6, Luxembourg, Luxembourg
[2] Concordia University
Concordia Institute for Information Systems Engineering (CIISE)
Montreal, Quebec, H3G 1M8, CANADA.

Abstract. Loiss is a byte-oriented stream cipher designed by Dengguo Feng *et al.* Its design builds upon the design of the SNOW family of ciphers. The algorithm consists of a linear feedback shift register (LFSR) and a non-linear finite state machine (FSM). Loiss utilizes a structure called Byte-Oriented Mixer with Memory (BOMM) in its filter generator, which aims to improve resistance against algebraic attacks, linear distinguishing attacks and fast correlation attacks. In this paper, by exploiting some differential properties of the BOMM structure during the cipher initialization phase, we provide an attack of a practical complexity on Loiss in the related-key model. As confirmed by our experimental results, our attack recovers 92 bits of the 128-bit key in less than one hour on a PC with 3 GHz Intel Pentium 4 processor. The possibility of extending the attack to a resynchronization attack in a single-key model is discussed. We also show that Loiss is not resistant to slide attacks.

1 Introduction

Several word-oriented LFSR-based stream ciphers have been recently proposed and standardized. Examples include ZUC [1], proposed for use in the 4G mobile networks and also SNOW 3G [3], which is deployed in the 3GPP networks. The usual word-oriented LFSR-based design consists of a linear part, which produces sequences with good statistical properties and a finite state machine which provides non-linearity for the state transition function.

In 2011, the Loiss stream cipher [4] was proposed by a team from the State Key Laboratory of Information Security in China. The cipher follows the above mentioned design approach: it includes a byte-oriented LFSR and an FSM. The novelty in the design of Loiss is that its FSM includes a structure called a Byte Oriented-Mixer with Memory (BOMM) which is a 16 byte array adopted from the idea of the RC4 inner state. The BOMM structure is updated in a pseudorandom manner.

The Loiss key scheduling algorithm utilizes a usual approach to provide non-linearity over all the inner state bits. During the initialization phase, the FSM output is connected to the LFSR update function. This ensures that after the initialization process, the LFSR content depends non-linearly on the key and the IV. Such an approach has been previously used in several LFSR-based word-oriented constructions such as the SNOW family of ciphers [3]. In Loiss,

however, the FSM contains the BOMM element which is updated slowly in a pseudo-random manner. The feedback to the LFSR, used in the initialization phase, passes through this BOMM which turns out to be exploitable in a differential-style attack since the BOMM does not properly diffuse differences. In this paper, we provide a related-key attack of a practical complexity against the Loiss stream cipher by exploiting this weakness in its key scheduling algorithm (see also [7] for a work that was done independently of our results). The attack requires two related keys differing in one byte, a computational work of around 2^{26} Loiss initializations, $2^{25.8}$ chosen-IVs for both of the related keys, offline precomputation of around 2^{26} Loiss initializations and a storage space of 2^{32} words. This shows that the additional design complication, i.e., the addition of the BOMM mechanism, weakens the cipher instead of strengthening it. We also discuss the possibility of extending such a related-key attack into a resynchronization single-key attack. Finally, we show that Loiss does not properly resist to slide attacks.

The rest of the paper is organized as follows. In section 2, we briefly review relevant specifications of the Loiss stream cipher. Our related-key attack is detailed in section 3 where we also discuss the possibility of extending the attack to the single-key scenario. In section 4, we show that Loiss is not resistant to slide attacks. Finally, our conclusion is given in section 5.

2 Specifications of Loiss

Figure 1 shows a schematic description of the Loiss stream cipher. In here, we

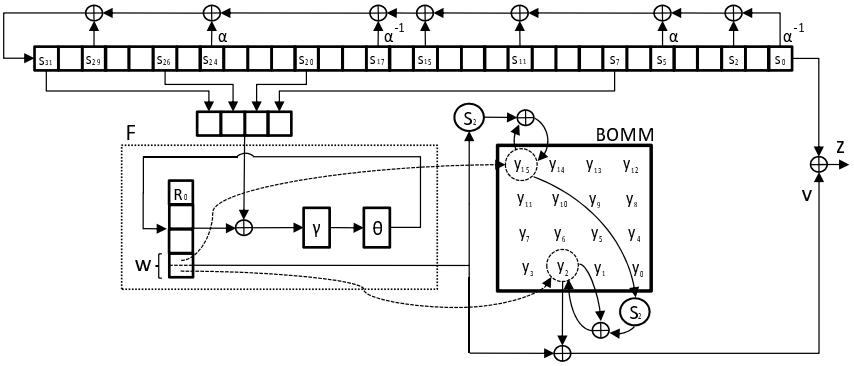

Fig. 1. Loiss stream cipher

briefly review relevant components of the cipher. Let F_{2^8} denote the quotient field $F_2[x]/(\pi(x))$, where the corresponding primitive polynomial $\pi(x) = x^8 + x^7 + x^5 + x^3 + 1$. If α is a root of the polynomial $\pi(x)$ in F_{2^8}, then the LFSR of Loiss is defined over F_{2^8} using the characteristic polynomial

$$f(x) = x^{32} + x^{29} + \alpha x^{24} + \alpha^{-1} x^{17} + x^{15} + x^{11} + \alpha x^5 + x^2 + \alpha^{-1}.$$

The usual bijection between the elements of F_{2^8} and 8-bit binary values is used. The LFSR consists of 32 byte registers denoted by s_i, $0 \leq i \leq 31$. Restating the above equation, if $s_0^t, \ldots, s_{31}^t$ denote the LFSR registers after t LFSR clocks, then the LFSR update function is defined by

$$s_{31}^{t+1} = s_{29}^t \oplus \alpha s_{24}^t \oplus \alpha^{-1} s_{17}^t \oplus s_{15}^t \oplus s_{11}^t \oplus \alpha s_5^t \oplus s_2^t \oplus \alpha^{-1} s_0^t \tag{1}$$

and $s_i^{t+1} = s_{i+1}^t$ for $0 \leq i \leq 30$.

The FSM consists of the function F and the BOMM. The function F compresses 32-bit words into 8-bit values. It utilizes a 32-bit memory unit R and takes LFSR registers s_{31}, s_{26}, s_{20} and s_7 as input. In particular, in each step, the output of F is taken to be the 8 leftmost bits of the register R, after which the R value is updated by

$$\begin{aligned} X &= s_{31}^t | s_{26}^t | s_{20}^t | s_7^t \\ R^{t+1} &= \theta(\gamma(X \oplus R^t)) \end{aligned}$$

where γ is the S-box layer which uses 8×8 S-box S_1 and is defined by

$$\gamma(x_1|x_2|x_3|x_4) = S_1(x_1)|S_1(x_2)|S_1(x_3)|S_1(x_4)$$

and θ is a linear transformation layer defined by

$$\theta(x) = x \oplus (x \lll 2) \oplus (x \lll 10) \oplus (x \lll 18) \oplus (x \lll 24)$$

Since the attack technique provided in this paper does not depend on the particular choice of the used S-boxes, we refer the reader to [4] for the specifications of S_1 and S_2.

As for the BOMM structure, it utilizes 16 memory units, i.e., bytes $y_0, \ldots, y_{15}$. The BOMM function maps 8-bit values to 8-bit values. Let w and v denote the input and output of the BOMM function. Denote the nibbles of its input w as $h = w \gg 4$ and $l = w \bmod 16$. Then, the BOMM function returns $v = y_h^t \oplus w$, after which the update of its memory units takes place as follows:

$$\begin{aligned} &y_l^{t+1} = y_l^t \oplus S_2(w) \\ &\text{If } h \neq l, \text{ then} \\ &\qquad y_h^{t+1} = y_h^t \oplus S_2(y_l^{t+1}) \\ &\text{else} \\ &\qquad y_h^{t+1} = y_l^{t+1} \oplus S_2(y_l^{t+1}) \\ &y_i^{t+1} = y_i^t, \text{ for } 0 \leq i \leq 15 \text{ and } i \notin \{h, l\} \end{aligned}$$

where S_2 is an 8×8 S-box. In the FSM update step, the input to the BOMM function, i.e., the w value, is taken to be leftmost byte of the output of the F function.

The initialization procedure of Loiss proceeds as follows. The register R is set to zero, i.e., $R^0 = 0$. If the key K and the initialization vector IV are represented byte-wise as

$$\begin{aligned} K &= K_{15}|K_{14}|\cdots|K_0 \\ IV &= IV_{15}|IV_{14}|\cdots|IV_0, \end{aligned} \tag{2}$$

then the starting inner state $(s_{31}^0, \ldots, s_0^0, R^0, y_{15}^0, \ldots, y_0^0)$ is loaded with the K and IV as follows:

$$s_i^0 = K_i, \quad s_{i+16}^0 = K_i \oplus IV_i, \quad y_i^0 = IV_i \tag{3}$$

for $0 \leq i \leq 15$. Then, Loiss runs for 64 steps and the output of the BOMM takes part in the LFSR update step. In other words, instead of (1), the following LFSR update function is used:

$$s_{31}^{t+1} = s_{29}^t \oplus \alpha s_{24}^t \oplus \alpha^{-1} s_{17}^t \oplus s_{15}^t \oplus s_{11}^t \oplus \alpha s_5^t \oplus s_2^t \oplus \alpha^{-1} s_0^t \oplus \boldsymbol{v^t} \quad (4)$$

Then, the keystream generation stage starts. Loiss generator produces one byte of keystream per step:

$$z^t = s_0^t \oplus v^t.$$

In general, except for the new BOMM component, the whole Loiss design is very similar to the design of the SNOW 3G cipher. It is also interesting to note that the same θ linear layer has been used in the SMS4 block cipher [2] and also in ZUC [1].

3 Proposed Attack

In this section, a differential-style attack against the Loiss key scheduling algorithm is presented. The attack requires two related keys that differ in one byte. It also requires the ability to resynchronize the cipher under the two keys with chosen IV values.

The attack starts by having the pair of inner states right after the key loading step differ only in one LFSR byte and one BOMM byte. Then, the idea is to have the LFSR difference fully cancelled. We use the fact that the BOMM output participates in the LFSR update step during the initialization and the BOMM difference helps us to cancel out the LFSR difference through the feedback. Once the difference in the LFSR is fully cancelled, only the BOMM component is active and moreover, with a single byte difference. Then, since the BOMM does not have proper diffusion properties, the single-byte difference stays localized in the BOMM until the end of the initialization, which can be detected from the keystream.

The probability of the event that a given BOMM byte is not used during the initialization is $(\frac{15}{16})^{128} \approx 2^{-12}$, since a BOMM element is consulted 128 times during the 64 initialization steps. If the active byte has not been used until the end of the initialization, the two instances of the cipher generate several equal keystream bytes with high probability. Namely, the difference at the point where the keystream is to be produced will be of low-weight and localized in the BOMM. Therefore, spotting large number of zero bytes in the starting keystream byte difference indicates that the LFSR difference cancellations described above took place. These cancellations happen only when certain equations in the starting LFSR bytes are satisfied and consequently, since the starting LFSR bits are related to the key bits, information about the key bits leaks.

Let K and K' differ only in the byte K_3. The steps of the attack can be summarized as follows:

- Construct a list of (IV, IV') pairs for which the LFSR state difference cancellation happens. The cancellation event is described in section 3.1, the distinguisher used to detect this event is given in section 3.2 and a procedure for collecting the (IV, IV') pairs is provided in section 3.3.
- Use this collection of IVs as input to the filtering procedure to filter the wrong key candidates, as described in section 3.4.

The attack recovers 92 bits of the key and the remaining $128 - 92 = 36$ bits can be obtained by brute force. In another variant of the attack, 112 bits of the key are recovered and the rest are found by brute-force.

3.1 Cancelling the LFSR difference

In this section, a necessary and sufficient condition for the starting inner state difference to be fully cancelled in the LFSR after 4 steps is provided. The condition is specified in terms of the leftmost byte of the R register in the first 4 steps. Then, the conditions on the R register as provided by Observation 1 below leak information on the early LFSR bytes and thus about the secret key.

The key-loading mechanism (3) allows having a chosen difference only at bytes s_3 and y_3 at time $t = 0$. Namely, it suffices to have

$$K_3 \oplus K'_3 = IV_3 \oplus IV'_3 = \delta \tag{5}$$

and the rest of the K, K' and also IV, IV' bytes to have a zero-difference. Moreover, the key-loading mechanism trivially allows choosing the starting values of the y_3 register. This is done by choosing the IV_3 byte, since the IV is simply copied into the BOMM. This shows that the assumptions required by Observation 1 (i.e., the particular difference value $0x02$ in s_3 and y_3 and also the $y_3 = 0x9d$ constant) can be satisfied. Recall that w^t denotes the leftmost byte of the R register at time $t \geq 0$.

Observation 1 *Let a pair of Loiss inner states have only s_3 and y_3 bytes active, both with difference $0x02$. Also, let $y_3 = 0x9d$. Then, after 4 steps, the LFSR does not contain any active byte if and only if*

$$(w^0, w^1, w^2, w^3) = (0x00, 0x33, 0xK?, 0x3?) \tag{6}$$

where K is any hexadecimal digit different from $0x3$ and the symbol "?" denotes any hexadecimal digit.

The proof of the observation above is given in Appendix B. Here, a descriptive overview of the cancellation specified by Observation 1 is provided. In Figure 2, the BOMM and the LFSR bytes s_3, s_2, s_1, s_0 are shown during the first four steps. In the second and the fourth states in the figure, the cancellation of the LFSR difference by the feedback byte to the LFSR update is denoted. In the first step, the difference does not enter neither the LFSR update function nor the feedback value (since $w^0 = 0x00$). In the second step, it is required that $w^1 = 0x33$ for the difference to be cancelled and also to be updated to the next necessary BOMM difference value, $2\alpha^{-1}$. In the third step, the difference is neither passed to the LFSR nor changed in the BOMM. Finally, in the fourth step, the difference in the LFSR byte s_0 is cancelled and the LFSR becomes fully inactive.

It should be noted that Observation 1 holds for other difference values apart from $\delta = 0x02$. The set Δ of such differences is given in Appendix A. In particular, Observation 1 is true for any $\delta \in F_2^8$ such that the input differences

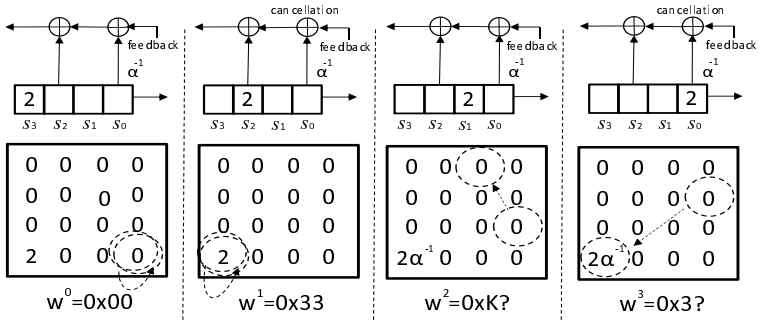

Fig. 2. Illustration of the differences in the BOMM structure at times $t = 0, 1, 2, 3$

$\alpha^{-1} \times \delta$ and δ cannot be mapped to the output differences $\alpha^{-1} \times \delta$ by the S_2 S-box (see the ($\Rightarrow$) part of the proof in Appendix B). For each difference from the set Δ, the initial constant for y_0^3 is calculated from (14).

The overall probability that there will be only one BOMM byte, y_3, active after all of the 64 steps of the key scheduling procedure is estimated next. For this event to happen, it suffices to have (6) satisfied in addition to ensuring that the y_3 difference does not propagate to other bytes during the initialization procedure. The event (6) happens with probability $p_w = 2^{-8} \times \frac{15}{16} \times 2^{-4} \approx 2^{-12.1}$. The event by which the y_3 difference does not propagate to any other byte is equivalent to the event of $w^t \bmod 16 \neq 0x3$ and $w^t \gg 4 \neq 0x3$ for $4 \leq t \leq 63$, and $w^2 \bmod 16 \neq 3$. The latter condition is included since Observation 1 does not rule out the possibility of the spreading of the y_3 difference to another byte during step 3. Thus, the probability that y_3 does not spread to any other byte is $p_s = (\frac{15}{16})^{2\times 60+1} \approx 2^{-11.3}$. Thus, a randomly chosen key-IV pair satisfying (5) such that the assumptions of Observation 1 are satisfied produces a pair of inner states with only one active byte with probability

$$p = p_s \times p_w = 2^{-12.1} \times 2^{-11.3} = 2^{-23.4} \tag{7}$$

under the usual independence assumption.

3.2 Distinguishing Loiss pairs

In the previous subsection, we showed that it is possible to have a pair of Loiss inner states with only one active byte (located in the BOMM) after the initialization. Here, a distinguisher for the keystreams generated by a pair of such states is provided. The goal is to minimize the probability of false positives and false negatives.

Let the time at which the two instances of the cipher differ by only one BOMM byte be $t = 0$. Since at this time most of the words are inactive, it is natural to attempt distinguishing Loiss key stream pairs from random keystream pairs by simply counting the number of equal bytes in the two outputs. Such a distinguisher depends on parameters n and m, where n is the number of

keystream generation steps that will be considered and m is the number of equal corresponding words in steps $0, \ldots, n-1$. The distinguisher can be formulated as:

- Count the number of indices $0 \leq i < n$ such that $z_i = z'_i$
- If this count is $\geq m$ return *Loiss keystreams*, otherwise return *Random*.

Good values for (n, m) can be chosen by consulting Table 1 Appendix C. In this table, the probability of false positives and false negatives for some representative (n, m) points has been tabulated. Details on how the values in the table have been calculated are provided below.

The false positive probability signifies the probability that in two random sequences of n bytes, more than m corresponding bytes will be equal. On the other hand, the false negative probability signifies the probability that two Loiss instances with only one active byte located in the BOMM, will produce strictly less than m equal bytes. For the purpose of the attack above, it is necessary to keep the probability of false positives low, since a false positive would lead to generating equations that have incorrect key values as solutions. As for the false positive probability, it has been calculated by using the formula describing the probability that in n randomly generated bytes, at least m of them are equal to zero. Namely, if l denotes the number of zeros in the sample, then $P[\text{false positive}] = P[l \geq m] = \sum_{l=m,\ldots,n} \binom{n}{l} \left(\frac{1}{256}\right)^l \left(\frac{255}{256}\right)^{n-l}$.

The false negative probability has been calculated experimentally by randomly generating a pair of equal Loiss inner states and then inducing a random difference at a random BOMM byte. After running the cipher for n steps, the number of equal bytes is counted. If such number is strictly smaller than m, a counter is incremented. After repeating the previous procedure for 2^{28} times and dividing the resulting counter by 2^{28}, an approximation of the probability of a false negatives is obtained.

For the purpose of the distinguisher used in the next subsection, taking $(n, m) = (32, 10)$ makes the probability of the attack failure marginally small, i.e., equal to around $2^{25.8} \times 2^{-54.2}$, since the distinguisher is applied for around $2^{25.8}$ times and a false positive answer would lead to wrong conclusions about the value of key bytes.

3.3 Finding the correct IVs

According to the cancellation probability (7), for around one in $2^{23.4}$ randomly chosen IVs, if the key-IV pair satisfies (5), the inner state right after the initialization will have only the y_3 BOMM byte active. Given the choices for the distinguisher given in Table 1, such event can be reliably detected. Hereafter, such IVs will be called *correct* IVs. In this section, it is shown that the correct IVs can be found with probability better than $2^{-23.4}$, which helps us reduce the final number of chosen-IVs required for the attack.

In particular, once one correct IV is obtained, more such correct IVs can be found with better probability. Namely, changing certain IV bytes in a correct IV does not influence all w_1, w_2 and w_3 bytes. For instance, perturbing byte IV_{11} in a correct IV does not change $w^1 = 0x33$ value and the the probability (7) that the new IV will also be a correct one increases by a factor of 2^8. More precisely, let T_1 denote a collection of IV bytes such that any change in bytes from T_1 leaves R^1 unchanged, but changes R^t, $t \geq 2$. It is easy to verify that $T_1 = \{IV_1, IV_5, IV_8, IV_{11}, IV_{13}\}$.

Thus, after finding one correct IV, varying only the bytes from T_1 can serve to find more correct IVs with better probability. Such a set of IVs would result in the IVs for which the R^1 word is constant. However, the attack step provided in subsection 3.4, which takes the correct IV set as its input, requires that the IVs produce about 5 different R^1 values. Similarly, there have to be around 360 different R^2 values. These two numbers of required different R^1 and R^2 values are necessary to minimize the number of key byte candidates that will be recovered, as will be explained in the next subsection. Therefore, the search procedure that produces the input to the procedure in the next subsection can proceed as follows:

- Let sets $L_0 = L_1 = L_2 = L_3 = L_4 = \emptyset$.
- Generate 5 correct IVs randomly and place them in sets L_i, $0 \leq i \leq 4$, respectively. In more detail, for each randomly generated IV, compute IV' according to (5) and apply the distinguisher from subsection 3.2. If the distinguisher returns a positive answer, a correct IV has been found.
- For $0 \leq i \leq 4$
  - Using the IV from each L_i, generate more corrects IVs such that the L_i sets contain 72 IVs each. In particular, the new correct IVs are generated by randomizing the starting IV bytes specified by T_1 and applying the distinguisher.

The output of the above procedure are sets L_i, $0 \leq i \leq 3$, each containing 72 IVs for which the R^1 is constant. This procedure takes around $5 \times 2^{23.4} + 5 \times 72 \times 2^{23.4-8} \approx 2^{26}$ chosen-IV queries on both Loiss instances. If instead of applying the previous procedure, all of the $5 \times 72 = 360$ correct IVs were generated randomly, the number of chosen IV queries would be $360 \times 2^{23.4} \approx 2^{31.9}$.

3.4 Filtering the key bytes

In each Loiss step, the function F updates the register R by a transformation similar to one round of a block cipher, where the R value plays the role of the plaintext and the four LFSR registers play the role of the round key. The goal hereafter is to recover the LFSR registers fed to F in the first three initialization steps, i.e., s_{7+i}, s_{20+i}, s_{26+i}, s_{31+i} for $0 \leq i \leq 2$. In particular, since the LFSR bytes in question can be represented as a sum of the key and the IV, the goal is to recover the key part in these bytes. First, the application of the F function in the first three steps is represented in the form of

$$R^{i+1} = F(R^i, k_3^i \oplus iv_3^i | k_2^i \oplus iv_2^i | k_1^i \oplus iv_1^i | k_0^i) \tag{8}$$

for $0 \leq i \leq 2$, where k_3^i, k_2^i, k_1^i and k_0^i depend only on the original key bytes and iv_3^i, iv_2^i and iv_1^i depend only on the IV bytes. More precisely, in the first step

$$\begin{aligned} k_3^0 = K_{15}, k_2^0 = K_{10}, k_1^0 = K_4, k_0^0 = K_7 \\ iv_3^0 = IV_{15}, iv_2^0 = IV_{10}, iv_1^0 = IV_4 \end{aligned} \tag{9}$$

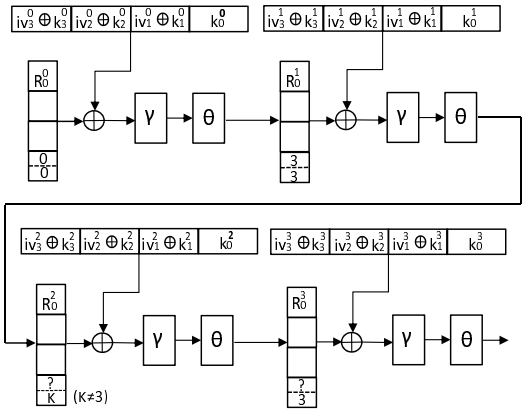

Fig. 3. The R register in times $0 \leq t \leq 3$

In the second step, we have

$$\begin{aligned} &k_3^1 = K_{13} \oplus \alpha K_8 \oplus \alpha^{-1} K_1 \oplus K_{15} \oplus K_{11} \oplus \alpha K_5 \oplus K_2 \oplus \alpha^{-1} K_0 \\ &k_2^1 = K_{11}, k_1^1 = K_5, k_0^1 = K_8 \\ &iv_3^1 = IV_{13} \oplus \alpha IV_8 \oplus \alpha^{-1} IV_1 \oplus IV_{15} \oplus IV_{11} \oplus \alpha IV_5 \oplus \\ &\qquad\qquad IV_2 \oplus \alpha^{-1} IV_0 \oplus f^1 \\ &iv_2^1 = IV_{11}, iv_1^1 = IV_5 \end{aligned} \tag{10}$$

and in the third step

$$\begin{aligned} &k_3^2 = K_{14} \oplus \alpha K_9 \oplus \alpha^{-1} K_2 \oplus K_0 \oplus K_{12} \oplus \alpha K_6 \oplus K_3 \oplus \alpha^{-1} K_1 \\ &k_2^2 = K_{12}, k_1^2 = K_6, k_0^2 = K_9 \\ &iv_3^2 = IV_{14} \oplus \alpha IV_9 \oplus \alpha^{-1} IV_2 \oplus IV_0 \oplus IV_{12} \oplus \alpha IV_6 \oplus IV_3 \oplus \\ &\qquad\qquad \alpha^{-1} IV_1 \oplus f^2 \\ &iv_2^2 = IV_{12}, iv_1^2 = IV_6 \end{aligned} \tag{11}$$

where f^1, f^2 represent the feedback bytes. If the IV bytes in the right-hand side of (9), (10) and (11) are taken from a correct IV, then (6) will hold. In that case, also, the feedback bytes will be $f^1 = IV_0$ and $f^2 = IV_3 \oplus 0x33$. The first three steps of the F function when a correct IV is used are represented schematically in Figure 3.

Then, the filtering procedure for recovering k_j^i, $0 \leq i \leq 2$, $0 \leq j \leq 3$ amounts to substituting the F function key guesses into (8) along with the iv bytes derived from a correct IV and then verifying whether (6) holds. In particular, the filtering procedure is done round by round. As for the first F round, (6) amounts to $R^1 \gg 24 = 0x33$ and thus a candidate for $k^0 = k_3^0|k_2^0|k_1^0|k_0^0$ passes the criterion with probability 2^{-8}, which implies that 5 correct IVs are sufficient to uniquely determine k^0 with a good probability. We have verified experimentally that there is enough diffusion in one F-round to find the key uniquely with just 5 correct IVs.

As for the second step of the initialization phase, where (8) is executed for $i = 1$, first it should be noted that R^1 is known for each IV since $k_3^0|k_2^0|k_1^0|k_0^0$ is known. According to (6), the second F round criterion amounts to $R^2 \gg 28 \neq 3$. Thus, a guess for $k^1 = k_3^1|k_2^1|k_1^1|k_0^1$ passes the criterion with probability $\frac{15}{16}$. Assuming that all the wrong key bits can be eliminated, around 332 correct IV values will be required, since $2^{32} \times (\frac{15}{16})^{332} \approx 1$. In the previous section, 360 correct IVs has been generated, which ensures the unique recovery of k^1 with good probability. Throughout all our experiments, the number of candidates for k^1 that pass the test was consistently equal to 16. Without going into why 16 candidates always pass the test, it is noted that these candidates can be eliminated during the third F round filtering. The third F round criterion is $R^3 \gg 28 = 3$ and one can expect that the candidate for $k^2 = k_3^2|k_2^2|k_1^2|k_0^2$ passes with probability 2^{-4}, meaning that around 8 correct IV values will be required. The filtering is done for each of the 16 candidates for k^1. Again, experimentally, it was found that 16 candidates for k^2 always pass the test and therefore there will be 16 candidates at the end of the filtering procedure. It remains to state how the correct IVs are drawn from L_i, $0 \leq i \leq 4$ to derive the iv^i values specified by (9), (10) and (11). For the first F round filtering, the 5 IVs are chosen from L_0, L_1, L_2, L_3 and L_4, respectively, which ensures that different 5 iv^0 values will be derived and that the filtering procedure will properly work. The second and third round choice of the IVs is arbitrary.

Attack complexity: After the filtering procedure described above, there will remain 16 candidates for k_j^i, $0 \leq i \leq 2$, $0 \leq j \leq 3$ (96 bits). Each of the 16 candidates yields a linear system in the cipher key bytes determined by (9), (10) and (11). Since the linear equations in the system are independent, it follows that a $96 - 4 = 92$-bit constraint on the key K is specified. At this point, the attacker can either brute-force the remaining $128 - 92 = 36$ key bits or continue with the filtering process described above to deduce more key bits. In case of brute-forcing the 36 bits, the total complexity of the attack is dominated by around 2^{36} Loiss initialization procedures.

In the case where the filtering process is continued, the criterion $R^4 \gg 28 \neq 3$ can be used to filter out more key bits. Namely, expanding the corresponding iv^3 and k^3 values in a way analogous to (9)-(11), while taking into account the feedback byte in the LFSR update, reveals that altogether 20 more key bits can be recovered. In that case, the total complexity is dominated by the complexity of the above filtering procedures. The most expensive step is the filtering based on the second F round. We recall that in this filtering step, for each of the 360 correct IVs, each 32-bit key value is tested and eliminated if $R^2 \gg 28 \neq 3$ does not hold. Instead of applying the F function $2^{32} \times 360 \approx 2^{40.5}$ times, one can go through all key candidates for a particular IV, eliminate around $\frac{15}{16}$ of them and then, for the next IV, only go through the remaining candidates. In such a case, the number of applications of F is $\sum_{i=0}^{360} (\frac{15}{16})^i 2^{32} \approx 2^{36}$. To have further optimization, a table containing 2^{32} entries and representing F function can be prepared in advance. To measure the computational complexity of the attack in terms of Loiss initializations, a conservative estimate that one table lookup costs around 2^{-4} of a reasonable implementation of one Loiss initialization step could be accepted. Then, since there are $64 = 2^6$ steps in the initialization, the final complexity amounts to around 2^{26} Loiss initializations, $2^{25.8}$ chosen-IVs for both keys, storage space of 2^{32} 32-bit words and offline precomputation of 2^{32} applications of F, which

is less than 2^{26} Loiss initializations, since each Loiss initialization includes 2^6 F computations.
Our attack was implemented and tested on a PC with 3 GHz Intel Pentium 4 processor with one core. Our implementation takes less than one hour to recover 92 bits of the key information and the attack procedure was successful on all the tested 32 randomly generated keys.

3.5 Towards a resynchronization attack

Here, some preliminary observations on the possibility of adapting the above attack to the single-key model are provided. In the single-key resynchronization attack, only the IV can have active bytes, which means that only the left-hand half of the LFSR, i.e., registers $s_{16}, \ldots, s_{31}$ as well as the BOMM will contain active bytes. As in the related-key attack above, the strategy is to have the difference cancelled out in the LFSR and localized only in the BOMM early during the initialization. One of the obstacles is that the R register will necessarily be activated when the difference reaches byte s_7, since the left-hand half of the LFSR contains active bytes. We note that this obstacle can be bypassed by cancelling the introduced R difference by having more than one LFSR byte active. Let LFSR bytes s_9, s_8 and s_7 be active with differences δ_2, δ_1, δ_0 at some time t during the initialization procedure. Also, assume that the word R and the BOMM bytes to be used in the next three steps are inactive. Below, we determine how many of the $(\delta_2, \delta_1, \delta_0)$ values can leave R inactive after 3 steps (after having passed through s_7) and also the probability of occurrence of such an event. For this purpose, note that the R cancellation event occurs if

$$\gamma(F(x^t) \oplus u^{t+1}) \oplus \gamma(F(x^t \oplus \delta_0) \oplus u^{t+1} \oplus \delta_1) = \theta^{-1}\delta_2 \tag{12}$$

where $x^t = R^t \oplus u^t$ and u^t denotes the 32-bit words fed to the F function from the LFSR in t-th step. By using a precomputed table for the S-box S_1 that, for each input and output difference, contains the information whether it is possible to achieve the input-output difference pair or not, we exhaustively checked for which values of $(\delta_2, \delta_1, \delta_0)$ equation (12) has solutions in x^t and u^{t+1}. The result of the finding is that only $2^{-12.861}$ of $(\delta_2, \delta_1, \delta_0)$ values cannot yield an R difference cancellation event. For the remaining $(\delta_2, \delta_1, \delta_0)$, for which (12) does have a solution, the probability of the R difference cancellation is $2^{-4} \times 2^{-28} = 2^{-32}$.
The analysis above indicates that attackers can choose almost any $(\delta_2, \delta_1, \delta_0)$ starting difference at three consecutive LFSR bytes and then bypass an R activation with a probability of 2^{-32}. A possible favorable position to introduce such $(\delta_2, \delta_1, \delta_0)$ difference can be in registers s_{18}, s_{17}, s_{16}, since the R register will only be activated through byte s_7. This can be done by activating IV_2, IV_1, IV_0 bytes. The 3-byte difference that arises in the BOMM then needs to be used for cancellations whenever some of the active LFSR bytes pass through the taps. Due to the relatively high number of cancellations that need to happen as the difference moves towards the right, we have not been able to bring the cancellation probability sufficiently high enough to have a practical attack. Controlling the difference propagation as done in [6] may be useful for that purpose. It is left for future research to verify whether a practical resynchronization single-key attack can be mounted against Loiss.

4 Sliding properties of Loiss

In [5], a slide attack on SNOW 3G and SNOW 2.0 was provided. This attack is a related-key attack and involves a key-IV pair (K, IV) and (K', IV'). The idea is to have the inner state of the (K, IV) instance after $n \geq 1$ steps be a *starting* inner state. Then, the corresponding (K', IV') initializes to this starting state and the equality of the inner states is preserved until the end of the procedure. The similarity between the two keystreams is detected and this provides a basis for the key-recovery attack. Since LFSR-based word-oriented stream ciphers usually do not use counters which are the usual countermeasure against this kind of slide attacks, one way to protect against sliding is to have the initial inner state populated by the key, IV and constants so that it disallows the next several states to be starting states. For example, in ZUC [1], constants are loaded in a way that makes it difficult to mount a slide attack. In the following, we point out that Loiss, similar to SNOW 2.0 and SNOW 3G, does not properly defend against sliding. If $C_0 = S_1^{-1}(0)$ and $C_1 = S_2(0)$, a slide by one step can be achieved as follows.

Observation 2 *Let* $K = (K_{15}, \ldots, K_0)$ *and* $IV = (A, \ldots, A, B)$*, where*

$$A = (\alpha \oplus \alpha^{-1} \oplus 1)^{-1}(K_0 \oplus \alpha^{-1} K_0 \oplus \alpha^{-1} K_1 \oplus K_2 \oplus \alpha K_5 \oplus \alpha K_8 \oplus K_{11} \oplus K_{13} \oplus C_0)$$

and B *is determined by* $B \oplus C_1 \oplus S_2(B \oplus C_1) = A$*. Also, assume that* $K_7 = C_0$ *and* $K_4 = K_{10} = K_{15} = C_0 \oplus A$*. Then, for* $K' = (K_0 \oplus B, K_{15}, \ldots, K_1)$ *and* $IV' = (A, \ldots, A)$*, we have*

$$z'_0 = z_1 \tag{13}$$

The proof of the observation is given in Appendix B.
Due to the requirement on bytes K_7, K_4, K_{10} and K_{15} from the formulation of the observation above, a Loiss key K has a related key pair specified by the observation above with probability 2^{-32}. For the related keys K and K' satisfying the conditions above, the attack can be performed by going through all $A \in F_2^8$ and verifying whether the relation (13) is satisfied for $IV = (A, \ldots, A, B)$, and $IV' = (A, \ldots, A)$. If yes, then such an A byte is a candidate for the right-hand side of the equation above specifying A, which depends only on K bytes. Each false candidate out of 2^8 candidates for A will pass the test (13) with probability 2^{-8}. That way, around one byte of the key information leaks. Slides by more than one step may also be possible.

5 Conclusion

We presented a practical-complexity related-key attack on the Loiss stream cipher. The fact that a slowly changing array (the BOMM) has been added as a part of the FSM in Loiss allowed the difference to be contained (i.e., do not propagate) during a large number of inner state update steps with a relatively high probability. The attack was implemented and our implementation takes less than one hour on a PC with 3GHz Intel Pentium 4 processor to recover 92 bits of the 128-bit key. The possibility of extending the attack to a resynchronization attack in a single-key model was discussed. We also showed that a slide attack is possible for the Loiss stream cipher.

References

1. Specification of the 3GPP Confidentiality and Integrity Algorithms 128-EEA3 and 128-EIA3. Document 2: ZUC Specification, 2010. http://www.dacas.cn.
2. Specification of SMS4, Block Cipher for WLAN Products - SMS4 (in Chinese), Declassified in: September 2006. http://www.oscca.gov.cn/UpFile/200621016423197990.pdf.
3. ETSI/SAGE. Specification of the 3GPP Confidentiality and Integrity Algorithms UEA2&UIA2 Document 2: SNOW 3G Specification (version 1.1), September 2006. http://www.3gpp.org/ftp.
4. Feng, D., Feng, X., Zhang, W., Fan, X., and Wu, C. Loiss: A byte-oriented stream cipher. In *IWCC* (2011), Y. M. Chee, Z. Guo, S. Ling, F. Shao, Y. Tang, H. Wang, and C. Xing, Eds., vol. 6639 of *Lecture Notes in Computer Science*, Springer, pp. 109–125.
5. Kircanski, A., and Youssef, A. On the Sliding Property of SNOW 3G and SNOW 3.0. IET Information Security, Vol. 4, Issue 5, pg. 199-206, December 2011.
6. Knellwolf, S., Meier, W., and Naya-Plasencia, M. Conditional differential cryptanalysis of TRIVIUM and KATAN. In *Selected Areas in Cryptography* (2011), A. Miri and S. Vaudenay, Eds., vol. 7118 of *Lecture Notes in Computer Science*, Springer, pp. 200–212.
7. Lin, D., and Jie, G. Cryptanalysis of Loiss Stream Cipher. *To appear in: The Computer Journal* (2012). Oxford University Press. Available online: http://comjnl.oxfordjournals.org/content/early/2012/05/21/comjnl.bxs047.short?rss=1.

A The set of possible differences

Observation 1 is true for the following values (shown in hexadecimal):

$$\Delta = \{2, 5, 7, 9, d, 10, 11, 13, 15, 16, 18, 19, 1a, 1c, 1d, 1f, 20, 21, 25, 27, 2a, 2b, 2c, 2e, 2f, 31, 32, 37, 38, 39, 3d, 3e, 45, 48, 4a, 4b, 4d, 4f, 50, 54, 56, 57, 5b, 5c, 5d, 60, 61, 63, 64, 65, 66, 69, 6a, 6b, 6c, 6f, 70, 72, 74, 75, 77, 79, 7a, 7b, 7d, 7f, 80, 81, 82, 87, 89, 8b, 8d, 8e, 92, 94, 96, 97, 98, 99, 9a, 9c, 9d, 9e, a0, a1, a9, aa, ac, ae, af, b0, b2, b5, b8, ba, bc, bd, bf, c0, c1, c3, c4, c5, c7, ca, cd, d1, d2, d3, d4, d6, d7, d8, da, dc, de, df, e1, e2, e8, eb, ed, f0, f1, f2, f3, f4, f7, f9, fb, fc, ff\}$$

B Proof of Observations 1 and 2

In this appendix, we provide proofs for the two observations listed in the paper.

Proof of Observation 1:

From the cipher specification, $w^0 = 0x00$ is true regardless of the condition on the left-hand side. The two directions of the proof are provided as follows. ($\Leftarrow$): The change of the difference in the BOMM is described in Figure 2. In the first step, since $w^0 = 0x00$, both the value and the difference of y_3^0 remain unchanged and the LFSR difference is moved from s_3 to s_2. Since $w^1 = 0x33$

and both s_2 and y_3^1 are active with the same difference, they cancel out and the corresponding LFSR byte becomes inactive. As for the LFSR difference, it is just moved to s_1. Another effect of the second step is the change of the difference in y_3 byte from $0x02$ to $\alpha^{-1} \times 2$. Namely, expanding the difference in the y_3 byte and substituting the initial choice of $y_3^0 = 0x9d$ and also the choice of the starting difference $\delta = 0x02$ gives

$$y_3^2 \oplus y_3^{'2} = \delta \oplus S_2(y_3^0 \oplus S_2(0x33)) \oplus S_2(y_3^0 \oplus \delta \oplus S_2(0x33)) = \alpha^{-1} \times 0x02 \quad (14)$$

The third step moves the s_1 active byte to s_0, since $w^2 \gg 4 \neq 3$ and leaves the y_3 difference unchanged. Finally, since $w^3 \gg 4 = 0x3$, the difference in y_3 cancels out the difference in the LFSR update function (4) in the fourth step and this direction of the proof follows.

($\Rightarrow$): Clearly, $w^1 \gg 4 = 0x3$ since otherwise s_{31}^1 would be active and the LFSR after 4 steps would necessarily have at least one active byte. Moreover, $K = w^2 \gg 4 \neq 0x3$ holds since y_3^2 is necessarily active and otherwise there would be a difference introduced to the LFSR on byte s_{31}^2.

To show that $w^1 \bmod 4 = 0x3$, assume the contrary. In that case, the full LFSR cancellation in the fourth step cannot happen. Namely, in the second step, the difference in register y_3^1 remains unchanged, i.e., it remains equal to $0x02$. Therefore, during the third step, the existing one byte difference in the BOMM has to evolve to $\alpha^{-1} \times 2$ in order for the LFSR cancellation to happen in the fourth step. However, according to the S_2 specification, the input S_2 difference $0x02$ cannot be transformed to the output difference $\alpha^{-1} \times 2$ and thus $w^1 \bmod 4 = 0x3$.

Now, according to the ($\Leftarrow$) direction of the proof, (14) holds. To show that $w^3 \gg 4 = 0x3$, suppose the contrary. Since the LFSR byte s_0 is active at the fourth step (with the difference $0x2$), for this difference to be cancelled out, the BOMM output byte at step four has to be active with the same difference. Thus, the difference in y_3^2 which is equal to $\alpha^{-1} \times 0x02$ has to remain $\alpha^{-1} \times 0x02$ after passing through the S_2 S-box. This difference will necessarily be induced on some other BOMM byte since $K \neq 3$. However, such a possibility is ruled out by the S_2 specification: the S_2 S-box cannot map the input difference $\alpha^{-1} \times 2$ to $\alpha^{-1} \times 2$ output difference. It should be noted that this was possible in (14), since the same byte was updated twice in step 1. Therefore, $w^3 \gg 4 = 0x3$ has to hold. □

Proof of Observation 2:

We will show that

$$\begin{aligned} IS^1 &= (s_{31}^1, \ldots, s_0^1, R^1, y_{16}^1, \ldots, y_0^1) \\ &= (s_{31}^{'0}, \ldots, s_0^{'0}, R^{'0}, y_{16}^{'0}, \ldots, y_0^{'0}) = IS^{'0} \end{aligned} \quad (15)$$

As for the BOMM bytes y_i, $15 \leq i \leq 0$, in the (K, IV) instance of the cipher, only y_0 will be updated since $R^0 = 0$. In other words, $y_i^1 = A$ for $15 \leq i \leq 1$. Moreover, from the specification of B, it follows that $y_0^1 = A$. Since $IV' = (A, \ldots, A)$, $y_i^{'0} = A$ for $15 \leq i \leq 0$ as well, i.e., (15) holds for the BOMM bytes. As for the equality between R^1 and $R^{'0}$, by the initialization procedure, $R^{'0} = 0$. To have $R^1 = 0$ as well, it suffices to have each of the four LFSR registers $s_{31}^0, s_{26}^0, s_{20}^0, s_7^0$ equal to $C_0 = S^{-1}(0)$, which is exactly the case due to the values to which bytes K_{15}, K_8, K_4 and K_7 are set. Finally, to establish the equality of the LFSR values in (15), the expression defining

A are substituted into the way the LFSR is updated during the initialization procedure with the feed-forward, verifying that $s_{31}^1 = s_{31}^{'0} = K_{15} \oplus A$. As for the other LFSR values, $s_i^1 = s_i^{'0}$ holds directly due to the specification of K, IV, K', IV'.
Thus, the initialization procedures of the two cipher instances are slided, i.e., $IS^t = IS^{'t-1}$ for $1 \leq t \leq 64$. At time $t = 64$, in the (K, IV) instance of the cipher, a regular keystream step is applied, whereas in the (K', IV') instance, an initialization step is applied which destroys the slide property by introducing a difference between s_{31}^{65} and $s_{31}^{'64}$. However, it can be verified that this difference does not affect the two corresponding first keystream words, which proves (13). □
It should be noted that, as we verified by solving $B \oplus C_1 \oplus S_2(B \oplus C_1) = A$ for each $A \in F_2^8$, there always exists a byte B specified by this observation.

C Distinguisher performance for different (n, m)

The following table shows the numerical values for false positive and false negative probabilities for the distinguisher presented in section 3.2.

(n, m)	P[false positive] $\approx$	P[false negative] $\approx$
$(16, 6)$	$2^{-35.1}$	$2^{-22.41}$
$(16, 8)$	$2^{-50.4}$	$2^{-16.00}$
$(24, 8)$	$2^{-44.6}$	$2^{-24.01}$
$(24, 10)$	$2^{-59.2}$	$2^{-19.91}$
$(32, 10)$	$2^{-54.2}$	$2^{-27.6}$
$(32, 12)$	$2^{-68.3}$	$2^{-20.68}$

Table 1. Effectiveness of the distinguisher for different (n, m) parameters