State convergence and keyspace reduction of the Mixer stream cipher

Sui-Guan Teo[1], Kenneth Koon-Ho Wong[1], Leonie Simpson[1,2], and Ed Dawson[1]

[1] Information Security Institute,
Queensland University of Technology
{sg.teo,kkwong,e.dawson}@qut.edu.au
[2] Faculty of Science and Technology,
Queensland University of Technology
GPO Box 2434, Brisbane Qld 4001, Australia
lr.simpson@qut.edu.au

Keywords: Stream cipher, initialisation, state convergence, Mixer, LILI, Grain

Abstract. This paper presents an analysis of the stream cipher Mixer, a bit-based cipher with structural components similar to the well-known Grain cipher and the LILI family of keystream generators. Mixer uses a 128-bit key and 64-bit IV to initialise a 217-bit internal state. The analysis is focused on the initialisation function of Mixer and shows that there exist multiple key-IV pairs which, after initialisation, produce the same initial state, and consequently will generate the same keystream. Furthermore, if the number of iterations of the state update function performed during initialisation is increased, then the number of distinct initial states that can be obtained decreases. It is also shown that there exist some distinct initial states which produce the same keystream, resulting in a further reduction of the effective key space.

1 Introduction

Many keystream generators for stream ciphers are based on shift registers, particularly Linear Feedback Shift Registers (LFSRs). Using the output of a regularly-clocked LFSR directly as keystream is cryptographically weak due to the linear properties of LFSR sequences. To mask this linearity, stream cipher designers use LFSRs and introduce non-linearity either explicitly through the use of nonlinear Boolean functions or implicitly through through the use of irregular clocking.

Some stream ciphers use both explicit and implicit methods to provide non-linearity. For example, the LILI family of keystream generators [1, 5] produce keystream by using the contents of one LFSR to control the clocking of a second LFSR. A nonlinear filter function is applied to the contents of the second LFSR to produce the keystream. Thus the LILI family of keystream generators may be viewed as irregularly clocked nonlinear filter generators.

In this paper, we analyse the Mixer stream cipher [3], with a particular focus on the initialisation function. Mixer is another stream cipher which uses both implicit and explicit methods to provide nonlinearity. The explicit nonlinearity comes from the use of a nonlinear fedback shift register, while the implicit nonlinearity is derived from the use of a function which controls how many times a particular shift register is clocked.

In this paper we describe two major problems with Mixer. Firstly, the initialisation function of Mixer results in state convergence, so that by the time the initialisation phase is complete, multiple key-IV pairs have converged to produce the same initial state. Furthermore, the number of key-IV pairs which produce the same initial state increases as the number of iterations of the state update function performed during the initialisation increases. That is, the number of distinct initial states decreases as the number of iterations performed in the initialisation process increases. Secondly, the use of the shrinking generator style irregular clocking during keystream generation results in the existence of equivalent initial states - that is, distinct initial states which produce the same keystream. This is not a desirable feature.

The remainder of this paper is organised as follows. Section 2 describes the Mixer specification in detail. In Section 3, we present our analysis, outlining weaknesses in both the initialisation process and the keystream generation process. We demonstrate both the state convergence that occurs during the initialisation process and the circumstances under which Mixer will generate the same keystream even when the initial states are distinct. Section 4 compares the security provided by Mixer with that of two well-known ciphers which use similar components, and discusses how different compositions of similar components can affect the security. Section 5 presents our conclusions and outlines possible directions for future work.

2 Specification of the Mixer keystream generator

The Mixer keystream generator design has much in common with two well-known stream ciphers, the LILI family of stream ciphers [1, 5] and Grain-80 [2]. The clock-control operation from LILI and the identical nonlinear Boolean function from Grain-80 are used in Mixer to provide nonlinearity. Like both LILI and Grain, the Mixer design is based on two shift registers. Specific details of the structure, initialisation and keystream generation processes for Mixer are described below.

2.1 Components of Mixer

The keystream generator of Mixer [3] is based on two shift registers, denoted A and B, of lengths 128-bits and 89-bits, respectively. Register A is a regularly clocked LFSR with the feedback function $A(x)$. Register B is an irregularly clocked NFSR with the feedback function $B(x)$, with clocking controlled by register A. In this paper, we use the notation $A_t[i]$ to denote the contents of

the ith stage of register A at time t, where $i \in \{0, 1, \ldots 127\}$. Similarly, we use the notation $B_t[j]$ to denote the contents of the jth stage of register B at time t, where $j \in \{0, 1, \ldots 88\}$. The feedback functions for the two registers are as follows.

The feedback function of A is the weight 67 primitive polynomial $A(x)$ defined as:

$$\begin{aligned}A(x) &= 1 + x + x^6 + x^7 + x^8 + x^9 + x^{11} + x^{14} + x^{15} + x^{16} + x^{17} \\ &+ x^{18} + x^{23} + x^{25} + x^{29} + x^{30} + x^{35} + x^{36} + x^{39} + x^{40} \\ &+ x^{43} + x^{44} + x^{47} + x^{49} + x^{50} + x^{51} + x^{52} + x^{53} + x^{55} \\ &+ x^{56} + x^{57} + x^{58} + x^{60} + x^{61} + x^{65} + x^{66} + x^{67} + x^{70} \\ &+ x^{71} + x^{72} + x^{73} + x^{74} + x^{77} + x^{79} + x^{80} + x^{81} + x^{82} \\ &+ x^{87} + x^{90} + x^{94} + x^{96} + x^{102} + x^{103} + x^{104} + x^{105} \\ &+ x^{106} + x^{108} + x^{111} + x^{115} + x^{117} + x^{119} + x^{122} \\ &+ x^{123} + x^{124} + x^{125} + x^{126} + x^{128}\end{aligned}$$

where + denotes addition in GF(2).

The feedback function for NLFSR $B(x)$ is the composition of a linear feedback polynomial $B_L(x)$ and the nonlinear function $B_{NL}(x)$. That is, $B(x)$ is defined as:

$$B(x) = B_L(x) + B_{NL}(x) \tag{1}$$

where $B_L(x)$ is defined as:

$$1 + x + x^{39} + x^{42} + x^{53} + x^{55} + x^{80} + x^{83} + x^{89} \tag{2}$$

and $B_{NL}(x)$ is:

$$B_{NL}(x) = \prod_{j=1}^{88} (1 \oplus B_j(t)) \tag{3}$$

Note that $B_{NL}(x)$ is equal to 1 with probability 2^{-88}. In the analysis presented in this paper, the feedback function $B(x)$ is approximated by $B_L(x)$, an approximation that holds with very high probability.

The clocking of Register B is under the control of register A. An integer function, F_{INT}, takes the contents of w selected stages of register A as input and outputs an integer $c(b)$, which is the number of times register B is to be clocked. F_{INT} is defined as:

$$c_t(b) = F_{INT} = 1 + 2^0 \cdot A_t[i_0] + 2^1 \cdot A_t[i_1] + \ldots + 2^{w-1} \cdot A_t[i_{w-1}]$$

where $w \in \{0, 1, 2, \ldots 127\}$ and $i_0, i_1, \ldots i_{w-1} \in \{0, 1, 2, \ldots, 127\}$. Note that the Mixer specification does not fix the value for w, nor does it specify the tap

positions for the inputs to F_{INT}, but it does recommend that $w \in \{2, 3, \ldots 7\}$ be used for efficiency reasons.

A nonlinear Boolean function, $g(x)$, is used to determine whether the output of Register B will be used or discarded. The function $g(x)$ is defined as:

$$\begin{aligned} g(x) =& x1 + x4 + x0x3 + x2x3 + x3x4 + x0x1x2 + x0x2x3 + \\ & x0x2x4 + x1x2x4 + x2x3x4 \end{aligned}$$

The inputs $x0$, $x1$, $x2$, $x3$, $x4$ at time t are the contents of the five stages of register A: $A_t[7]$, $A_t[37]$, $A_t[73]$, $A_t[91]$ and $A_t[123]$, respectively.

Figure 1 illustrates the components of Mixer and the interaction of its components during both the initialisation (includes both solid and dotted lines) and keystream generation (solid lines only) processes.

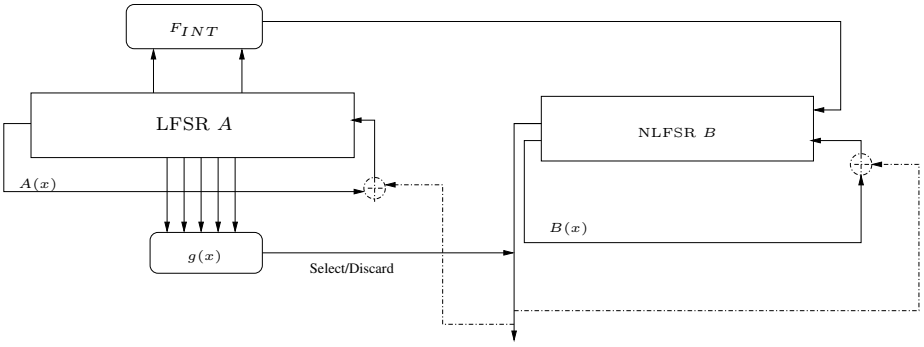

Fig. 1. Mixer state update functions

2.2 Initialisation of Mixer

The Mixer initialisation process can be divided into two phases: the loading phase, during which the key and IV are loaded into the shift registers; and the initialisation state update phase, where the key and IV values are 'mixed' to prepare an initial state before keystream generation begins.

Let the individual bits of the 128-bit key K be denoted $k_0, k_1, \ldots k_{127}$, and the individual bits of the 64-bit IV, V be denoted by $v_0, v_1, \ldots v_{63}$. The loading phase of Mixer is performed as follows. The key, K, is loaded into register A such that $A[i] = k_i$, for $0 \leq i \leq 127$. The IV, V, is loaded into register B such that $B[i] = v[i]$, for $0 \leq i \leq 63$. The remaining stages of register B are filled with ones. In this paper, we refer to a Mixer internal state at the completion of the loading phase as a *loaded state*.

To complete the initialisation process, the initialisation state update phase must be performed. This involves performing 200 iterations of the initialisation

state update function. Consider S_t, the internal state at time t, (the contents of the two registers A_t and B_t). For each iteration the following process, as shown in Figure 1 (with both solid and dotted lines), is followed:

1. Clock register A once.
2. For the updated state A_{t+1}, calculate:
   (a) The integer value $c_{t+1}(b)$ using F_{INT}.
   (b) The output of the nonlinear Boolean function $g(x)$
3. Clock register B $c(b)$ times.
4. If $g(x) = 0$ then this iteration is complete.
5. If $g(x) = 1$ then XOR the output bit of B (after $c(b)$ clocks) with the contents of both register stages $A[127]$ and $B[88]$.

In this paper, the XOR operation described in Step 5 is referred to as the *mixing* operation. This is the only operation in the initialisation state update function where contents of the two registers are directly combined. The output bit from register B which is XORed is referred to as *the mixing bit*, and denoted m.

During the initialisation phase, no keystream is produced. After 200 iterations of the initialisation state update function, registers A and B are in an *initial state* and ready to begin keystream generation.

2.3 Keystream Generation

During keystream generation the state update function is similar to the initialisation state update function. The only difference is that the mixing operation is not used, and the output of register B is used directly as the keystream. That is, for each iteration the following process, as shown in Figure 1 (with solid lines only), is followed:

1. Clock register A once.
2. For the updated state A_{t+1}, calculate:
   (a) The integer value $c_{t+1}(b)$ using F_{INT}.
   (b) The output of the nonlinear Boolean function $g(x)$
3. Clock register B $c(b)$ times.
4. If $g(x) = 0$ then this iteration is complete.
5. If $g(x) = 1$ then the output bit of B becomes the keystream bit z_j.

This process continues until sufficient keystream bits have been generated to encrypt or decrypt the message. Mixer is intended as a binary additive stream cipher, so encryption and decryption are performed by XORing the keystream produced with plaintext or ciphertext, respectively.

3 Analysis of Mixer

Mixer takes a 128-bit key, K, and a 64-bit IV, V, as inputs to the initialisation process. Thus, there are a total of $2^{128+64} = 2^{192}$ possible loaded states. The total Mixer internal state size is the sum of the lengths of registers A and B: $128 + 89 = 217$, so there are a total of 2^{217} possible internal states. Ideally, the initialisation process would result in 2^{192} distinct initial states for keystream generation, and Mixer could produce 2^{192} distinct possible keystream sequences. That is, each key-IV pair should produce a distinct keystream. However this is *not* the case for Mixer. This analysis explores two main causes for the reduction in the Mixer keystream generator: the internal state convergence that occurs during the initialisation process, and the existence of "equivalent states" — distinct initial states which produce the same keystream. These two problems each result in a reduction of the effective keyspace of Mixer.

3.1 Analysis of the Mixer initialisation process

Our analysis of the initialisation process is based on the observation that the initialisation state update function is not bijective. Considering the state transition in the forwards direction, given a value of S_t, there is a single next state value for S_{t+1}. However, when considering the state transition in the reverse direction, given a value of S_{t+1}, there may be multiple possible state values for S_t. That is, there is the possibility of state convergence occurring during an iteration of the initialisation state update function. The Mixer initialisation process involves 200 iterations of the state update function. In this section we examine the state convergence during one iteration of the initialisation state update function, and across the 200 iterations of the initialisation process.

State transition possibilities during initialisation. Recall the Mixer initialisation state update function given in Section 2.2 requires calculation of the output of the Boolean function $g(x)$, with the update of the two register stages $A[127]$ and $B[88]$ with the mixing value m conditional on the value of $g(x)$. The possibilities for the state transitions from S_t to S_{t+1} are:

1. $g(x) = 0$. No mixing operation occurs, regardless of the value of m.
2. $g(x) = 1$ and $m = 0$. The mixing operation occurs. However, as the mixing operation is $GF(2)$ addition, the contents of $A[127]$ and $B[88]$ remain unchanged after the mixing operation. That is, the outcome is the same as when $g(x) = 0$.
3. $g(x) = 1$ and $m = 1$: The mixing operation occurs, and the contents of $A[127]$ and $B[88]$ are complemented. We refer to this as *effective* mixing.

Since A has a primitve feedback function and $g(x)$ is a balanced nonlinear Boolean function, the probability that $g(x) = 1$ is expected to be very close to 0.5. Of the 4 possible $g(x)$ and m value combinations, effective mixing occurs with a probability of 0.25. Thus, when considering state convergence when

Mixer's initialisation function is being run forwards, the probability of state convergence occurring is also 0.25. When we consider all possible 217-bit states, and perform an initialisation process consisting of R iterations of the initialisation state update function, clearly, due to state convergence, the set of possible initial states that can be obtained will contain fewer than 2^{217} distinct states and could be estimated by $2^{217} \times 0.75^R$. Note that in the initialisation process we begin the state update phase of the initialisation process with only 2^{192} possible loaded states and not 2^{217}. If we assume that the iterations of Mixer's initialisation process are independent events then n_u, the upper-bound on the number of possible distinct initial states can be estimated by

$$n_u = 2^{192} \times 0.75^R \tag{4}$$

Consider inverting the initialisation state update function. That is, given S_{t+1} we want to obtain S_t. Both A and B are shift registers, and can be clocked in the reverse direction. Recall from 2.1 that register A is a regularly clocked LFSR, which controls the clocking of register B. The only uncertainty in performing this is whether there was *effective* mixing to create S_{t+1}. The possibilities for the state transitions from S_{t+1} to S_t are conditional on $g(x)$ and m:

1. $g(x) = 0$. No mixing occurred. In this case, we use A_{t+1} to calculate $c(b) = F_{INT}(A_{t+1})$, and clock register A back one time and register B back $c(b)$ times.
2. $g(x) = 1$. Mixing has occurred, but the effect depends on the value of m:
   (a) If $m = 0$ then (as for the case when $g(x) = 0$) use A_{t+1} to calculate $c(b) = F_{INT}(A_{t+1})$, and clock register A back one time and register B back $c(b)$ times.
   (b) If $m = 1$ then complement both $A[127]$ and $B[88]$, and then use A_{t+1} to calculate $c(b) = F_{INT}(A_{t+1})$, and clock register A back one time and register B back $c(b)$ times.

The value of $g(x)$ is readily obtained from A_{t+1}. The difficulty in inverting the state update function lies with computing the value of m. We cannot obtain this directly from B_{t+1} as it is discarded from register B after the mixing operation. Therefore, given $g(x) = 1$ we have two possibilities to consider (that m could have been either 1 or 0) corresponding to two possible previous states. If each of the iterations of Mixer's initialisation process resulted in state convergence, the number of possible distinct initial states, is given by

$$n_l = 2^{192} \times 0.5^R \tag{5}$$

This is possible but highly unlikely, as $g(x)$ is a balanced function, so Equation 5 forms a lower bound.

Bounds on number of equivalent loaded states Equations 4 and 5 above give an upper and lower bounds on the predicted number of distinct initial states that will be obtained after an R iteration process. Table 1 provides the computed

value for n_u and n_l for the various values of R between 0 and 200. From Table 1, for 200 iterations of Mixer's initialisation process, the lower bound on total number of distinct states would result in all possible loaded states converging to a single initial state and the upper bound would result in 2^{109} distinct initial states, which is less than the total key-space of Mixer.

R	0	25	50	100	150	200
n_l	2^{192}	2^{167}	2^{142}	2^{92}	2^{42}	1
n_u	2^{192}	$2^{181.6}$	$2^{171.2}$	$2^{150.5}$	$2^{129.7}$	2^{109}

Table 1. Bounds on the predicated number of distinct initial states, n_l and n_u, after an R iteration initialisation process

From Table 1 we observe two things. Firstly, is the large potential reduction of the possible state space from 2^{217} to 2^{109}. This is smaller 2^{128}, the total number of possible keys. This implies that there is the possibility that the same secret key and different IVs would produce the same initial state. Secondly, notice the trend that as R increases, the number of possible distinct initial states decreases. This implies that the more rounds of initialisation we perform, the number of distinct initial states decreases.

The bounds given in Table 1 do not take into account the formatting requirements when the IV is loaded in the NLFSR. Recall from Section 2.2 in the first phase of initialisation. The 64-bit IV is loaded into the 89-bit NLFSR and the remaining stages are filled with ones. Therefore, each loaded state will produce an initial state. However, if we take an initial state and invert the initialisation state update function R times, the set of possible states we obtain may include states that are not valid loaded states. We still need to check that the last 25-bits of the NLFSR are all-ones. This formatting requirement makes calculating the actual number of distinct states difficult, so we refer to the estimates provided in Table 1.

3.2 Equivalent states during Mixer keystream generation

During keystream generation, LFSR A is autonomous. That is, there is no mixing operation introducing values from register B into A. Therefore, state convergence due to the 'mixing' operation will not occur during the keystream generation process.

Assume that β is the total number of distinct initial states we can obtain after all 2^{192} possible key-IV pairs have been used in initialisation. At this point, we would expect that Mixer will produce β distinct keystreams. However, this is not the case. This is because the Mixer keystream generator employs a "shrinking generator" style mechanism to determine whether the output of register B will be used as a keystream bit or discarded. Thus it will also suffer from a known

weakness of shrinking generators. For the shrinking generator there exist distinct initial states, known as equivalent states, which produce the same keystream [4].

Recall from Section 2.3, if the value of $g(x)$ is 0, the output of register B is discarded and no keystream is produced. It is not until the value of $g(x)$ is 1 that the first bit of keystream is produced. Suppose we have an initial state, S_0^1, with component register states A_0^1 and B_0^1. For this initial state $g(x) = 1$, so the first keystream bit is produced immediately. Let the keystream produced when keystream generation is commenced in this state be denoted Z. Now consider an alternative initial state, S_0^0, with component register states A_0^0 and B_0^0. Suppose for this initial state $g(x) = 0$, so no keystream bit is produced. Further suppose that after the state update function is applied, $S_1^0 = S_0^1$. That is, we are now in the state from which the production of Z began. Therefore the two distinct states S_0^0 and S_0^1 can be considered equivalent, as both produce the same keystream sequence, Z.

Note that the binary sequence formed by successive outputs of $g(x)$ is the output of the nonlinear filter generator formed by applying $g(x)$ to the five stages of LFSR A. Statistics regarding run lengths for the binary sequences produced by LFSRs are well known, but less is known regarding the distribution of nonlinearly filtered LFSR. Analysis by Teo et al. [6] on the tuple distributions of nonlinear filtered LFSRs outputs show that they are not uniform for small LFSR sizes. This implies that the number of initial states generated for each keystream is not uniformly distributed. However, as $g(x)$ is balanced, we expect that $P(g(x) = 0) = 0.5$ so the number of distinct number keystreams is expected to be about half the number of distinct initial states $\frac{\beta}{2}$.

4 Security comparison with ciphers based on similar components

The Mixer design has some similarities to other well known stream ciphers. Like both the LILI [1, 5] and Grain [2] families of stream ciphers, Mixer is based on two shift registers. In fact, $B_L(x)$ used in Mixer is the same as the feedback function for the 89-bit LFSR used in LILI-128. Mixer also uses two additional functions: an integer clocking function and a nonlinear Boolean function as a filter function. The LILI keystream generators also use two such functions.

The Mixer integer clocking function is also similar to that used in the LILI keystream generators and the Mixer nonlinear filter function is the same as the filter function used in Grain-80. However, all three ciphers use these components in different ways. A summary of the components of all three ciphers is given in Table 2, and a brief outline of how the three ciphers use these components during initialisation and keystream generation is given in Table 3. For a full description of the operations, the reader is referred to the specification papers of the individual ciphers.

Cipher	LILI-II	Mixer	Grain
Register A	128-bit LFSR	128-bit LFSR	80-bit LFSR
Register B	127-bit LFSR	89-bit NLFSR	80-bit NLFSR
Irregular clocking	✓	✓	✗
Boolean function	12-bit Boolean function	Same $g(x)$ function	

Table 2. Comparison of components in Mixer, LILI and Grain.

Cipher	Initialisation	Keystream generation
Mixer	Output from nonlinear Boolean function is used to decide if the mixing operation occurs.	Output from nonlinear is used in a shrinking-generator fashion to decide if output of B is to be used as keystream bit.
Grain-80v1	Output from nonlinear function is XORed with the feedback bit in both registers at every clock	Output from nonlinear function is used as the keystream bit.
LILI-II	Output from nonlinear Boolean function is used to reload LILI-II's registers.	Output from nonlinear filter is used as the keystream bit.

Table 3. State update functions in Mixer, LILI and Grain.

4.1 Comparison between Mixer and Grain

As Table 3 shows, the initialisation function of Mixer and Grain are similar. However, there are some notable differences. For Grain, the nonlinear Boolean function takes selected stages *from both registers* as inputs to a nonlinear function, the output of which is used in the mixing operation. This mixing operation happens every time the registers are clocked, regardless of the output from the nonlinear filter. In comparison, the inputs to the Boolean function used in Mixer are selected stages *from the LFSR only*, and the output of the function is used in a shrinking-generator fashion, to determine which of the output bits from the NLFSR will be used in the mixing operation. This slight modification in the interaction between the three similar components during initialisation has a great impact on the security properties of both ciphers.

In the Mixer keystream generator proposal [3], the cryptanalysis section discusses a number of possible attack scenarios, including a chosen-IV attack. Kanso claims the initialisation process provides resistance to this type of attack, based on the work of Wu and Preneel [7] "that the initial states for any two chosen IV's are algebraically and statistically unrelated". This is the same claim made in the Grain proposal [2] on its resistance to chosen-IV attacks. Although the initialisation functions for Mixer and Grain are similar in terms of the mixing operation, they are not the same. Therefore the security provided by the initialisation function for Grain is not necessarily transferable to that of Mixer. However, as there are some similarities in the initialisation strategies, it is worth considering whether known attacks on the initialisation process of Grain can also be applied to Mixer.

Zhang and Wang [8] point out the existence of particular loaded states for which the Grain cipher will provide minimal security. They use the term, *a weak key-IV* to describe a loaded state which, after the initialisation process is complete, results in the LFSR being in an all zero state. When the LFSR is in an all-zero state after initialisation, the Grain keystream generator is only dependent on the NLFSR. In this case, Zhang and Wang were able to approximate the Grain NLFSR, mount distinguishing attacks, and recover the initial state of Grain using algebraic attacks. For an implementer of the Grain algorithm, it is important to check that neither register is in a all-zero state before keystream generation commences. Zhang and Wang calculated that Grain80-V1 could have as many as 2^{64} weak key-IV pairs. Having this many weak key-IV pairs could result in performance degradation if Grain80-V1 was used in applications requiring frequent re-keying, possibly making it unsuitable for time-critical applications.

The attack of Zhang and Wang could also be extended to find weak-key IV pairs for other states. For example, it could be possible to find weak-key IV pairs which would, after Grain's initialisation is complete, result in Grain's LFSR being in an all one state. In their paper, Zhang and Wang give a linear recursion for Grain's NLFSR [8]. Therefore, it could be possible to find weak key-IV pairs which would give the same initial state for both Grain's LFSR and NLFSR. However, such state could be more difficult to find, since both registers are regularly clocked, in contrast to Mixer, which uses irregular clocking.

Although the feedback functions of Mixer ensure that the all-zero state in Mixer for either register is not possible, this does not imply that the initialisation process is secure. Nor does it guarantee "'that the initial states for any two chosen IV's are algebraically and statistically unrelated"'. The implication of our findings on state convergence are that, in order to reduce the possibility of equivalent loaded states, we should perform few iterations of the initialisation state update function, and preferably not perform any initialisation operations at all, since for $R = 0$, $AST = 2^{217}$. However, not mixing the contents of registers A and B leaves Mixer vulnerable to other attacks. If no mixing occurs, then the contents of register B (from which the keystream is obtained) are known, and the key forms the initial state of LFSR A. Under these circumstances chosen-IV attacks and a differential approach may be quite effective in revealing the key.

4.2 Comparision of Mixer and LILI-II

Both LILI-II and Mixer use an integer function to control the number of times the clock-controlled LFSR is clocked. During LILI-II's initialisation function, two sets of 255-bit output sequences are generated. These sets of 255-bit output sequences are used to re-populate LILI's LFSRs. After the second 255-bit output sequence is loaded into the LFSRs, keystream generation is ready to commence.

This might seem similar to what Mixer and Grain do during their initialisation process, but there are two differences. The first difference is that while Mixer and Grain update their registers with the mixing bit using addition modulo 2, LILI *replaces* the contents of the registers with the 255-bit output sequence. Secondly, each reloading procedure of LILI requires 255-bits of output to be produced before the reloading operation commences. This requires 255 clocks of the clock-control register. For Mixer and Grain, the mixing operation can (could in the case of Mixer) occur each time a register is clocked.

The design aspect which caused state convergence in Mixer and the all-zero state LFSR state in Grain can be attributed to the way the mixing operation is implemented in Mixer and Grain. Since the same operation is not employed in LILI-II's initialisation function, the same vulnerability would not be present in LILI-II.

5 Conclusion

This paper presents an analysis of the Mixer keystream generator, with a particular focus on the initialisation process. Two major problems with this process are identified.

Firstly, the initialisation function of Mixer results in state convergence, so that multiple key-IV pairs (different loaded states) produce the same initial state for keystream generation, and hence the same keystream. This is due to a combination of irregular clocking and the mixing operation employed during the initialisation state update function. There are multiple key-IV pairs which converge to the same initial state. Furthermore, the number of key-IV pairs

which produce the same initial state increases as the number of iterations of the state update function performed during the initialisation increases. One way to prevent this state convergence problem is not perform any initialisation steps at all and immediately produce keystream the moment Mixer is in its loaded state. However, this could leave Mixer vulnerable to other cryptanalytic attacks.

Secondly, the use of the shrinking generator style irregular clocking during keystream generation results in the existence of equivalent initial states - that is, distinct initial states which produce the same keystream. This results in a further reduction of the effective key space.

Mixer has structural similarities with several other well-known ciphers, particularly the LILI and Grain families of keystream generators. Mixer also employs a similar initialisation strategy to that of the Grain family of stream ciphers. However, the ciphers are not identical and the security analysis provided for those ciphers does not suffice as a security analysis for Mixer. It is shown that Mixer does not have the weak Key-IV problem which exists for the Grain ciphers. However, the Mixer initialisation function has other fundamental flaws, identified in this paper. Based on our research, the Mixer cipher cannot claim to offer 128-bit security, and should not be considered suitable for cryptographic use where this is required.

References

1. Clark, A., Dawson, E., Fuller, J., Golić, J.D., Lee, H.J., Millan, W., Moon, S.J., Simpson, L.: The LILI-II Keystream Generator. In Batten, L.M., Seberry, J., eds.: ACISP 2002. Volume 2384 of Lecture Notes in Computer Science., Springer (2002) pp 25–39
2. Hell, M., Johansson, T., Maximov, A., Meier, W.: The Grain Family of Stream Ciphers. In Robshaw, M., Billet, O., eds.: New Stream Cipher Designs: The eSTREAM Finalists. Volume 4986 of Lecture Notes in Computer Science., Springer (2008) pp 191–209
3. Kanso, A.A.: Mixer — A new stream cipher. Journal of Discrete Mathematical Sciences and Cryptography **11**(2) (2008) pp 159–179
4. Simpson, L., Golić, J.D., Dawson, E.: A Probabilistic Correlation Attack on the Shrinking Generator. In Boyd, C., Dawson, E., eds.: Information Security and Privacy (ACISP 98). Volume 1438 of Lecture Notes in Computer Science., Springer (1998) pp 147–158
5. Simpson, L., Dawson, E., Golić, J.D., Millan, W.: LILI Keystream Generator. In Stinson, D.R., Tavares, S., eds.: SAC 2000. Volume 2012 of Lecture Notes in Computer Science., Springer (2000) pp 248–261
6. Teo, S.G., Simpson, L., Dawson, E.: Bias in the Nonlinear Filter Generator Output Sequence. In Ariffin, M.R.K., Ahmad, R., Said, M.R.M., Goi, B.M., Heng, S.H., Abu, N.A., Mas'ud, M.Z., eds.: Proceeding of Cryptology 2010; The Second International Cryptology Conference. (2010) pp 40–46
7. Wu, H., Preneel, B.: Chosen IV Attack on Stream Cipher WG. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/045 (2005) Available from `http://www.ecrypt.eu.org/stream/papersdir/045.pdf`,.

8. Zhang, H., Wang, X.: Cryptanalysis of Stream Cipher Grain Family. Cryptology ePrint Archive, Report 2009/109 (2009) Available from `http://eprint.iacr.org/2009/109`.