State convergence in bit-based stream ciphers*

Sui-Guan Teo, Harry Bartlett, Ali Alhamdan, Leonie Simpson, Kenneth Koon-Ho Wong and Ed Dawson †

Institute for Future Environments,
Science and Engineering Faculty,
Queensland University of Technology

11th February 2013

Abstract

Well-designed initialisation and keystream generation processes for stream ciphers should ensure that each key-IV pair generates a distinct keystream. In this paper, we analyse some ciphers where this does not happen due to state convergence occurring either during initialisation, keystream generation or both. We show how state convergence occurs in each case and identify two mechanisms which can cause state convergence.

1 Introduction

Modern stream cipher applications use a secret key and a publicly known initialisation vector (IV) to form an initial internal state for the keystream generator, before keystream generation begins. This approach is common in digital communications, where a single communication in frame-based applications can consist of multiple frames. A communication will use a single key for the entire communication and each frame will be encrypted using that key and a distinct IV. For example, a mobile phone conversation is divided into many frames. Each frame in the communication is encrypted separately using the same key and using the frame number as the IV. Given a suitable state size (at least equal to the sum of the key and IV lengths), a good initialisation process should ensure that each key-IV pair generates a distinct keystream. Furthermore, two keystreams which are distinct at the beginning of keystream generation should not converge to the same keystream sequence at a future point in time.

State convergence in a keystream generator occurs when two distinct internal states generate the same next state. That is, the state update function is not one-to-one. State convergence can occur either during initialisation, during keystream generation, or both. Hence, the state-update functions for both phases should be carefully considered.

This paper explores state convergence for several stream ciphers. We show how state convergence occurs in each case. Based on the analysis of the state-update function in these ciphers, we identify two mechanisms which can cause state convergence in stream ciphers.

*Portions of this paper are based on the authors' work on Sfinks [1] and A5/1 [21].

†E-mail: teosuiguan@gmail.com, {h.bartlett, lr.simpson, kk.wong, e.dawson}@qut.edu.au, a.alhamdan@student.qut.edu.au

2 Background and Notation

Keystream generators for stream ciphers operate by maintaining an internal state and applying update and output functions to the state. In many cases, the state space is provided by a combination of linear and/or nonlinear feedback shift registers (LFSR/NLFSR respectively). We use the notation $R_i(t)$ to denote the contents of stage i of register R at time t where $i = 0, 1, \ldots, r-1$, for an r-stage register. The state S of a stream cipher is of size s bits.

Modern keystream generators take two inputs: a secret key and an IV, of sizes l and j bits respectively and thus have a keyspace of 2^l bits and an IV-space of 2^j bits. Before keystream generation commences, a key-IV pair is used to form an internal state value. This process is referred to as initialisation and can be considered as a mapping from binary vectors of length $l + j$ to those of length s.

To prevent time-memory-data tradeoff attacks, modern stream ciphers have internal states which are at least the size of the key-IV pair, that is, $2^s \geq 2^{l+j}$. Since the state space is at least the size of the space spanned by all key-IV pairs, it is reasonable to expect that the initialisation process will be one-to-one, that is, each distinct key-IV pair should map to a distinct state at the end of the initialisation process.

The purpose of the initialisation process is to diffuse the key-IV pair across the entire state and make mathematical relationships between the key-IV pair and the keystream hard to establish. The initialisation process is often performed in three phases: key-loading, IV-loading and the diffusion phase. In some keystream generators, the key-loading and IV-loading phase are conducted simultaneously, that is, the secret key and IV are transferred to the stream cipher's state at the same time.

When both the secret key and IV have been transferred, the stream cipher is in a "*loaded state*". Following this, the diffusion phase begins. This consists of a number of iterations, denoted α in this paper, of the initialisation state-update function. The value of α requires careful consideration. A small number can be performed quickly, which is desirable in real-time applications where rekeying is frequent. However, an initialisation process with few iterations may not provide sufficient diffusion and could leave the cipher vulnerable to attacks. After the initialisation process is complete, the keystream generator is said to be in its *initial state* and the keystream generation phase begins.

During keystream generation, the internal state is updated with a state-update function and an output function is applied to this internal state to generate the keystream. The state-update function may be the same function used in the initialisation phase or a different function. This state-update function can be in the form of regular clocking or irregular clocking. For a regularly clocked shift register, the internal state of the shift register is always updated once at each clock. Clock-controlled keystream generators generally use the contents of one or more registers to control the clocking of itself and/or other registers. For irregularly clocked shift registers, the register state can be updated more than once or not at all at each clock.

When the state-update function is applied, for any state S_t, there is a single next state S_{t+1}. However, it is possible that there may be multiple states which have the same next state S_{t+1}. Once distinct states converge to the same next state, they can not subsequently diverge from that point on. Depending on the relevant state update functions, this convergence can occur during initialisation, keystream generation or both. Thus, the number of distinct keystreams the generator can produce is reduced. We note that although state convergence commonly occurs in irregularly clocked ciphers, it may also occur in regularly clocked ciphers as well.

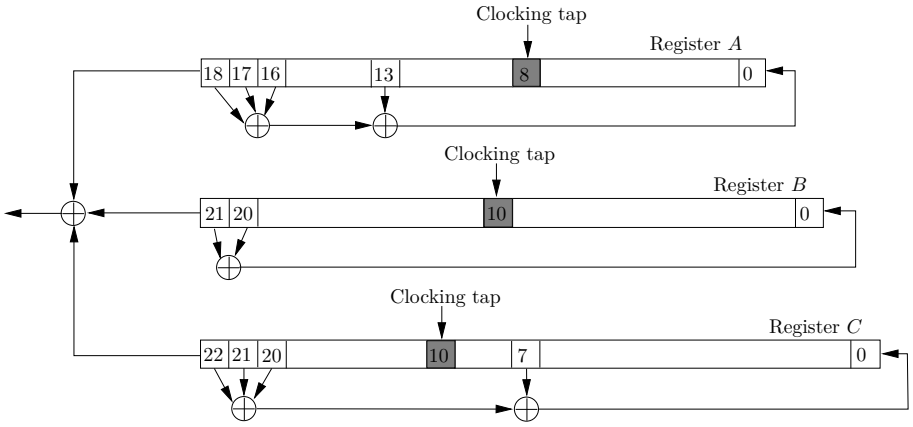

Figure 1: Diagram of A5/1

3 State convergence in irregularly-clocked stream ciphers

A number of stream ciphers which use irregular clocking mechanisms are known to experience state convergence. These ciphers include A5/1 [8] and Mickey [3].

3.1 A5/1

A5/1 is a well-known bit based stream cipher based on three binary LFSRs; denoted A, B and C; of lengths 19, 22, and 23 bits respectively. Hence A5/1 has a total state size of 64 bits. A single 64-bit secret key is used for each communication, and a 22-bit frame number is used as the IV for each frame in the communication. A diagram showing the structure of A5/1 can be found in Figure 1. The three registers are regularly clocked during loading of the key and IV (frame number). Following this, a majority clocking mechanism is used for the diffusion phase and also for keystream generation. This majority clocking is the only nonlinear operation performed.

To implement the majority clocking scheme, each register has a clocking tap: stages $A_8(t)$, $B_{10}(t)$ and $C_{10}(t)$. The contents of these stages determine which registers will be clocked at the next iteration: those registers for which the clock control bits agree with the majority value are clocked. For example, if $A_8(t) = 0$, $B_{10}(t) = 1$ and $C_{10}(t) = 0$, then the majority value is 0 and registers A and C are clocked. Thus, either two or three registers are clocked at each step.

3.1.1 Initialisation and keystream generation processes.

Prior to loading, all stages of the three registers are set to zero. Each register is autonomous during key and IV loading. During the key loading process, each register is regularly clocked 64 times and each key bit is XORed with the register feedback to form the new value of stage 0 of all three registers. A similar process is used for the IV loading process. The diffusion phase involves performing 100 iterations of the initialisation state update function using the majority clocking scheme. At the end of this phase an initial state is obtained.

Keystream is generated as a linear combination of the contents of one stage from each of the three registers: at time t, $z_t = A_{18}(t) \oplus B_{21}(t) \oplus C_{22}(t)$, where $\oplus$ denotes binary addition or xor. The majority clocking process continues until sufficient keystream, 228 bits, is generated to encrypt the frame. Then the keystream generator is re-initialised using the same secret key and the next frame number, to produce keystream for the next frame.

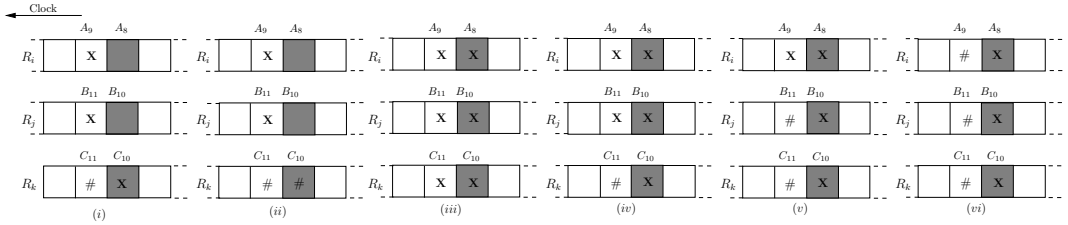

Figure 2: A5/1 preimage cases identified by Golić's cases [12]

Case	(i)	(ii)	(iii)	(iv)	(v)	(vi)
Proportion of states	$\frac{3}{8}$	$\frac{3}{8}$	$\frac{1}{32}$	$\frac{3}{32}$	$\frac{3}{32}$	$\frac{1}{32}$
Number of pre-images	0	1	1	2	3	4

Table 1: Proportions of states in A5/1 for Golić's cases [12]

3.1.2 State convergence in A5/1

During the loading phase in A5/1, the 64 bit key and 22 bit IV are linearly loaded into the register. As the size of A5/1's register is 64 bits, it is not possible for each key-IV pair to generate a unique loaded state. In fact, it can be shown that 2^{22} key-IV pairs correspond to each loaded state. This further compounds the state convergence which is experienced by A5/1 during initialisation and keystream generation.

Using majority clocking as the state-update function introduces nonlinearity during the diffusion phase and during keystream generation. In fact, it is the only nonlinear operation in A5/1. However considering the possible prior states for given states of a certain format or pattern reveals an interesting phenomenon. For some patterns there are no prior states while others have one or more prior states. Clearly the state update function is not one-to-one.

Golić [12] considered the inverse mapping for the majority clocking function and identified some states with no pre-image and which therefore cannot be reached from any loaded state in a single iteration. He demonstrated that these states comprise $\frac{3}{8}$ of the loaded states of the system. Thus, the usable state space shrinks by a factor of $\frac{5}{8}$ (from 2^{64} to $5 \times 2^{61} \approx 2^{63.32}$) at the first iteration of the diffusion phase. Golić also identified some states with unique pre-images and others with up to four pre-image states. Figure 2 presents a graphical summary of the six cases identified by Golić. In this figure, (R_i, R_j, R_k) is any permutation of the set $\{A, B, C\}$ of a particular set of registers stages in A5/1, where the shaded stage in each register is its clocking tap. The symbol **x** represents either 0 or 1, while # represents the complement of **x**; a blank square represents a bit which can take either value. The proportion of loaded states for each case in Figure 2 is presented in Table 1, along with the corresponding number of pre-images. Note that the case identified as (i) cannot be clocked back to any valid state. That is, states of this form cannot be reached after the first iteration of A5/1 initialisation state update function.

Biryukov et al. [6] also provided convergence estimates when exploring the efficiency of their proposed attacks on A5/1. They reported that, of 10^8 randomly chosen states, only about 15% can be clocked back 100 iterations. That is, 85% of chosen states could not be obtained after 100 iterations of the majority clocking process. A recent paper by Kiselev and Tokareva [17] analysed A5/1's state-update function and claimed that after eight iterations of the state-update function, the number of loaded states is reduced from 2^{64} to $2^{60.2355}$. However, this estimated appears to be flawed as it does not agree with our experimental work and Biryukov et al.'s [6] work.

α (number of iterations)	1	2	3	4	5	6
new proportion inaccessible	$\frac{3}{8}$	$\frac{3}{64}$	$\frac{9}{512}$	$\frac{57}{4096}$	$\frac{423}{32768}$	$\frac{6453}{524288}$
cumulative proportion inaccessible	0.375	0.422	0.439	0.453	0.466	0.479
proportion accessible	0.625	0.578	0.561	0.547	0.534	0.521
number of accessible states	$2^{63.322}$	$2^{63.209}$	$2^{63.165}$	$2^{63.129}$	$2^{63.094}$	$2^{63.061}$

Table 2: Proportion of available states in A5/1 after α iterations

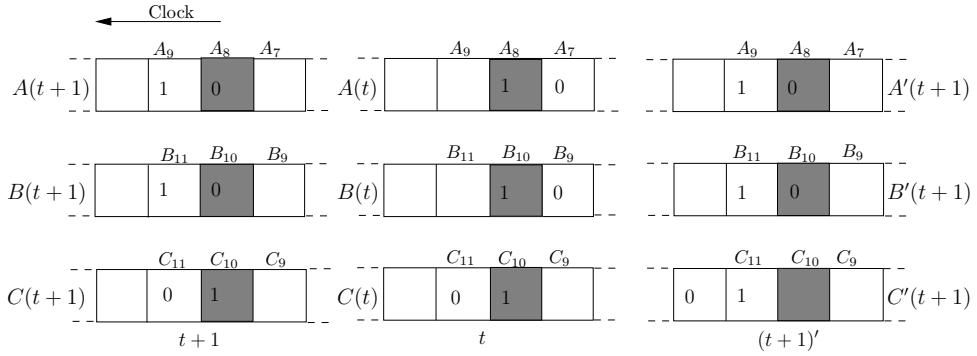

Figure 3: A5/1 preimage case(i) example

We explored the states which could not be accessed after $2 \leq \alpha \leq 6$ iterations. We note that the rate of state convergence is not uniform at each iteration. The proportion of inaccessible states increases as α increases. Table 2 summarises the proportion of inaccessible states for $1 \leq \alpha \leq 6$. From these results, we estimate that the number of accessible states at the end of initialisation ($\alpha = 100$) to be approximately 5% of the number of loaded states. However, computer simulations using a scaled-down version of A5/1 yielded an estimated number of accessible states after initialisation of approximately 19.2% of the number of loaded states. The latter result is similar to previous experimental results for A5/1 reported by Biryukov et al. [6].

The above analyses of the A5/1 state-update function demonstrate that increasing the number of iterations of the state-update function during initialisation decreases the number of distinct initial states. Also if two loaded state converge during initialisation, the same keystream will be produced in both cases.

Cause of state convergence in A5/1 We now illustrate why a particular A5/1 state, which falls into case (i) shown in Figure 3, cannot be reached after the first iteration of the A5/1 state-update function. Assume we have an A5/1 state at time $t+1$ whose register stages have the following values: $A_9(t+1) = B_{11}(t+1) = C_{10}(t+1) = 1$ and $A_8(t+1) = B_{10}(t+1) = C_{11}(t+1) = 0$. Recall that for A5/1's state-update function, register stages whose contents agree are clocked *once*. At time $t+1$, the contents of $A_9(t+1)$ and $B_{11}(t+1)$ agree, while the contents of $C_{11}(t+1)$ disagree. By applying the rules of A5/1's majority-clocking scheme, we clock A and B back once. That is, the contents of $A_9(t+1)$ and $B_{11}(t+1)$ are shifted to $A_8(t)$ and $B_{10}(t)$ respectively, while the contents of $A_8(t+1)$ and $B_{10}(t+1)$ are shifted to $A_7(t)$ and $B_9(t)$

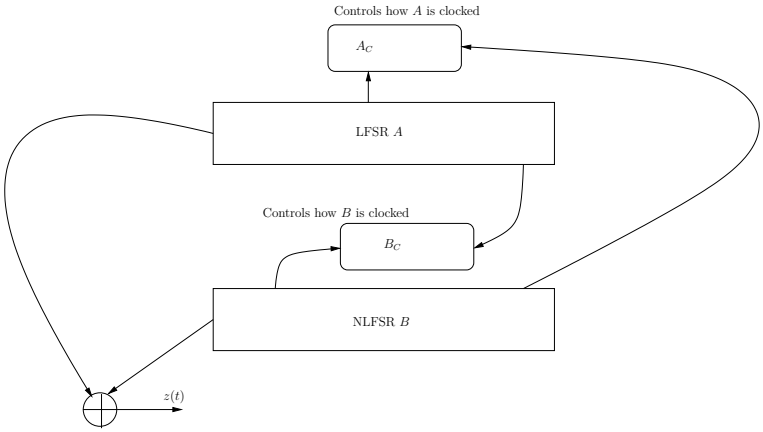

Figure 4: General diagram for the Mickey stream cipher

respectively. We do not clock C back as the contents of $C_{11}(t+1)$ does not agree with $A_9(t+1)$ and $B11(t+1)$. This single application of the majority-clocking scheme gives us the A5/1 state with the register contents A, B, and C at time t. The contents of the A5/1's clocking taps at time t are $A_8(t) = B_{10}(t) = C_{10}(t) = 1$. By applying the rules of A5/1's majority-clocking scheme to this A5/1 state, we clock all three registers once, as the contents of the clocking taps all agree. giving us the state at time $(t+1)'$. However, the contents of A5/1's state at time $(t+1)'$, consisting of $A'(t+1)$, $B'(t+1)$, and $C'(t+1)$ is not the same as $A(t+1)$, $B(t+1)$, and $C(t+1)$. Therefore, the state at time $t+1$ is not a state which can be attained after one iteration of A5/1 initialisation state update function. The register stages needed for analysing the number of pre-images which can be obtained when an A5/1 state at time $t+1$ is clocked back are A_8, A_9, B_i and C_i for $i \in \{10, 11\}$. If we were to treat these stages as a 6-tuple and examine the possible ways we can clock A5/1's registers backwards for each of the $2^6 = 64$ possible values the register stages A_8, A_9, B_i and C_i for $i \in \{10, 11\}$ can attain at $t+1$, we are able to obtain Golić's six preimage cases shown in Figure 2.

3.2 Mickey family of stream ciphers

Mickey-v1 is a bit-based stream cipher submitted to the eSTREAM project [11] by Babbage and Dodd. After state convergence was reported by Hong and Kim [14] in Mickey-v1, an updated version, referred to as Mickey-v2 [4], was submitted in 2006 and is one of the stream ciphers in the final eSTREAM portfolio. In this section, we describe the operation of Mickey-v1. The operation of Mickey-v2 is very similar but with details adjusted to suit the larger registers used.

Mickey-v1 has two 80-bit shift registers: LFSR A and NLFSR B, giving a total state size of 160 bits. These shift registers are configured as Galois style registers, that is, the contents of a single register stage are used to update multiple stages of the register. (By contrast, a Fibonacci style register updates a single stage of the register using a value calculated from several other stages.)

A diagram showing the components and their interactions is shown in Figure 4. An 80-bit key and an 80-bit IV are used to initialise the internal state.

3.2.1 Mickey initialisation and keystream generation

The basic idea of Mickey is that the registers A and B are mutually clocked, that is, the clocking of each register depends on values from both registers. Specifically, the clock control bit for LFSR A is $A_c = B_{27} \oplus A_{53}$, while the control bit for the NLFSR B is $B_c = B_{53} \oplus A_{26}$. If $A_c = 0$,

A is clocked once, whereas if $A_c = 1$, A is updated using an operation proposed by Jansen [15] which is equivalent to clocking A $2^{40} - 23$ times. Similarly, the choice of nonlinear function used to update register B depends on the value of B_c. This clocking scheme applies during both initialisation and keystream generation; the only thing that changes between these phases is how the feedback bits for the registers are calculated.

We denote the feedback bits for registers A and B as A_f and B_f respectively. During the various phases of operation, these feedback bits are calculated as follows:

- during IV loading, $A_f = A_{79} \oplus B_{40} \oplus v_i$, $B_f = B_{79} \oplus v_i$, (for $0 \leq i \leq 79$)
- during key loading, $A_f = A_{79} \oplus B_{40} \oplus k_j$, $B_f = B_{79} \oplus k_j$, (for $0 \leq j \leq 79$)
- during diffusion, $A_f = A_{79} \oplus B_{40}$, $B_f = B_{79}$, for 80 clocks.
- during keystream generation, $A_f = A_{79}$, $B_f = B_{79}$

During keystream generation, the keystream bit is calculated as $z(t) = A_0(t) \oplus B_0(t)$. This is done before the internal state is updated. The designers also impose a restriction on the amount of keystream which can be generated using a single key-IV pair: a maximum of 2^{40} bits should be generated before rekeying.

Since A is an LFSR, once the control bit for A is known, it is possible to calculate $A(t)$ given $A(t+1)$. Similarly, the feedback function of B is invertible for a given value of the control bit. That is, once the control bit of B is known, it is possible to calculate the previous internal state $B(t)$ given $B(t+1)$. However, despite the fact that the individual clocking functions for these two registers are invertible (one-to-one), the use of mutual clocking means that the combined clocking function is not necessarily one-to-one, as demonstrated below.

3.2.2 Current analysis of state convergence in Mickey-v1

State convergence in Mickey-v1 during keystream generation was reported by Hong and Kim [14]. They claim that this state convergence is due to the fact that the state-update function of Mickey-v1 uses two control bits, which consequently affect the very bits used to obtain the control bits, and claim that this self-dependent operation usually produces state convergence in the keystream generator. In their paper, they describe the algorithm they applied to determine how many pre-images a random state has. This algorithm is as follows:

1. Choose random states for both A and B.
2. Calculate the reverse clocking of register A, assuming the control bit is set to 0, and call this state A^0. Calculate what the state of A would have been assuming that the control bit was 1, and call this state A^1. Do the same for register B, and call the clocked-back states B^0 and B^1, where B^0 is the state B clocks back to when the control bit is 0, and B^1 is the state B clocks back to when the control bit is 1.
3. For each of the four possible (A, B) pairs after A and B are clocked back, calculate the two control bits, check to see if these match with the control bits actually used and count the number of matches.

For each of 2^{20} randomly chosen initial states, Hong and Kim ran the algorithm described above. They found that some states had none, one, two, or four pre-images. None of the 2^{20} randomly chosen initial states had three pre-images. The total number of pre-images Hong and Kim found for the 2^{20} randomly chosen states can be found in Table 3. Hong and Kim provided no

No. of pre-images	0	1	2	3	4
No. of states	307988	452017	279418	0	9153

Table 3: 2^{20} randomly chosen states, and the number of pre-images which produce them [14]

explanation as to why there were no states with three pre-images. In the following, we provide a theoretical argument for why this occurs.

The existence of state convergence in Mickey-v1 during keystream generation can be shown theoretically as follows. Start with a state $(A(t+1), B(t+1))$ and clock A and B back to A^0, A^1, B^0 and B^1, as above. We note that the two clocking alternatives for B produce identical results in the case where the feedback bit $B_f = 0$, so we separate our argument into two cases. (Note that $B_f(t) = B_{79}(t) = B_0(t+1)$ for both clocking alternatives.)

Case 1: $B_f(t) = 0$. As noted above, $B^0 = B^1 = B'$ (say) in this case and the value of B_c is irrelevant. However, $A^0 \neq A^1$. The conditions for A^0 and A^1 to be valid pre-images of $A(t+1)$ are that $A_c = B_{27} \oplus A_{53}$ in each case, that is:

- if $A^0_{53} = B'_{27}$, then (A^0, B') is a valid pre-image of $(A(t+1), B(t+1))$
- if $A^1_{53} = B'_{27} \oplus 1$, then (A^1, B') is a valid pre-image of $(A(t+1), B(t+1))$

Assuming that the contents of $A(t+1)$ are distributed uniformly, each of these conditions holds independently of the other with probability 0.5, so the probability that $(A(t+1), B(t+1))$ has no valid pre-images (conditional on $B_f(t) = 0$) is 0.25 and the (conditional) probabilities that this state has one or two valid pre-images are 0.5 and 0.25 respectively.

Case 2: $B_f(t) = 1$. In this case, $B^0 \neq B^1$ and we must consider the four potential pre-images (A^0, B^0), (A^0, B^1), (A^1, B^0) and (A^1, B^1). The conditions for these to be valid pre-images are (respectively);

1. a) $A^0_{53} = B^0_{27}$ and b) $A^0_{26} = B^0_{53}$
2. a) $A^0_{53} = B^1_{27}$ and b) $A^0_{26} = B^1_{53} \oplus 1$
3. a) $A^1_{53} = B^0_{27} \oplus 1$ and b) $A^1_{26} = B^0_{53}$
4. a) $A^1_{53} = B^1_{27} \oplus 1$ and b) $A^1_{26} = B^1_{53} \oplus 1$

By examining these conditions, we see that exactly zero, two or four of conditions 1a, 2a, 3a and 4a can hold, and similarly for conditions 1b, 2b, 3b and 4b. For example, if 1a and 2a both hold, it follows that $B^0_{27} = B^1_{27}$ and hence that either both or neither of 3a and 4a is true. The same argument applies if 1a and 2a are both false, while if only one of 1a and 2a is true, then $B^0_{27} = B^1_{27} \oplus 1$ and exactly one of 3a and 4a must be true. A similar analysis applies to conditions 1b, 2b, 3b and 4b.

If we again assume that the contents of $A(t+1)$ are distributed uniformly, conditions 1a and 2a each hold independently of the other with probability 0.5 but the probabilities of conditions 3a and 4a are dependent on how many of 1a and 2a are true. By considering these probabilities and then combining them appropriately with those for conditions 1b to 4b, we arrive at the probability distribution for the number N' of valid pre-images (conditional on $B_f(t) = 1$) shown in Table 4.

Finally, if we assume that the probabilities of $B_f(t) = 0$ and $B_f(t) = 1$ are equal, we obtain the unconditional distribution for the number of valid pre-images N of a randomly chosen state

N'	0	1	2	3	4
Prob(N')	$\frac{21}{64}$	$\frac{3}{8}$	$\frac{9}{32}$	0	$\frac{1}{64}$

Table 4: Pre-images and their probabilities when $B(t) = 1$

N	0	1	2	3	4
Prob(N)	$\frac{37}{128}$	$\frac{7}{16}$	$\frac{17}{64}$	0	$\frac{1}{128}$
Expected frequencies for 2^{20} states	303104	458752	278528	0	8192

Table 5: Combined distribution of number of pre-images during keystream generation

that is presented in Table 5. Table 5 also shows the expected frequencies that are obtained when this distribution is applied to a sample of 2^{20} states, as in Hong and Kim's experiments. While these expected frequencies are numerically close to Hong and Kim's experimental frequencies, the variation between them is statistically significant. We note, however, that Hong and Kim's observations are statistically consistent with a combined distribution involving a slight bias of $B_0(t+1) = B_f(t)$ towards the case $B_f(t) = 1$. As Hong and Kim note that the rand() function they used to generate random states "is known to be not so random" [14, p. 175], such a bias is possible and would provide a plausible explanation for the discrepancy between our analysis and their observations.

From both our theoretical analysis and the experimental work of Hong and Kim, we can see that many states in Mickey have either multiple pre-images or no pre-images, and hence it is clear that state convergence must occur in this cipher during keystream generation.

3.2.3 State convergence during initialisation

The above analysis can be extended to show that state convergence also occurs during the initialisation phase of Mickey.

During initialisation, the feedback bit $A_f(t)$ may sometimes be equal to $A_{79}(t)$ (as during keystream generation) and sometimes to $A_{79}(t) \oplus 1$. In the latter case, it is still possible to clock back uniquely from $A(t+1)$ regardless of the value of A_c. We will denote the resulting pre-images as A^{*0} and A^{*1} respectively. Likewise, $B_f(t)$ may sometimes be equal to $B_{79}(t)$ and sometimes to $B_{79}(t) \oplus 1$ and it is again possible to clock back uniquely for both values which B_c can attain, yielding pre-images B^{*0} and B^{*1} respectively. We also define $A^*(t+1)$ to be the state obtained from $A(t+1)$ by complementing all stages that are directly affected by the feedback bit during state update. Using the linearity of register A, it is easy to show that $A^*(t+1)$ clocks back to A^{*0} and A^{*1} when $A_f = A_{79}(t)$ and to A^0 and A^1 when $A_f = A_{79}(t) \oplus 1$. From the structure of A it can also be shown that $A^0_{53} = A^{*0}_{53}$ and $A^1_{53} = A^{*1}_{53}$, while $A^0_{26} = A^{*0}_{26}$ and $A^1_{26} = A^{*1}_{26} \oplus 1$.

We first consider the diffusion phase. Throughout this phase $B_f(t) = B_{79}(t)$, so $B(t+1)$ always clocks back to B^0 and B^1, but now $A_f(t) = A_{79}(t) \oplus B_{40}(t)$. We again consider two cases, according to the value of B_f.

If $B_f(t) = 0$, we again have $B^0 = B^1 = B'$. Depending on the value of B'_{40}, $A(t+1)$ will either clock back to A^0 and A^1 or to A^{*0} and A^{*1}. In the former case, the previous analysis applies verbatim, while in the latter case it applies with A^{*0}_{53} replacing A^0_{53} and A^{*1}_{53} replacing A^1_{53}, and hence the same probabilities are obtained as previously.

N'	0	1	2	3	4
Prob(N')	$\frac{5}{16}$	$\frac{7}{16}$	$\frac{3}{16}$	$\frac{1}{16}$	0

Table 6: Pre-images and their probabilities when $B_f = 1$ (initialisation)

N	0	1	2	3	4
Prob(N)	$\frac{73}{256}$	$\frac{29}{64}$	$\frac{31}{128}$	$\frac{1}{64}$	$\frac{1}{256}$

Table 7: Combined distribution of number of pre-images during initialisation

If $B_f(t) = 1$, we again have $B^0 \neq B^1$ and must again consider four potential pre-images. Now if $B^0_{40} = B^1_{40} = 0$ these are the same pre-images as previously and the same argument applies, while if $B^0_{40} = B^1_{40} = 1$ the potential pre-images become (A^{*0}, B^0), (A^{*0}, B^1), (A^{*1}, B^0) and (A^{*1}, B^1) and the argument applies with A^{*0} replacing A^0 and A^{*1} replacing A^1 throughout. If $B^0_{40} = 0$ but $B^1_{40} = 1$, the potential pre-images are (A^0, B^0), (A^{*0}, B^1), (A^1, B^0) and (A^{*1}, B^1); in this case, the conditions for these pre-images to be valid are

1. a) $A^0_{53} = B^0_{27}$ and b) $A^0_{26} = B^0_{53}$
2. a) $A^{*0}_{53} = B^1_{27}$ and b) $A^{*0}_{26} = B^1_{53} \oplus 1$
3. a) $A^1_{53} = B^0_{27} \oplus 1$ and b) $A^1_{26} = B^0_{53}$
4. a) $A^{*1}_{53} = B^1_{27} \oplus 1$ and b) $A^{*1}_{26} = B^1_{53} \oplus 1$

Since $A^0_{53} = A^{*0}_{53}$ and $A^1_{53} = A^{*1}_{53}$, the same remarks apply to conditions 1a–4a as previously; however, the fact that $A^0_{26} = A^{*0}_{26}$ and $A^1_{26} = A^{*1}_{26} \oplus 1$ means that exactly one or three of conditions 1b–4b are valid here. A similar argument applies to the final case ($B^0_{40} = 1$ and $B^1_{40} = 0$), and the constraints on conditions 1–4 in these two cases lead to the probability distribution for the number of valid pre-images to the state $(A(t+1), B(t+1))$ shown in Table 6. Combining these results with those for the first two cases and for $B_f(t) = 0$, we obtain the combined distribution table for the number of pre-images, as shown in Table 7.

Finally, consider the IV loading and key loading phases. During these phases, $B_f(t)$ may sometimes be equal to $B_{79}(t)$ and sometimes to $B_{79}(t) \oplus 1$. In the former case, $B(t+1)$ still clocks back to B^0 and B^1 but when the latter applies, we simply replace these by B^{*0} and B^{*1} in our argument. We again consider the cases when $B_f(t) = 0$ and $B_f(t) = 1$, and the arguments go through identically to those for the diffusion phase, except that all references to B^0_{40}, B^1_{40} and B'_{40} are replaced by $B^{*0}_{40} \oplus 1$, $B^{*1}_{40} \oplus 1$ and $B^{*'}_{40} \oplus 1$ whenever $B_f(t) = B_{79}(t) \oplus 1$ (that is, when $v_i = 1$ or $k_j = 1$). Thus, state convergence is also seen to occur during initialisation. By comparing the distributions in Tables 5 and 7, we see that the rates of convergence during initialisation and keystream generation are very similar.

Hong and Kim estimated that after 2^{40} iterations of the state-update function during keystream generation, Mickey-v1's internal state has lost approximately 39 bits of entropy. Mickey-v1's initialisation phase requires 240 ($2^{7.91}$) iterations of the state-update function. Extrapolating from the estimates given by Hong and Kim, one would expect that about seven bits of entropy will be lost during initialisation. However, the total convergence is virtually unchanged if a large amount of keystream is generated: as 240 is negligible compared to 2^{40}, the amount of entropy lost after initialisation and generating 2^{40} keystream bits will still be about 39 bits.

3.2.4 Mickey-v2

In response to the attacks by Hong and Kim, the designers of Mickey-v1 released a new version of Mickey, referred to as Mickey-v2 [4] in 2006. The register lengths of A and B were increased to 100 bits each, and the tap positions of the clocking bits were adjusted accordingly. The actual mutual clocking mechanism remains unchanged. As the general structure and components are similar to Mickey-v1, the reader is referred to Figure 4 for a diagram of Mickey-v2.

The designers argue that while state convergence is still possible, the expected entropy will drop from 200 bits to about 160 bits after 2^{40} keystream bits have been generated, and since this is "twice the key size, ... we no longer have a problem" [4].

This argument is clearly not correct. The initial entropy can not be larger than 160 bits of loaded key and IV data. Although the registers are now longer, as the mechanism which causes convergence is unchanged the number of distinct states will still decrease with additional iterations of the register update functions. In fact, the effectiveness of their strategy requires the assumption that the initialisation phase distributes the 2^{160} possible loaded states randomly within the state space of size 2^{200}. This assumption is equivalent (under the arguments of Hong and Kim) to having started with a full state entropy of 200 bits and then having undertaken approximately 2^{41} state update iterations. This then implies that the further 2^{40} iterations (for keystream generation) will result in an additional entropy loss of only approximately 0.6 bits $(\log_2(\frac{2^{41}+2^{40}}{2^{41}}))$. While this is not a large enough reduction to give rise to a serious attack, it does show that state convergence needs to be considered carefully and allowed for appropriately when designing ciphers. It is also clear that further entropy loss (state convergence) would result if a user of this cipher contravened the design restrictions and generated keystream in excess of the 2^{40} bits permitted.

4 State convergence in regularly-clocked stream ciphers

In the previous section, state convergence for the ciphers examined was associated with irregular clocking. However, the use of regular clocking is not sufficient to avoid state convergence. In this section we analyse some regularly clocked, bit-based stream ciphers for which state convergence occurs. These include the Sfinks stream cipher [7] and the Summation generator [19].

4.1 Sfinks stream cipher

The Sfinks stream cipher was submitted to eSTREAM project in April 2005, by Braeken et al [7]. It is a bit-based stream cipher that takes an 80-bit secret key and 80-bit IV as inputs to a 256-bit internal state. Sfinks consists of two main components: a 256 bit-based shift register, R, and a nonlinear one-to-one inversion function INV as shown in Figure 5.

The nonlinear function INV can be considered as a 16×16 bit S-box. The inversion function is used during both initialisation and keystream generation in different ways. Let x and y denote the 16-bit input and output of the S-box respectively, where $x = (x_{15}, \ldots, x_0)$ and $y = (y_{15}, \ldots, y_0)$. The 16 input bits are taken from 16 register stages and the 16 output bits are fed back to specified stages of the shift register during the initialisation as shown below; during keystream generation only one bit of the output of the S-box contributes to the formation of the keystream bit. The nonlinear function S-box is combined with a pipeline (memory) of seven 16-bit to store the output of the S-box. This memory is used to apply seven steps of delay to the

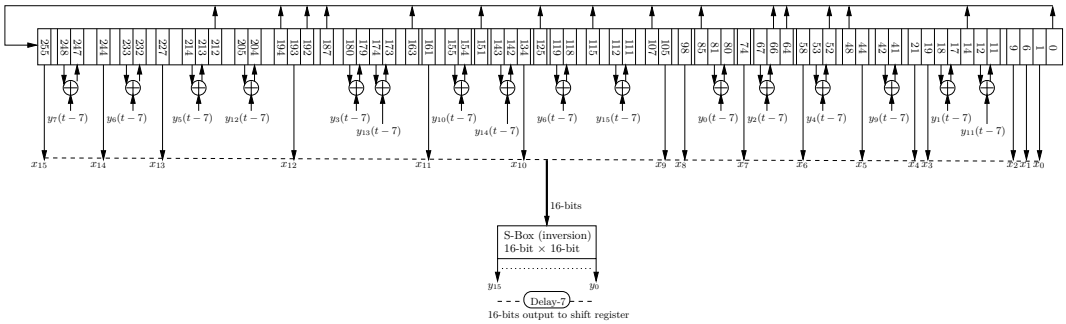

Figure 5: Initialisation process of Sfinks

feedback to the shift register.

$$(x_{15}, \ldots, x_0) = (R_{255}, R_{244}, R_{227}, R_{193}, R_{161}, R_{134}, R_{105}, R_{98}, R_{74}, R_{58}, R_{44}, R_{21}, R_{19}, R_9, R_6, R_1) \quad (1)$$

$$R_i(t) = R_{i+1}(t-1) \oplus y_{i \bmod 16}(t-7) \quad (2)$$

for $i = \{11, 17, 41, 52, 66, 80, 111, 118, 142, 154, 173, 179, 204, 213, 232, 247\}$.

4.1.1 Initialisation and keystream generation process

Sfinks loads 80-bit key and 80-bit IV to specific stages and added specific format of padding to the state to produce a 256-bit loaded state. The pipeline (memory) of the S-box is set to zeros at the beginning. The state update function is iterated 128 times to generate the initial state. Each iteration updates 17 stages from the shift register: 1 from the linear feedback and 16 from the output of the S-box. Figure 5 gives a general overview of state update function during the initialisation process. The state update function can be described as follows:

$$R_i(t) = \begin{cases} R_{i+1}(t-1) & \text{for } i = \{0, 1, \ldots, 254\} \text{ except } \{11, 17, 41, 52, 66, 80, \\ & 111, 118, 142, 154, 173, 179, 204, 213, 232, 247\} \\ R_{i+1}(t-1) \oplus y_{(i \bmod 16)}(t-7) & \text{for } i = \{11, 17, 41, 52, 66, 80, 111, 118, 142, 154, \\ & 173, 179, 204, 213, 232, 247\} \\ \bigoplus_j R_j(t-1) & \text{for } i = 255 \\ & \text{for } j = \{212, 194, 192, 187, 163, 151, 125, 115, 107, \\ & 85, 66, 64, 52, 48, 14, 0\} \end{cases}$$

Once the initialisation process is completed, keystream generation can begin. During the keystream generation, the register feedback is linear and the least significant bit of the 16-bit output value of the S-box is XORed with the value of stage R_0 to produce a keystream bit, with a delay of 7 steps applied to both values. That is, $z_t = R_0(t-7) \oplus y_0(t-7)$.

4.1.2 State convergence in Sfinks

In our analysis, the stages of R which provide inputs to the S-box or receive outputs from the S-box are denoted as input and output stages respectively. We note that the distance between

certain input and output stages is equal to the delay length of seven steps, and observe that this correspondence can lead to state convergence, as follows.

Suppose that R_{i+7} is an input stage and R_i is an output stage; then

$$\begin{aligned} R_i(t) &= R_{i+1}(t-1) \oplus y_{(i \bmod 16)}(t-7)) \\ &= R_{i+7}(t-7) \oplus y_{(i \bmod 16)}(t-7)) = \bar{R}_{i+7}(t-7) \oplus \bar{y}_{(i \bmod 16)}(t-7)) \end{aligned} \tag{3}$$

where $\bar{R}$ and $\bar{y}$ represent the complements of R and y respectively. If complementing the contents of the input stage $R_{i+7}(t-7)$ (with other inputs unchanged) causes the output bit $y_{(i \bmod 16)}(t-7))$ to change also then state convergence occurs, since two separate states at time $t-7$ both lead to the same state at time t.

An examination of the register R shows that there is only one input stage with a distance to the next output stage equal to seven steps; this is $R_{161} = x_{11}$, with corresponding output stage R_{154} receiving the output y_{10}. According to Equation 3, $R_{154}(t)$ will be unchanged if complementing x_{11} results in y_{10} also being complemented. We therefore look for pairs of S-box inputs $(x_{15}, \ldots, x_0)$ which differ only in bit x_{11} and for which the corresponding pair of outputs $(y_{15}, \ldots, y_0)$ differ in bit y_{10}.

In general, the other output bits may vary, except for y_0, y_1, y_4, y_9, y_{11} and y_{15}. Therefore, we consider pairs of the S-box inputs which differ only in x_{11} and for which the output bits differ in y_{10} and are the same in y_0, y_1, y_4, y_9, y_{11} and y_{15}.

By using an exhaustive search process, 273 pairs of the S-box inputs (and corresponding outputs) with such patterns were found. Table 8 gives an example of one S-box input and output pair. Note that the condition of difference is the underlined bold bit x_{11} and y_{10} and the output bits y^0, y^1, y^4, y^9, y^{11} and y^{15} (shown in italics) should be the same in each pair. Table 9 provides an example of two states based on that pattern that converge to the same state after 7 iterations.

	Input	Output
S-box sequence	$x^{16} \ldots\ldots\ldots\ldots\ldots\ldots x^1$	$y^{15} \ldots\ldots\ldots\ldots\ldots\ldots y^0$
Stage No.	255 244 227 193 161 134 105 98 74 58 44 21 19 9 6 1	111 142 173 204 11 154 41 232 247 118 213 52 179 66 17 80
1st value	0100**0**000000000000	*0100**0**10111111111*
2nd value	0100**1**000000000000	*0010**1**10001101111*

Table 8: A pair of S-box inputs and outputs that can causes state convergence

$R_A(t)$	**3**19B7E15AF4FF1318**9D**F0800AD**A**C56A4**0**913E4B90CBEEBD3A0074AFD3351E58**0**
$R_B(t)$	**7**19B7E15AF4FF1318**DC**F0802AD**8**C56A4**2**913E4B90CBEEBD3A0074AFD3351E58**1**
$R(t+7)$	3EE336FC2B7E9FE2631BBE10015B18AD485227C972187DD3A7500C95FA64A3CB

Table 9: Two states (hex) which converge after 7 clocks

From above, there are 273 pairs of the S-box inputs satisfying the convergence conditions out of the 2^{15} possible input pairs. If we assume the possible values for the S-box input bits are distributed randomly and independently, this state convergence has a probability of $\frac{273}{2^{15}} = 2^{-6.9}$ at each step. There are 120 iterations of the initialisation process at which state convergence can occur, so the approximate number of reachable distinct states at the end of initialisation is $(1-2^{-6.9})^{120} \times 2^{160} = 2^{158.55}$.

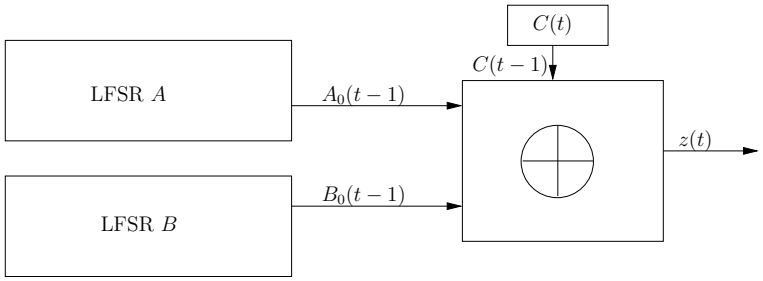

Figure 6: Summation generator diagram

The individual components (linear feedback and the S-box) of the state update function are one-to-one. However, the combination of these components which is used during the initialisation process is not one-to-one. Therefore, state convergence can occur in Sfinks.

4.2 Addition-with-carry keystream generators

In this section, we discuss two keystream generators which use an Addition-with-carry mechanism to generate keystream. These are the summation generator and the F-FCSR stream cipher.

4.2.1 Summation generator

The summation generator [19] is a bit-based stream cipher proposed by Ruppel in 1985. The original summation generator proposed by Ruppel consisted of two binary symmetric source output A and B of sizes a bits and b bits respectively, and a single memory bit C. This gives the summation generator a state space S of $s = a + b + 1$ bits. The keystream output from a summation generator is formed by combining the output of A and B and the previous contents of C. Therefore, a summation generator can be viewed as a nonlinear combiner with a single memory bit. In this section, the two binary symmetric source outputs are assumed to be two LFSRs A and B, whose state-update functions are implemented using primitive feedback polynomials. The output of two LFSRs A and B of lengths a and b, at time t is combined with a memory bit $C(t-1)$ to form a keystream bit $z(t)$, and the memory bit $C(t)$ is calculated using the following functions respectively:

$$z(t) = A_0(t-1) \oplus B_0(t-1) \oplus C(t-1) \tag{4}$$

$$C(t) = A_0(t-1)B_0(t-1) \oplus (A_0(t-1) \oplus B_0(t-1))C(t-1) \tag{5}$$

A diagram of the summation generator can be found in Figure 6. In the next section, we show why state convergence occurs in the summation generator and estimate the loss of state entropy in the summation generator arising from this state convergence.

State convergence in the summation generator If the state-update functions of A and B are implemented using primitive feedback polynomials, there is a one-to-one mapping from a particular state of either shift register at time t to another state at time $t + 1$. Furthermore, the memory bit C is not used to update A or B. Thus, no state convergence occurs in either register A and B.

However, when we consider the entire state of the keystream generator, we see that it is possible for combinations of $A_0(t-1)$, $B_0(t-1)$ and $C_0(t-1)$ to generate the same $C(t)$ value. Table 10 shows how the keystream bit $z(t)$, and the memory bit $C(t)$ are calculated based on the values of $A_0(t-1)$, $B_0(t-1)$, and $C(t-1)$. From Table 10, we can observe that when

$A_0(t-1)$	$B_0(t-1)$	$C(t-1)$	$C(t)$	$z(t)$
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Table 10: State transition table for $C(t)$ and keystream generation output

$A_0(t-1) = B_0(t-1)$, the calculation of the value $C(t)$ is not affected by $C(t-1)$. Consider two distinct summation generators with states at time $t-1$ comprising $A(t-1) = A'(t)$, $B(t) = B'(t)$ and $C(t) \neq C'(t)$. For example, let $A_0(t-1) = A'_0(t-1) = B'_0(t-1) = B'_0(t-1) = 0$, and $C(t-1) = 1$, and $C'(t-1) = 0$. At time t, $A(t) = A'(t)$ and $B(t) = B'(t)$ have the same values, as the state-update functions for A and B are autonomous and are not affected by the value of $C(t-1)$. However, note the value of $C(t)$ is now 0. Hence, two distinct states $A(t-1), B(t-1), C(t-1)$, and $A'(t-1), B'(t-1), C'(t-1)$ have now converged to the same state, and will generate the same keystream from this point on. Likewise, the states

- $A'_0(t-1) = B'_0(t-1) = 1$, $C(t-1) = 0$ and
- $A'_0(t-1) = B'_0(t-1) = 1$, $C(t-1) = 1$

will also converge to the same state at the next clock step, and will produce identical keystream from that point on. If the initial state of a summation generator was initialised randomly, $\frac{1}{2}$ of the states can be paired up, and each pair will generate the same state at time $t+1$. At time $t+2$, $\frac{1}{4}$ of the initial states can be paired up, and each pair will generate the same state at time $t+3$. Therefore, after two iterations of the summation generator's state-update function, $\frac{1}{2} + (\frac{1}{2} \times \frac{1}{2}) = \frac{3}{4}$ of the states would have experienced state convergence. After α iterations, the summation generator would have lost $(\sum_{i=0}^{\alpha} \frac{1}{2^\alpha})$ bits of entropy, approximately 1 bit for suitably large α. As mentioned above, if A and B are autonomous LFSRs, state convergence will not occur in either register. In this section, it is shown that state convergence can occur in the carry register C. Thus, the summation generator will at most lose one bit of entropy, resulting from the state convergence caused by C. As a result, the effective state space of the summation generator is $a + b$ bits.

4.2.2 State convergence in the F-FCSR stream cipher.

The F-FCSR stream cipher [2], designed by Arnault and Berger in 2005 also uses a carry-register based design. It uses a 128-bit key and a 64-bit IV to initialise a 196 bit internal state. Studies by Jaulmes and Muller [16] and Röck [18] show that state convergence also occurs in F-FCSR. In particular, Röck showed that that the effective state size of F-FCSR will always be approximately 128 bits.

Type of mechanism	Stream ciphers
Mutual clock-control	• A5/1 • Mickey-v1/v2
Self-update	• Sfinks • Summation Generator

Table 11: Causes of state convergence summary table

5 Mechanisms which can cause state convergence

So far in this paper, we have studied stream ciphers which have state convergence problems. Based on this study, we have identified two mechanisms which can cause state convergence. These are:

- Mutual clock-control,
- Self-update mechanisms

The classification of the studied stream ciphers into the two categories is shown in Table 11. All the mechanisms shown in Table 11 are nonlinear operations. However, the types of registers used in the stream ciphers can either be linear and/or nonlinear. The stream ciphers which update the internal state in a linear way do so via the XOR operation. For A5/1, this is accomplished through the feedback polynomial of the LFSR. For Sfinks, the update of the internal state is also accomplished through the LFSR's feedback polynomial, along with the direct XOR of S-Box output bits with specified bits of the LFSR. The remaining ciphers, the Summation generator and Mickey, consist of ciphers which use a combination of linear and nonlinear functions to update the internal state of the keystream generator. In Section 5.1 and Section 5.2, we discuss why these mechanisms can cause state convergence in keystream generators.

5.1 Mutual clock-control

Mutual clock-control mechanisms are the combination of two different mechanisms: mutual update, and clock-control mechanisms. In this section, we discuss both mechanisms and discuss the situations where state convergence may or may not occur in keystream generators which use clock-control mechanisms in their keystream generators.

5.1.1 Mutual-update mechanisms

Mutual-update mechanisms are mechanisms where each register is updated using input from the stages of another register. That is, none of the registers are autonomous. The use of mutual-update does not cause state convergence by itself. For example, a mutual update mechanism is used for the state-update function of Trivium [9]. It should be noted that the state-update function of Trivium is reversible, that is, each internal state has only one prior state and one next state. Therefore, state convergence will not occur either during initialisation, or keystream generation.

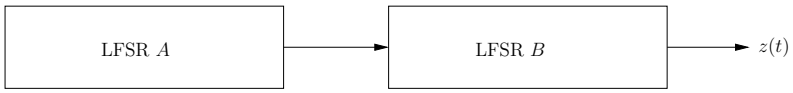

Figure 7: Step-1/2 generator

5.1.2 Clock-control mechanisms

Clock control mechanisms are used in keystream generators which have two or more registers. They typically use an integer function which takes inputs from selected stages of a particular register. Clock-control mechanisms can be found in ciphers like the Step-1/2 generator proposed by Gollmann and Chambers [13], and the LILI family of stream ciphers [10, 20]. In the Step-1/2 generator, clocking register A controls how many times the clock-controlled generator B is clocked for. If the output of A is 0, B is clocked once, and if the output of A is 1, B is clocked twice. A diagram showing the components in the Step-1/2 generator can be seen in Figure 7. State convergence will not occur in the Step-1/2 generator, as the state-update function is bijective. Given a state $S(t+1)$ at time $t+1$, we perform the following operations to obtain the unique pre-image of $S(t+1)$.

1. Calculate the previous output bit of A by clocking A backwards once.
2. Once the previous output bit of A is obtained, we know how many times B was clocked, and can clock it the appropriate number of times to obtain the state at time t.

The LILI family of stream ciphers make use of an integer function during the diffusion and keystream generation process. At time t, this integer function I_B takes as input stages from the clock-control register A and outputs the integer value $I_B(t)$. The clock-controlled register B is then clocked $I_B(t)$ times. Similar to the Step-1/2 register, this state-update function is bijective. Given a state at time $t+1$, we perform similar operations to obtain $t+1$'s unique pre-image.

1. Calculate the state of A at time t by clocking A backwards once.
2. Once the previous state of A is obtained, calculate the integer value $I_B(t)$ obtained at time t using the integer function, and clock B backwards that number of times.

Biham and Dunkelman [5] observed that the LILI-II stream cipher may also experience state convergence during the loading phase[1]. During LILI-II's loading phase, the 128 bit register A is loaded using the XOR of the 128 bit key and 128 bit IV. To load the 127 bit register B, we first drop the first bit of the key, and drop the last bit of the IV, then XOR the two 127 bit values together. During the diffusion phase, 255 bits of output is generated. This 255 bit value is loaded into the two registers (128 bits in A, and 127 bits in B). The cipher is run again to produce 255 bits of output. As before, this 255 bit value is loaded into the two registers (128 bits in A, and 127 bits in B). At this point, LILI-II is in an initial state and ready to produce keystream.

However, the linear operation of XORing the key and IV together causes state convergence during the loading phase. Biham and Dunkelman [5] noted that if two distinct keys-IV pairs (K^1, V^1), and (K^2, V^2) satisfy the differential $K^1 \oplus K^2 = V^1 \oplus V^2 = 1^{128}$, the 255 bit output produced during the diffusion phase will be the same for the two key-IV pairs. Since the same 255 bit output is directly loaded into the two registers to form the state which is used during the diffusion phase, the two key-IV pairs will produce the same initial state and consequently, the same keystream. To prevent this from occurring, XORing the key-IV pair during the keystream

[1]State convergence does not occur in LILI-128, as the 128-bit secret key was used to form the initial state directly.

$B_{12}(t)$	$B_{20}(t)$	$I_A(t)$	$A_{12}(t)$	$A_{20}(t)$	$I_B(t)$
0	0	1	0	0	1
0	1	2	0	1	2
1	0	3	1	0	3
1	1	4	1	1	4

Table 12: Output of I_A and I_B based on their inputs.

generator's loading process should be avoided, and a direct loading of the key and IV into a keystream generator's internal state should be used instead.

Before we explain why state convergence occurs in mutually clock-controlled ciphers, we discuss why state convergence *does not* occur in both the Step-1/2 generator, and the LILI family of stream ciphers during keystream generation. In both of these keystream generators, the bijectiveness of the state-update function is due to the fact the register A is autonomous and regularly clocked, while register B relies on some output derived from selected stages of register A to determine how many times to clock. Analysing the state update function for A and B, we know that since B relies on the output of A to determine how many times B is clocked, it would be possible for two distinct B states , B, and B', at time t to converge to the same state at time $t+1$ for two different clocking values obtained from register A. However for that to happen, register A at time $t+1$ must have had two distinct preimages, A and A', at time t, since a different number of clocks would be required for the two B distinct states at time t to be equal at $t+1$. But register A can only have one pre-image at time t, since register A is clocked regularly, so state convergence is not possible for the state-update function in the Step-1/2 generator and the LILI family of stream ciphers. In contrast, using outputs from a clock-controlled register to control how the clock-control register is clocked has the potential to cause state convergence. To illustrate why this can occur, we use a modified version of LILI, LILI-M1 as a case study.

5.1.3 A case study in state convergence in mutual clock-control stream ciphers

The LILI-family of stream ciphers consists of the stream ciphers: LILI-128 [10] and LILI-II [20]. Although both ciphers differ in the various functions used, the structure of both ciphers can be generalised. This structure consists of two LFSRs: LFSR A, LFSR B, a nonlinear output function g and an integer function I_B. I_B takes as input selected stages from A and outputs an integer $I_B(t)$. $I_B(t)$ determines how many times register B is clocked.

In LILI-M1, we retain these components and add an additional integer function I_A, which takes as input selected stages from register B and outputs an integer $I_A(t)$. $I_A(t)$ determines how many times register A is clocked. The functions for I_A and I_B output the integer value $I_A(t)$ and $I_B(t)$ using the following equations.

$$I_A(t) = 2 \times B_{12}(t) + B_{20}(t) + 1$$
$$I_B(t) = 2 \times A_{12}(t) + A_{20}(t) + 1$$

The outputs of I_A and I_B can be seen in Table 12. During each iteration of LILI-M1's state-update function, the integer values $I_B(t)$ and $I_A(t)$ are calculated from the internal state $A(t)$ and $B(t)$ respectively. Registers $A(t)$ and $B(t)$ are then clocked $I_A(t)$ and $I_B(t)$ times respectively. Diagrams showing the difference in structures of the original LILI family of stream ciphers and LILI-M1 is shown in Figure 8 and Figure 9 respectively.

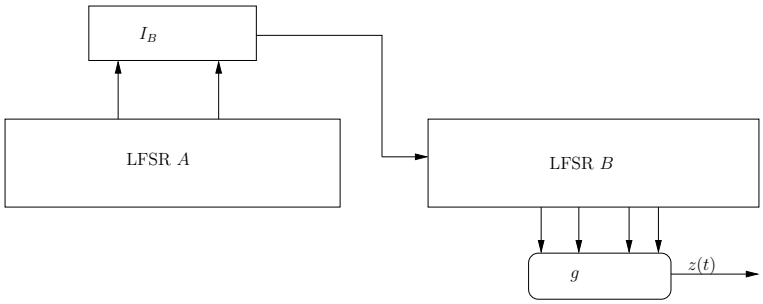

Figure 8: General structure and components of the LILI keystream generators

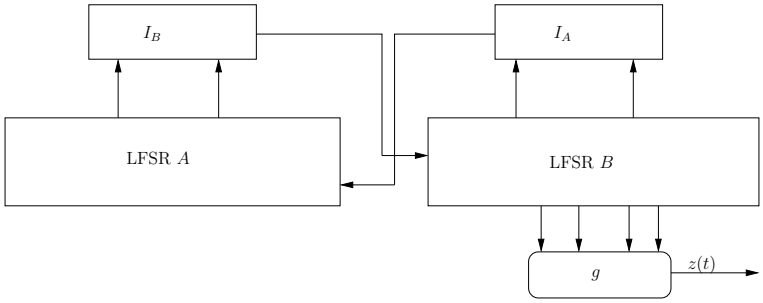

Figure 9: Structure and components of LILI-M1

Determining if state convergence occurs in LILI-M1. Computer simulations were performed to determine if state convergence occurs in LILI-M1. The stages used as inputs to the integer functions I_A and I_B are B_{12} and B_{20}, and A_{12} and A_{20}, respectively. Since both registers A and B are LFSRs, the contents of $B_{12}(t+1)$ and $B_{20}(t+1)$, and $A_12(t+1)$ and $A_{20}(t+1)$ when the state-update functions of registers A and B are reversed will be replaced by the contents of stages immediately downstream at time t. The clocking rules for I_A and I_B means that each register can be clocked back either one, two, three or four times at each iteration. This means, for each of the four stages used in the functions I_A and I_B, we have to consider the contents of four stages immediately downstream. The contents of these stages can also be treated as a 16-bit pattern. As contents of stages B_{12}, B_{20}, A_{12} and A_{20} at time $t+1$ do not affect the results of clocking results of I_A and I_B, we can set the contents of these registers to arbitrary bits. For the purposes of our experiments, these four stages are set to zero. For each possible 16-bit pattern, we clock both registers backwards assuming the values of I_A and I_B were either of the values shown in Table 12. After both registers have been clocked backwards, the contents of the register stages B_{12}, B_{20}, A_{12} and A_{20} at time t is then used to calculate the values of I_A and I_B. If the values of I_A and I_B match the integer values used to clock registers A and B back, we record it was a valid pre-image. Otherwise, the state at time $t+1$ is considered as a state which could not been obtained by a state at time t and we regard the state at time $t+1$ was a state with no pre-image.

The results of our experiments are shown in Table 13. This table shows how many pre-images each of the 65536 internal states have. It is clear from Table 13 that state convergence occurs in LILI-M1, as there are states which can have 0,2,3,4 pre-images.

Why state convergence occurs in LILI-M1. Assume that the current internal state of LILI-M1 is $(A(t+1), B(t+1))$. To calculate how many times $A(t+1)$ needs to be clocked back in LILI-M1, we need to know the value of $I_A(t)$. The determination of $I_A(t)$ requires knowledge of what the state $B(t)$ was, which in turn requires knowledge of $I_B(t)$. Determining $I_B(t)$ requires

No. of pre-images	No. of states
0	16920
1	33184
2	13968
3	1440
4	24

Table 13: No. of pre-images which can be obtained for 65536 states

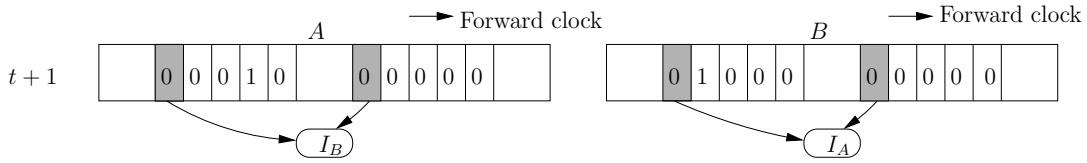

Figure 10: LILI-M1 state at time $t+1$ which has 0 pre-images

knowledge of $A(t)$ which is not possible without knowing what $I_A(t)$. If we knew what were the values of $I_A(t)$ and $I_B(t)$ from the states $A(t+1)$ and $B(t+1)$, we would be able to determine $(A(t+1), B(t+1))$'s unique pre-image. However we are not able to determine the original values of $I_A(t)$ and $I_B(t)$ from $A(t+1)$ and $B(t+1)$ as the relevant stages used in the calculation of I_B and I_A have already been clocked forward. Consequently, we have to assume each possible combination of $\{I_A(t), I_B(t)\}$ integer function outputs was equally likely. Using each possible combination $\{I_A, I_B\}$ outputs, we clock back registers A and B and check if the integer value output of $I_A(t)$ and $I_B(t)$ at time t match the number of times we clocked back registers A and B. If there are no matches, the state $(A(t+1), B(t+1))$ has no valid pre-images. If there is one matching pair, the state $(A(t+1), B(t+1))$ has one unique pre-image. If there is more than one match, there is more than one pre-image for that particular $(A(t+1), B(t+1))$ state. Figures 10 shows an example of a LILI-M1 states which has 0 pre-images. In the figures, the shaded stages represent inputs to the integer function I_A and I_B. The left shaded stage in each register are the stages A_{12} and B_{12} while the shaded stage on the right in each register represent the stages A_{20} and B_{20}.

5.2 Self-update mechanisms

Self-update operations for stream ciphers are mechanisms which use the output from a particular register to update the same register. Self-update mechanisms can be found in the initialisation functions of Sfinks [7] and the Summation generator.

5.2.1 Cause of state convergence in self-update mechanisms

In Sfinks, the self-update operation comes from the design decision of using the outputs of the Sfinks's S-box, after a delay of seven clocks, to update selected stages of Sfinks's LFSR. In Section 4.1, the cause of state convergence in Sfinks occurs was discussed. The cause was due to the fact that one of the stages used as input to the S-box to calculate the S-box's 16-bit output was later updated using one of the S-box's output bits via the XOR operation. For Sfinks, the delay of seven iterations means that during initialisation, $A_{154}(t+7)$ is updated by xoring it with the a bit from the S-box output calculated at time t (specifically, $y_{10}(t)$), of which $A_{161}(t)$ is one

$A_{12}(t)$	$B_{20}(t)$	$I_B(t)$
0	0	1
0	1	2
1	0	3
1	1	4

Table 14: Output of LILI-M2's I_A function based on its inputs.

of the S-Box inputs. Since Sfinks' register A is a regularly clocked LFSR, $A_{161}(t) = A_{154}(t+7)$ after seven clocks. By flipping the bit at $A_{161}(t)$, there is a chance that, with all other stages in A being the same, the flipped bit in $A_{161}(t)$ will result in a different value in $y_{10}(t)$. When $y_{10}(t)$ is used to update $A_{154}(t+7)$, there is a chance two distinct states will converge to the same state. The equations which have to be met for this to happen are detailed in Section 4.1. One possible way of avoiding state convergence is to change the time delay by which the stages in Sfinks' LFSR are updated by the S-Box output word. However, care must be taken when selecting the amount of delay to ensure that state convergence does not re-occur.

Unlike Sfinks, where the delay used to update a particular stage in the internal state is seven clocks, the delay used in the summation generator to update the memory bit is one clock. For the summation generator, state convergence will occur if the memory bit C of the summation generator was randomly initialised. That is, if two summation generator states had the same contents for registers A and B, and the contents of memory bit C for the two states are complements of each other, state convergence will occur. To prevent state convergence, register C of a summation generator has to be initialised to a fixed value (either 0 or 1). If the memory bit C was fixed during initialisation, the state convergence problem described above will not occur. However, it should be noted that by fixing the value of that memory bit, the state entropy will still be reduced by one bit.

5.2.2 A case study in state convergence in stream ciphers using self-update mechanisms

In this section, we present a modified version of LILI, LILI-M2 as an additional case study into how state convergence can occur in stream ciphers using self-update mechanisms.

LILI-M2 retains all the components of the LILI stream ciphers. The only modification comes from the selection of stages which are used in the clocking function I_B. In LILI-M2, one of the input stages comes from register A, while the other comes from register B. The function I_B is defined as

$$I_B(t) = 2 \times A_{12}(t) + B_{20}(t) + 1$$

The output of I_B based on its respective inputs can be seen in Table 14. Thus, in LILI-M2, register A is regularly clocked, as was the case in the original LILI stream ciphers, while the clocking of register B is determined by I_B. A diagram of LILI-M2 can be seen in Figure 11.

Determining if state convergence occurs in LILI-M2. Similar to the simulations done for LILI-M1, simulations were also performed to determine if state convergence occurs in LILI-M2.

The results of our experiments are shown in Table 15. This table shows how many pre-images were obtained for the 32 possible internal states have. It is clear from Table 15 that state convergence occurs in LILI-M2, as there are states which can have zero or two pre-images.

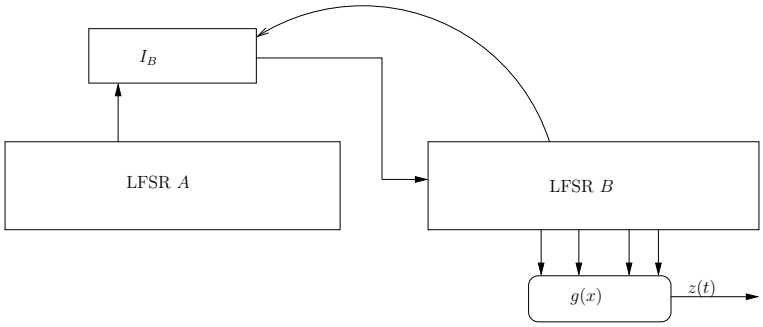

Figure 11: Structure and components of the LILI-M2

No. of pre-images	No. of states
0	8
1	16
2	8

Table 15: No. of pre-images which can be obtained for 32 states

Why state convergence occurs in LILI-M2. In LILI-M2, register A is regularly clocked and autonomous, while the clocking of register B is determined by the output of the control function $I_B(t)$. Since register A is autonomous, given the state $A(t+1)$, it is possible to determine the unique pre-image $A(t)$ and determine its bit contribution in the calculation of the $I_B(t)$. The state convergence problem arises when we try to determine $B(t+1)$'s pre-image. Since we do not know what the actual value of $B_c(t)$ was, we are not able to determine which update function to use to clock $B(t+1)$ back to $B(t)$. Therefore, we have to assume that either control bit value (0 or 1) is possible. Clocking $B(t+1)$ back assuming the control bit was 0 or 1 may yield two possible pre-images, $B(t)$ and $B'(t)$. Consequently, two unique states, ($A(t)$,$B(t)$) and ($A(t)$,$B'(t)$) which are obtained when clocking back register B assuming that $B_c(t)$ was 0 and assuming $B_c(t)$ was 1, may yield two pre-images which when clocked forwards, give $(A(t+1), B(t+1))$. Figure 12 shows an example of a LILI-M2 states which has 0 pre-images. In the figures, the shaded stages represent inputs to the integer function I_B. The shaded stage in register A is A_{12}, while the shaded stage in register B is B_{20}.

6 Conclusion

In this paper, we examined several stream ciphers known to experience state convergence and examined why this occurs. The ciphers analysed were: A5/1, Mickey, Sfinks and the summation generator.

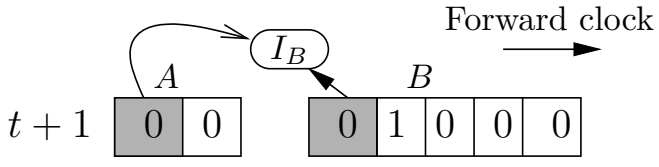

Figure 12: LILI-M2 state at time $t+1$ which has 0 pre-images

We provide estimates on the amount of distinct initial states which remain after 2–6 iterations of A5/1's state-update function and discuss why state convergence occurs in A5/1. For the Mickey stream cipher, we show that state convergence can occur during its initialisation phase. We also show that the effective state size of Mickey-v2 drops below 160 bits, contrary to the claims of the designers. The analysis of Sfinks' state-update function used during initialisation indicates that state convergence occurs and we show that after 120 iterations of the state-update function, the number of distinct initial states remaining is $2^{158.55}$. We demonstrated why state convergence occurs in the summation generator, resulting in the lost of one bit of entropy.

From the analyses of the above-mentioned ciphers, we identified two mechanisms which can cause state convergence. These mechanisms are: mutual clock-control and self-update operations. A5/1 and Mickey can be grouped into the mutual clock-control category, while Sfinks and the summation generator can be grouped into the self-update category.

When state convergence occurs during initialisation of stream ciphers, the number of distinct initial states will decrease as the number of iterations of the initialisation state update function performed increases. This decreases the effective key-IV space of the keystream generator while simultaneously decreasing the efficiency of the rekeying process with no corresponding increase in security. If the state-update function used during initialisation is also used to generate keystream, state convergence problems will continue and can result in a further reduction of the effective key-IV space.

Avoiding state convergence requires careful analysis of the state-update functions used during both initialisation and keystream generation to ensure that they are one-to-one. These one-to-one state-update functions should also be carefully used. The state convergence experienced by Sfinks is an example that demonstrates that the use of one-to-one components in composing a state update function is not enough to ensure that state convergence does not occur, and that designers should ensure that the overall state update function is also one-to-one. Through the use of two case studies, we also demonstrated how simple modifications to the state-update function used in the LILI family of stream ciphers can cause state convergence when the original versions do not suffer from this problem.

Current stream cipher proposals commonly include an analysis section detailing their claimed resistance to various attacks. We recommend that future stream cipher designers pay careful attention to the choice of state-update functions used during initialisation and keystream generation, as functions which are not one-to-one may make many attacks more effective than anticipated.

References

[1] Alhamdan, A., Bartlett, H., Simpson, L., Dawson, E., Wong, K.K.H.: State convergence in the initialisation of the Sfinks stream cipher. In: Pieprzyk, J., Thomborson, C. (eds.) Australasian Information Security Conference (AISC 2012). vol. 125, pp. 27–31. Australian Computer Society (2012)

[2] Arnault, F., Berger, T.P.: F-FCSR: design of a new class of stream ciphers. In: Gilbert, H., Handschuh, H. (eds.) Fast Software Encryption (FSE 2005). Lecture Notes in Computer Science, vol. 3557, pp. 83–97. Springer (2005)

[3] Babbage, S., Dodd, M.: The stream cipher MICKEY (version 1). eSTREAM, ECRYPT Stream Cipher Project, Report 2005/015 (2005), available from `http://www.ecrypt.eu.org/stream/ciphers/mickey/mickey.pdf`

[4] Babbage, S., Dodd, M.: The stream cipher MICKEY 2.0 (2006), available from `http://www.ecrypt.eu.org/stream/p3ciphers/mickey/mickey_p3.pdf`

[5] Biham, E., Dunkelman, O.: Differential Cryptanalysis in Stream Ciphers. Cryptology ePrint Archive, Report 2007/218 (June 2007), available from `http://eprint.iacr.org/2007/218.pdf`

[6] Biryukov, A., Shamir, A., Wagner, D.: Real Time Cryptanalysis of A5/1 on a PC. In: Schneier, B. (ed.) Fast Software Encryption (FSE 2000). Lecture Notes in Computer Science, vol. 1978, pp. 1–18. Springer (2001)

[7] Braeken, A., Lano, J., Mentens, N., Preneel, B., Verbauwhede, I.: SFINKS : A Synchronous Stream Cipher for Restricted Hardware Environments. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/026 (2005), available from `http://www.ecrypt.eu.org/stream/ciphers/sfinks/sfinks.ps`

[8] Briceno, M., Goldberg, I., Wagner, D.: A pedagogical implementation of A5/1 (1999), available from `http://cryptome.org/jya/a51-pi.htm`

[9] Canniére, C.D., Preneel, B.: Trivium. In: Robshaw, M.J.B., Billet, O. (eds.) New Stream Cipher Designs: The eSTREAM Finalists. Lecture Notes in Computer Science, vol. 4986, pp. 244–266. Springer (2008)

[10] 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.) Information Security and Privacy (ACISP 2002). Lecture Notes in Computer Science, vol. 2384, pp. 25–39. Springer (2002)

[11] European Network of Excellence for Cryptology: The eSTREAM Project, Available from `http://www.ecrypt.eu.org/stream/index.html`

[12] Golić, J.: Cryptanalysis of Three Mutually Clock-Controlled Stop/Go Shift Registers. IEEE Transactions on Information Theory 46(3), 1081–1090 (2002)

[13] Gollmann, D., Chambers, W.G.: Clock-Controlled Shift Registers: A Review. IEEE Journal on Selected Areas in Communications 7(4), 525–533 (1989)

[14] Hong, J., Kim, W.H.: TMD-Tradeoff and State Entropy Loss Considerations of Streamcipher MICKEY. In: Maitra, S., Madhavan, C.E.V., Venkatesan, R. (eds.) INDOCRYPT 2005. Lecture Notes in Computer Science, vol. 3797, pp. 169–182. Springer (2005)

[15] Jansen, C.J.: Stream Cipher Design: Make your LFSRs jump! Presented at SASC 2004 (2004), available from `http://www.ecrypt.eu.org/stvl/sasc/`

[16] Jaulmes, É., Muller, F.: Cryptanalysis of the F-FCSR Stream Cipher Family. In: Bart Preneel and Stafford E. Tavares (ed.) Selected Areas in Cryptography (SAC 2005). Lecture Notes in Computer Science, vol. 3897, pp. 20–35. Springer (2006)

[17] Kiselev, S.A., Tokareva, N.N.: Reduction of the Key Space of the Cipher A5/1 and Invertibility of the Next-State Function for a Stream Generator. Journal of Applied and Industrial Mathematics 6(2), 194–202 (2012)

[18] Röck, A.: Entropy of the Internal State of an FCSR in Galois Representation. In: Nyberg, K. (ed.) Fast Software Encryption (FSE 2005). Lecture Notes in Computer Science, vol. 5086, pp. 343–362. Springer (2008)

[19] Rueppel, R.A.: Correlation Immunity and the Summation Generator. In: Williams, H.C. (ed.) Advances in Cryptology — CRYPTO '85. Lecture Notes in Computer Science, vol. 218, pp. 260–272. Springer (1985)

[20] Simpson, L., Dawson, E., Golić, J.D., Millan, W.: LILI Keystream Generator. In: Stinson, D.R., Tavares, S. (eds.) SAC - Selected Areas in Cryptograpy (SAC 2000). Lecture Notes in Computer Science, vol. 2012, pp. 248–261. Springer (2000)

[21] Teo, S.G., Al-Hamdan, A., Bartlett, H., Simpson, L., Wong, K.K.H., Dawson, E.: State Convergence in the Initialisation of Stream Ciphers. In: Parampalli, U., Hawkes, P. (eds.) Information Security and Privacy (ACISP 2011). Lecture Notes in Computer Science, vol. 6812, pp. 75–88. Springer (2011)