State convergence in the initialisation of the Sfinks stream cipher

Ali Alhamdan[1] **Harry Bartlett**[1,2] **Leonie Simpson**[1,2] **Ed Dawson**[1]
Kenneth Koon-Ho Wong[1]

[1] Information Security Institute
Queensland University of Technology,
126 Margaret Street, Brisbane Qld 4001, Australia,
Email: `a.alhamdan@student.qut.edu.au`, `{e.dawson, kk.wong}@qut.edu.au`

[2] Faculty of Science and Technology,
Queensland University of Technology,
GPO Box 2434, Brisbane Qld 4001, Australia
Email: `{h.bartlett, lr.simpson}@qut.edu.au`

Abstract

Sfinks is a shift register based stream cipher designed for hardware implementation. The initialisation state update function is different from the state update function used for keystream generation. We demonstrate state convergence during the initialisation process, even though the individual components used in the initialisation are one-to-one. However, the combination of these components is not one-to-one.

keywords: stream cipher, initialisation process, state convergence, Sfinks

1 Introduction

The Sfinks stream cipher was submitted to eSTREAM, the ECRYPT call for stream cipher proposals in April 2005, by Braeken, Lano, Mentens, Preneel and Verbauwhede (Braeken et al. 2005). It is a bit-based stream cipher that takes an 80-bit secret key and 80-bit IV as inputs and has a 256-bit internal state. Sfinks is categorized as PROFILE 2A, suitable for hardware applications and with an associated authentication method.

The Sfinks stream cipher was attacked by Courtois (2005) using basic and fast algebraic attacks. These algebraic attacks exploit the state update function used during keystream generation, but do not make use of the initialisation process. Courtois found that Sfinks can be broken with complexity of 2^{71} computations using 2^{43} keystream bits, which is faster than the claimed security level of 2^{80}.

As noted above, the Sfinks stream cipher is broken as a keystream generator. The purpose of this paper is to investigate the strategy used for initialisation. We note that the state update functions are different during initialisation and keystream generation. Using a modified version of the keystream generation state update function during initialisation may produce some security benefits. However, for Sfinks, we find that although the state update function during keystream generation is one-to-one, this is not the case during initialisation. We investigate the resulting state convergence in this paper. This paper considers specifically the properties of the initialisation process rather than the keystream generation of Sfinks stream cipher.

State convergence occurs when two or more states at time t are mapped to the same state after $t+\alpha$ iterations, for some $\alpha > 0$ and the states do not diverge after this point. That is state convergence occurs when the state update function is not one-to-one. State convergence may occur during the initialisation process or keystream generation. This may reduce the effective key-IV size and leave the stream cipher vulnerable to attacks such as distinguishing attacks (Rose & Hawkes 2002) or time-memory-data tradeoff attacks (Biryukov & Shamir 2000).

This paper is organized as follows. Section 2 presents a brief description of the Sfinks keystream generator, including details of the initialisation process. In Section 3, an analysis of the Sfinks initialisation process is presented. Section 4 discusses the results and concludes the work.

2 Description of Sfinks

The Sfinks stream cipher (Braeken et al. 2005) has two main components: a shift register, S, and a nonlinear one-to-one inversion function INV as shown in Figure 1. Let s_t^i denote the contents of register stage i at time t, where $i = 0, 1, \dots 255$ and $t \geq -128$. Sfinks uses an 80-bit secret key $K = k_{79}, \dots, k_0$ and 80-bit initial value IV$= v_{79}, \dots, v_0$.

During keystream generation, the 256-bit shift register is regularly clocked. The linear feedback function is described as following.

$$\begin{aligned} s_{t+1}^{255} = {} & s_t^{212} \oplus s_t^{194} \oplus s_t^{192} \oplus s_t^{187} \oplus s_t^{163} \\ & \oplus s_t^{151} \oplus s_t^{125} \oplus s_t^{115} \oplus s_t^{107} \oplus s_t^{85} \\ & \oplus s_t^{66} \oplus s_t^{64} \oplus s_t^{52} \oplus s_t^{48} \oplus s_t^{14} \oplus s_t \end{aligned} \quad (1)$$

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, but in different ways in each case. Let x and y denote the 16-bit input and output of INV respectively, where $x = (x^{16}, \dots, x^1)$ and $y = (y^{15}, \dots, y^0)$. INV is an invertible function, $\mathbb{F}_2^{16} \to \mathbb{F}_2^{16}$ that calculates the inverse of the 16-bit input, modulo the primitive polynomial $X^{16} + X^5 + X^3 + X^2 + 1$. The 16 input bits are taken from 16 register stages as follows.

$$(x^{16},\dots,x^{1}) = (s^{255}, s^{244}, s^{227}, s^{193}, s^{161}, s^{134}, s^{105}, s^{98}, s^{74}, s^{58}, s^{44}, s^{21}, s^{19}, s^{9}, s^{6}, s^{1}) \quad (2)$$

The 16-bit output of the S-box, y, is treated as 16 bit values $(y^{15},\dots,y^{0})$. During the initialisation, all of the 16-bit output of INV is fed back to specified stages of the shift register. During keystream generation, only one bit of the output of the INV contributes to the formation of the keystream bit.

2.1 Initialisation process

The initialisation process takes as input the 80-bit key and 80-bit IV and performs 128 iterations (starting at $t = -128$) to produce the 256-bit initial register state. Once this initial state is obtained, keystream generation can begin. The initialisation process is performed in two phases, which we refer to as *loading* and *diffusion*.

2.1.1 Loading phase

In the loading phase, firstly, all of the register stages are set to zero. Then the 80-bit secret key and 80-bit IV are transferred to specified positions in the shift register. The secret key is loaded into the state such that $s^{96+i}_{-128} = k_i$, for $0 \le i \le 79$, and the IV is loaded into the state such that $s^{176+i}_{-128} = v_i$, for $0 \le i \le 79$. The register stage $s^{95}_{-128} = 1$ and the remaining $s^{i}_{-128} = 0$, for $0 \le i \le 94$.

The output of the S-box is set to all-zero for the first seven 16-bit outputs, $y_{-134+t} = (0,\dots,0)$ for $0 \le t \le 6$. In (Braeken et al. 2005) the process is described as necessary to clear the pipeline stages in the hardware and to provide the initial values of the output of S-box to allow for the delay of 7 steps.

When both the secret key and IV have been transferred and the rest of state bits are fixed to the designated values, the Sfinks stream cipher is in its *loaded state*. Following this, the diffusion phase begins.

2.1.2 Diffusion phase

The diffusion phase consists of 128 iterations of the initialisation state-update function. Each iteration can be considered as a function which maps the state space to itself. After the diffusion phase is completed, the keystream generator is said to be in its *initial state*. Figure 2 gives a general overview of state update function during the diffusion phase of the Sfinks stream cipher.

For the first seven iterations, the initialisation processes are performed in a purely linear manner (with the effect of linear feedback of shift register only), as the output bits of the nonlinear function are zeros, $(y_{-134},\dots,y_{-128}) = (0,\dots,0)$.

For the remaining iterations, the initialisation process is performed using the linear feedback of the shift register and the output of the nonlinear S-box function. The S-box output feeds back into 16 specified stages of the shift register with a time delay of 7 steps as detailed below.

$$s^{i}_{t} = s^{i+1}_{t-1} \oplus y^{i \bmod 16}_{t-7} \quad (3)$$

for $i = \{11, 17, 41, 52, 66, 80, 111, 118, 142, 154, 173, 179, 204, 213, 232, 247\}$. All other bits are clocked normally, i.e. $s^{i}_{t} = s^{i+1}_{t-1}$ for all other i and the shift register feedback function (Equation 1) still applies. At each iteration, the shift register is clocked and then the INV function is called to calculate the inverse of the 16-bit input to the S-box. This is stored as the S-box output. The 16-bit output of the S-box is XORed with the contents of 16 specified stages of the shift register to form the contents of another 16 stages of the shift register. The S-box function is the only nonlinear component in the initialisation process. A complete description of the state update function is:

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

At $t = 0$, the Sfinks stream cipher has completed the initialisation processes and is ready for keystream generation. During keystream generation, the register feedback is linear. The least significant bit of the 16-bit output value of the S-box is XORed with the value of stage s^0 to produce each keystream bit, with a delay of 7 steps applied to both values. That is, $z_t = s^{0}_{t-7} \oplus y^{0}_{t-7}$.

3 Analysis of initialisation processes

Analysis of the Sfinks stream cipher initialisation process is complicated by the delay of 7 steps in feeding the S-box output back into the register. However we observe that the correspondence between this delay of seven steps and the difference between certain input and output taps leads to state convergence as shown below. In the remainder of this paper, we refer to stages of S which provide inputs to the S-box as input stages and stages of S which receive outputs from the S-box as output stages respectively.

From Figure 2, note that the distance between some input and output stages is equal to the delay time. Specifically, there is one case where s^{i}_{t} is an output stage and s^{i+7}_{t-7} is an input stage. In this case, recall from Equation 3 that $s^{i}_{t} = s^{i+1}_{t-1} \oplus y^{i \bmod 16}_{t-7}$, and note that if we complement both s^{i+1}_{t-1} and $y^{i \bmod 16}_{t-7}$ then the same value of s^{i}_{t} will be obtained. However, $s^{i+1}_{t-1} = s^{i+7}_{t-7}$ under regular clocking, and the S-box output $y^{i \bmod 16}_{t-7}$ depends on the contents of the input stage s^{i+7}_{t-7}. That is,

$$s^{i}_{t} = s^{i+7}_{t-7} \oplus y^{i \bmod 16}_{t-7} = \bar{s}^{i+7}_{t-7} \oplus \bar{y}^{i \bmod 16}_{t-7} \quad (4)$$

where $\bar{s}$ and $\bar{y}$ represent the complements of s and y respectively.

It is possible that complementing the contents of the input stage s^{i+7}_{t-7} may cause the required change in the S-box output bit $y^{i \bmod 16}_{t-7}$. This situation provides the basis of a search for states which converge.

3.1 States which converge

An examination of the Sfinks register shows there is only one input stage with a distance to the next output stage equal to 7 steps. That input stage is s^{161}

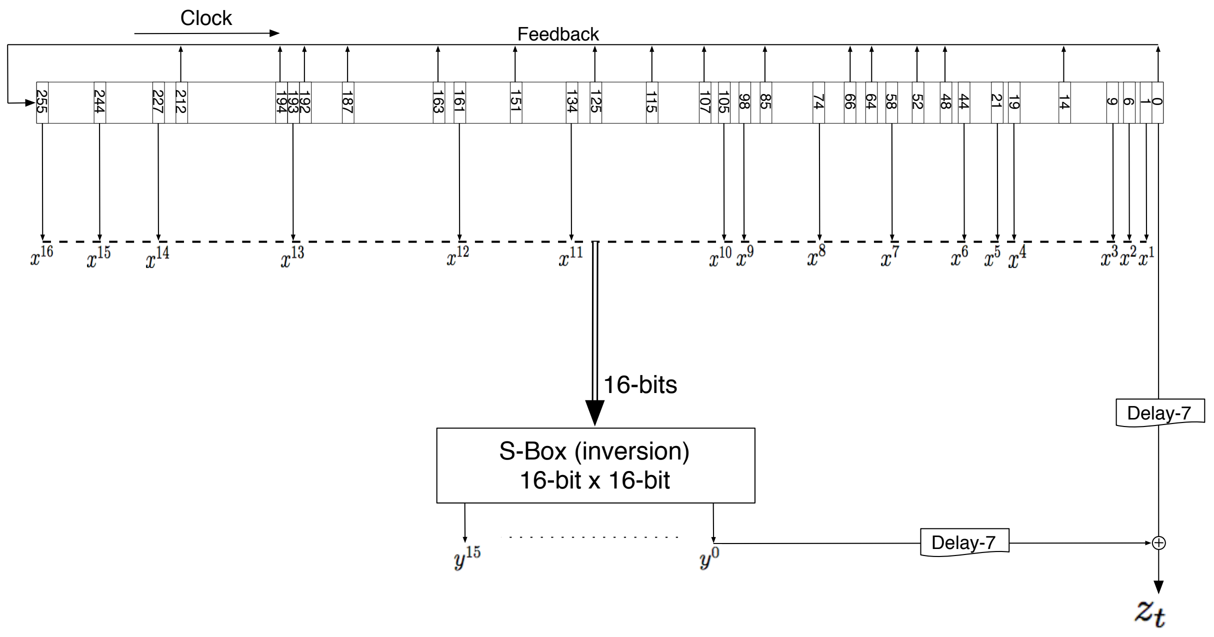

Figure 1: Keystream generator of Sfinks stream cipher

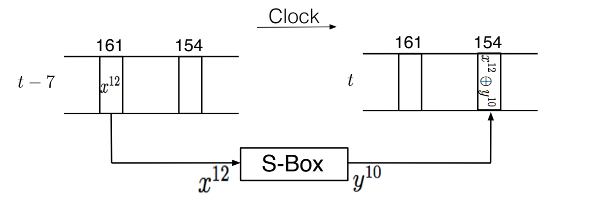

Figure 3: Input and Output stages have 7 steps delay

and the output stage is s^{154}. The contents of s^{161} correspond to the S-box input x^{12}, and the S-box output y^{10} is fed back to s^{154}. Specifically, $s^{161}_{t-7} = x^{12}_{t-7}$ and $s^{154}_t = s^{155}_{t-1} \oplus y^{10}_{t-7}$. According to Equation 4, the value of s^{154}_t will not be changed if complementing the input bit $s^{161}_{t-7} = x^{12}_{t-7}$ results in the output bit y^{10}_{t-7} being complemented as well. Therefore, we look for pairs of S-box inputs (x^{16}, …, x^1) which differ only in bit x^{12} and for which the corresponding pair of outputs (y^{15}, …, y^0) differ in bit y^{10}.

Consider firstly the input pair for which the output pair differ only in y^{10}, as illustrated in Figure 3. Such a pair of S-box inputs (and corresponding output) exists, and is presented in Table 1. For emphasis, x^{12} is underlined and bold font, as is y^{10}. Table 2 gives an example of two 256-bit register states S^A_{t-7} and S^B_{t-7} which converge to the same state after 7 iterations. For efficient presentation the hex representation of the 256-bit binary state is given. Note that all register stages except s^{161} are the same. S^A_{t-7} and S^B_{t-7} both converge to S_t.

Since the register S is 256 bits long, and there are 16 stages used as input to the S-box, if we fix the contents of these 16 stages to the pattern given in Table 1, we are free to choose any values for the remaining 240 stages. Therefore, there are 2^{240} pairs of states which converge after 7 iterations. One such pair is presented in Table 2.

For the S-box pair in Table 1 discussed in the illustration above the inputs differed only in position x^{12} and the outputs differed only in position y^{10}. In general, however, it is not necessary to apply such a strict condition on the output bits. Referring to Equation 3, and considering 6 consecutive steps of regular clocking, we have $s^i_t = s^{i+1}_{t-1} \oplus y^{i \bmod 16}_{t-7} = s^{i+7}_{t-7} \oplus y^{i \bmod 16}_{t-7}$ for any output bit. If complementing the input bit s^{161}_{t-7} (which is x^{12}_{t-7}) causes $y^{i \bmod 16}_{t-7}$ to be changed, it may be possible to complement s^{i+7}_{t-7} to obtain a second state that gives the same value for s^i_t. Recall that 16 stages of the shift register S receive the output of the S-box at each iteration of the state update function. Of these 16 output stages, there are only six (s^{11}_{t-7}, s^{17}_{t-7}, s^{41}_{t-7}, s^{52}_{t-7}, s^{80}_{t-7}, s^{111}_{t-7}) which directly or indirectly affect an input stage during the six consecutive clocks. For example, the output y^9 of the S-box is fed back to s^{41} and there is an input to the S-box within the delay time at stage s^{44}. Allowing this bit to change, may result in divergence in later steps. Note also that if an input bit of the shift register feedback is complemented at time $t-7$, we also need the stage s^0_{t-7} to be complemented to assure that the new bit s^{255}_{t-6} will not be changed. Therefore,

Table 1: A special S-box pair which differ in x^{12} and y^{10} only

	Input																Output															
S-box sequence	x^{16}	…	…	…	…	…	…	…	…	…	…	…	…	…	…	x^1	y^{15}	…	…	…	…	…	…	…	…	…	…	…	…	…	…	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	1	1	0	1	**0**	0	0	0	1	1	1	0	0	0	1	0	0	1	1	0	1	**1**	0	0	0	1	1	1	0	0	0	1
2nd value	1	1	0	1	**1**	0	0	0	1	1	1	0	0	0	1	0	0	1	1	0	1	**0**	0	0	0	1	1	1	0	0	0	1

Table 2: Two states (hex) differing only in stage s^{161} which converge

S^A_{t-7}	F19B7E15AF4FF1338DDF080**0**AD8C56A42913E4B90CBEEFD3A4075AFD3351E5C1
S^B_{t-7}	F19B7E15AF4FF1338DDF080**2**AD8C56A42913E4B90CBEEFD3A4075AFD3351E5C1
S_t	BBE336FC2B7E9FE2671B9E10055B58AD481227C972187DDFA7580EB5FA66ABCB

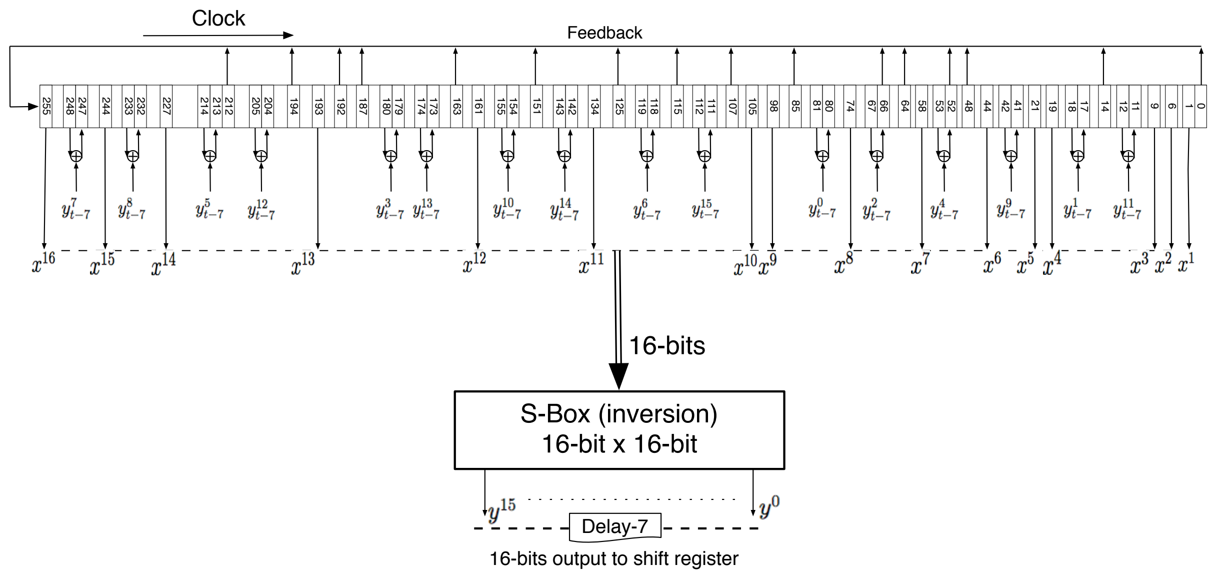
Clock
Feedback
255 248 247 244 233 232 227 214 213 212 205 204 194 193 192 187 180 179 174 173 163 161 155 154 151 143 142 134 125 119 118 115 112 111 107 105 98 85 81 80 74 67 66 64 58 53 52 48 44 42 41 21 19 18 17 14 12 11 9 6 1 0
y^7_{t-7} y^8_{t-7} y^5_{t-7} y^{12}_{t-7} y^3_{t-7} y^{13}_{t-7} y^{10}_{t-7} y^{14}_{t-7} y^6_{t-7} y^{15}_{t-7} y^0_{t-7} y^2_{t-7} y^4_{t-7} y^9_{t-7} y^1_{t-7} y^{11}_{t-7}
x^{16} x^{15} x^{14} x^{13} x^{12} x^{11} x^{10} x^9 x^8 x^7 x^6 x^5 x^4 x^3 x^2 x^1
16-bits
S-Box (inversion)
16-bit x 16-bit
y^{15}
y^0
Delay-7
16-bits output to shift register

Figure 2: Initialisation processes of Sfinks stream cipher

when considering pairs of inputs and outputs of the S-box for which convergence occurs the values of the six S-box outputs (y^0, y^1, y^4, y^9, y^{11} and y^{15}) must be fixed. Therefore, we look for pairs of the S-box inputs which differ only in x^{12} and for which the output bits differ in y^{10} and possibly in any bit of y^2, y^3, y^5, y^6, y^7, y^8, y^{10}, y^{12}, y^{13} and y^{14}.

We used an exhaustive computer search to look for pairs of S-box inputs that satisfy the conditions described above, and found 273 pairs of the S-box inputs (and corresponding outputs) with such patterns. Table 3 gives three different examples of these S-box input and output pairs. Note that the only difference in each input pair is the underlined bold bit x^{12}. In the output pair, the underlined bold bits are the bits which differ in each pair (which must include y^{10}) and the bits y^0, y^1, y^4, y^9, y^{11} and y^{15} (shown in italics) should be the same in each pair. For each pair in Table 3, Table 4 provides an example of two states based on that pattern that converge to the same state after 7 iterations.

Table 3: Examples of complying pairs of S-box inputs and outputs

		Input	Output
	S-box sequence	x^{16} . . . . x^1	y^{15} . . . . 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 pair	1st value	0 0 0 0 **0** 0 0 0 1 0 1 0 0 1 1 1	*0* **1** 0 0 *0* **1** *0* **0** **1** **0** **0** *1* 0 0 *1* *0*
	2nd value	0 0 0 0 **1** 0 0 0 1 0 1 0 0 1 1 1	*0* **0** 0 0 *0* **0** *0* **1** **0** **1** **1** *1* 0 0 *1* *0*
2nd pair	1st value	0 0 0 0 **0** 0 0 0 1 1 1 0 1 1 1 0	*0* *0* **1** 0 *0* **1** *1* 1 1 1 0 *0* 1 1 *1* *1*
	2nd value	0 0 0 0 **1** 0 0 0 1 1 1 0 1 1 1 0	*0* *0* **0** 0 *0* **0** *1* 1 1 1 0 *0* 1 1 *1* *1*
3rd pair	1st value	0 0 0 0 **0** 0 0 1 1 0 0 1 1 0 1 1	*1* **1** **1** **0** *1* **1** *0* **0** **1** 1 1 *1* 0 0 *0* *1*
	2nd value	0 0 0 0 **1** 0 0 1 1 0 0 1 1 0 1 1	*1* **0** **0** **1** *1* **0** *0* **1** **0** 1 1 *1* 0 0 *0* *1*

From above, there are 273 pairs of 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 $\frac{273}{2^{15}} = 2^{-6.9}$.

Table 4: Three examples; each two states which converge to the same state

1st pair	S^{A1}_{t-7}	**3**18B**7**E15**A**F4FF1318DDF080**0**AD**A**C56A4**2**913E4B90CBEEFD3A0075AFD3351E7C**3**
	S^{A2}_{t-7}	**7**18B**F**E15**B**F4FF1318DDF080**2**AD**8**C56A4**0**913E4B90CBEEFD3A0075AFD3351E7C**2**
	S_t	BAE316FC2B5E9FE2631BBE10055B18AD485227C972197DDFA7500EB5FA64A3CF
2nd pair	S^{B1}_{t-7}	718B7E15AF4FF1318D**C**F080**0**AD8C56A42913E4B90CBEEFD3A4075AFD3359E7C1
	S^{B2}_{t-7}	718B7E15AF4FF1318D**D**F080**2**AD8C56A42913E4B90CBEEFD3A4075AFD3359E7C1
	S_t	7E6317FC2B5E9FE26313BE10055B18AD481227C972187DDBA7480CB5FA64B3CF
3rd pair	S^{C1}_{t-7}	**3**18B**7**E15AF4**F**F1318D**C**F080**0**AD**A**C56A42913E4BD0CBEEFD3A0074AFD3379E5C3
	S^{C2}_{t-7}	**7**18B**F**E15AF4**7**F1318D**D**F080**2**AD**8**C56A42913E4BD0CBEEFD3A0074AFD3379E5C3
	S_t	FAE316FC2B7E9FE2631BBE10055B18AD4812A7C97A187DDFA7500E95FA66FBCB

3.2 State convergence across the initialisation process

Recall from Section 2.1.1, that the loaded state of Sfinks has a defined format, with $s^{95}_{-128} = 1$ and $s^{i}_{-128} = 0$ for $i = 0, \ldots, 94$. This slightly reduces the occurrence of state convergence for the first few iterations of the diffusion process but does not prevent it altogether.

According to the reference implementation of Sfinks (Lano 2005), the S-box first receives live inputs from the input stages at $t = -127$ (after the first clock of the shift register). Convergence can not occur until 7 steps after this, at $t = -120$.

Based on the general case discussed above, however, state convergence can occur immediately after these 7 iterations. All that is required to impose the additional condition that y^2 also remain unchanged when x^{12} is complemented. Table 5 shows a pair of input and output bits of the S-box that can occur after the first iteration and converge after 7 iterations. This pair satisfies the condition discussed above, and therefore will lead to convergence after 7 steps at $t = -120$. A pair of converging states based on this pattern is presented as an example in Table 6.

Thus, we see that state convergence can occur throughout the diffusion process of Sfinks cipher. There are 120 iterations that may carry a state convergence during the initialisation. Based on the probability of convergence detemined above, an approximate estimate for the proportion of distinct states re-

Table 5: A pair of S-box inputs and outputs that can occur at $t = -120$

	Input																Output															
S-box sequence	x^{16}	…														x^1	y^{15}	…														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	0	1	0	0	**0**	0	0	0	0	0	0	0	0	0	0	0	*0*	**1**	**0**	0	*0*	**0**	*1*	0	**1**	**1**	1	*1*	**1**	1	*1*	*1*
2nd value	0	1	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	*0*	**0**	**1**	0	*0*	**1**	*1*	0	**0**	**0**	1	*1*	**0**	1	*1*	*1*

Table 6: Two states (hex) which converge to the same state at $t = -120$

S^A_{-127}	**3**19B7E15AF4FF1318**9D**F080**0**AD**A**C56A4**0**913E4B90CBEEBD3A0074AFD3351E58**0**
S^B_{-127}	**7**19B7E15AF4FF1318**DC**F080**2**AD**8**C56A4**2**913E4B90CBEEBD3A0074AFD3351E58**1**
S_{-120}	3EE336FC2B7E9FE2631BBE10015B18AD485227C972187DD3A7500C95FA64A3CB

maining after 128 iterations is $(1-2^{-6.9})^{120} = 0.9963$. Thus, the number of reachable distinct states is approximately $(1-2^{-6.9})^{120} \times 2^{160} = 2^{158.55}$. This can be regarded as an approximate upper bound as there may be other mechanisms of convergence in addition to these we have identified.

4 Discussion and conclusion

This paper demonstrates that state convergence occurs during the initialisation phase of the Sfinks cipher. This is due to a combination of the delay in feeding the S-box output back to the shift register and the shift register tap spacing. Specifically, this is caused by the correspondence between the 7-clock feedback delay and the distance of 7 steps between certain input and output stages. The combination of this with the specific S-box used here results in an initialisation state update function that is not one-to-one, thereby resulting in the state convergence.

The relationship between the input/output stages and the delay time as shown by Equation 3 is 7 steps for the Sfinks stream cipher. However, if the delay in Equation 3 is changed to another number and the spacing between any input and output stages is equal to this delay number, then the state convergence can still occur. The approach described in this paper would be applied to detect such convergence with the focus on a different input to the S-box.

The delayed S-box feedback in Sfinks adds complexity to the analysis of the initialisation process for this cipher, at the expense of additional memory and a slight delay in producing keystream. However, it is questionable whether the increased complexity, (which reduces efficiency) has resulted in a corresponding increase in security. In fact, it may have contributed to a reduction of security, as the combination of this feedback delay and the shift register tap spacing result in the occurrence of state convergence in this cipher. It is actually possible to avoid the state convergence in this design by introducing minor changes to tap locations, even so, it is not clear that the delay itself is necessary for the security of the cipher.

When considering the design of the state update function used during initialisation, it is worth noting that the INV function (represented as the S-box) and the shift register feedback function used for keystream generation (Equation 1) are individually one-to-one. However, the combination of these functions which occurs during initialisation is not one-to-one. This demonstrates that designers should be careful when combining components to ensure that the combination does not have undesirable properties.

Sfinks is designed for use with an 80-bit key and 80-bit IV. As the state size (256 bits) is greater than the sum of the key and IV size, it seems reasonable to assume that 2^{160} different keystream could be produced. However, the state converges problem demonstrated in this paper reduces the number of distinct keystream. We estimate the number of distinct keystream is less than $2^{158.55}$. Although the impact of this convergence on the security of Sfinks is minor, it can be avoided entirely by more careful design.

The weakness in the initialisation process of Sfinks may occur in other ciphers which use a modified version of the keystream generation state update function as the initialisation method. Even slight modifications may change the properties of the state update function. To avoid state convergence, the state update function must be one-to-one. Even where individual components of the state update function are one-to-one as for Sfinks, it is important to check that the combination is one-to-one as well, to avoid state convergence.

References

Biryukov, A. & Shamir, A. (2000), Cryptanalytic time/memory/data tradeoffs for stream ciphers, *in* T. Okamoto, ed., 'ASIACRYPT', Vol. 1976 of *Lecture Notes in Computer Science*, Springer, pp. 1–13.

Braeken, A., Lano, J., Mentens, N., Preneel, B. & Verbauwhede, I. (2005), 'SFINKS: A synchronous stream cipher for restricted hardware environments', eSTREAM, ECRYPT Stream Cipher Project, Report 2005/026. `http://www.ecrypt.eu.org/stream`.

Courtois, N. (2005), Cryptanalysis of Sfinks, *in* D. Won & S. Kim, eds, 'ICISC', Vol. 3935 of *Lecture Notes in Computer Science*, Springer, pp. 261–269.

Lano, J. (2005), 'Sfinks stream cipher source code', eSTREAM, ECRYPT Stream Cipher Project. `http://www.ecrypt.eu.org/stream/sfinks.html`.

Rose, G. & Hawkes, P. (2002), On the applicability of distinguishing attacks against stream ciphers, *in* 'Proceedings of the 3rd NESSIE Workshop', Citeseer, p. 6.