Non-randomness in eSTREAM Candidates Salsa20 and TSC-4

Simon Fischer[1], Willi Meier[1],
Côme Berbain[2], Jean-François Biasse[2], and M.J.B. Robshaw[2]

[1] FHNW, 5210 Windisch, Switzerland
{simon.fischer,willi.meier}@fhnw.ch
[2] FTRD, 38–40 rue du Général Leclerc, 92794 Issy les Moulineaux, France
{come.berbain,jeanfrancois.biasse,matt.robshaw}@orange-ft.com

Abstract. Stream cipher initialisation should ensure that the initial state or keystream is not detectably related to the key and initialisation vector. In this paper we analyse the key/IV setup of the eSTREAM Phase 2 candidates Salsa20 and TSC-4. In the case of Salsa20 we demonstrate a key recovery attack on six rounds and observe non-randomness after seven. For TSC-4, non-randomness over the full eight-round initialisation phase is detected, but would also persist for more rounds.

Key words: Stream Cipher, eSTREAM, Salsa20, TSC-4, Chosen IV Attack

1 Introduction

Many synchronous stream ciphers use two inputs for keystream generation; a secret key K and a non-secret initialisation vector IV. The IV allows different keystreams to be derived from a single secret key and facilitates resynchronization. In the general model of a synchronous stream cipher there are three functions. During initialisation a function F maps the input pair (K, IV) to a secret initial state X. The state of the cipher then evolves at time t under the action of a function f that updates the state X according to $X^{t+1} = \mathsf{f}(X^t)$. Keystream is generated using an output function g to give a block of keystream $z^t = \mathsf{g}(X^t)$. While TSC-4 follows this model, Salsa20 has no state update function f and g involves reading out the state X. Instead, we view the IV to Salsa20 as being the combination of a 64-bit *nonce* and a 64-bit *counter* and keystream is generated by repeatedly computing $\mathsf{F}(K, IV)$ for an incremented counter.

In the analysis of keystream generators (*i.e.* in the analysis of f and g) it is typical to assume that the initial state X is random. Hence for a stream cipher we require that F has suitable randomness properties, and in particular, that it has good diffusion with regards to both IV and K. (Clearly this applies equally to the case when the output of F is the keystream.) Indeed, if diffusion of the IV is not complete then there may well be statistical or algebraic dependences in the keystreams for different IV's, as chosen-IV attacks on numerous stream ciphers demonstrate (*e.g.*, [6, 8, 9]). Good mixing of the secret key is similarly required

© Springer-Verlag Berlin Heidelberg 2006

and there should not be any identifiable subsets of keys that have a traceable influence on the initial state (or on the generated keystream), see [7]. Since, in many cases, F is constructed from the repeated application of a relatively simple function over r rounds, determining the required number of rounds r can be difficult. For a well-designed scheme, we would expect the security of the mechanism to increase with r, though there is a clear cost in reduced efficiency.

In this paper we investigate the initialisation of Salsa20 and TSC-4. We consider a set of well-chosen inputs (K, IV) and compute the outputs $\mathsf{F}(K, IV)$. Under an appropriate measure we aim to detect non-random behaviour in the output. Throughout we assume that the IV's can be chosen and that most, or all, of the key bits are unknown. The paper is organized as follows. The specification of Salsa20 is recalled in Section 2 with an analysis up to seven rounds in Section 3. TSC-4 is described in Section 4 with analysis in Section 5. We draw our conclusions in Section 6. For notation we use $+$ for addition modulo 2^{32}, $\oplus$ for bitwise `XOR`, $\wedge$ for bitwise `AND`, $\lll$ for bitwise left-rotation, and $\ll$ for bitwise left-shift. The most (least) significant bit will be denoted msb (lsb).

2 Description of Salsa20

A full description of Salsa20 can be found in [1]. As mentioned in the introduction, we view the initialisation vector as $IV = (v_0, v_1, i_0, i_1)$ where (v_0, v_1) denotes the nonce and (i_0, i_1) the counter. Throughout we consider the 256-bit key version of Salsa20 and we denote the key by $K = (k_0, \ldots, k_7)$ and four constants by $c_0, \ldots, c_3$ (see [1]). We denote the cipher state by $X = (x_0, \ldots, x_{15})$ where each x_i is a 32-bit word.

$$X = \begin{pmatrix} x_0 & x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 & x_7 \\ x_8 & x_9 & x_{10} & x_{11} \\ x_{12} & x_{13} & x_{14} & x_{15} \end{pmatrix} \text{ where } X^0 = \begin{pmatrix} c_0 & k_0 & k_1 & k_2 \\ k_3 & c_1 & v_0 & v_1 \\ i_0 & i_1 & c_2 & k_4 \\ k_5 & k_6 & k_7 & c_3 \end{pmatrix} .$$

At each application of the initialisation process $\mathsf{F}(K, IV)$ 512 bits of keystream are generated by using the entirety of the final state as the keystream. The computation F is built around the quarterround function illustrated in Fig. 1 with quarterround$(y_0, y_1, y_2, y_3) = (z_0, z_1, z_2, z_3)$.

The operation columnround function updates all 16 words of the state X and can be described as follows. Each column i, $0 \leq i \leq 3$, is rotated upwards by i array positions. Each column is then used independently as input to the quarterround function. The resulting set of four columns, $0 \leq i \leq 3$, are then rotated down by i array positions. The operation rowround can be viewed as being identical to the columnround operation except that the state array is transposed both before and after using the columnround operation. Salsa20 updates the internal state by using columnround and rowround one after the other. After r rounds, the state is denoted X^r and the keystream given by $z = X^0 + X^r$ using wordwise addition modulo 2^{32}. The original version of Salsa20 has $r = 20$, *i.e.* 10 rounds of columnround interleaved with 10 rounds of rowround, though shorter versions with $r = 8$ and $r = 12$ have been proposed [2].

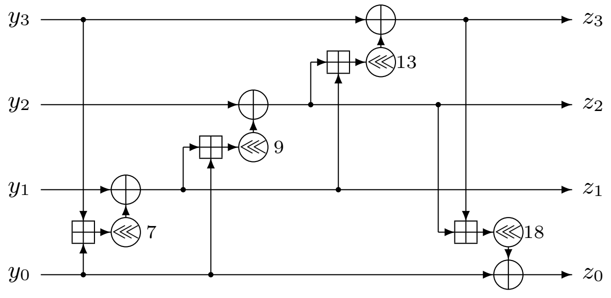

Fig. 1. The quarterround function of Salsa20.

3 Analysis of Salsa20

In this section we will demonstrate two related instances of non-random behaviour. These are detectable over six and seven rounds of Salsa20 respectively. Depending on the attack model this also permits key recovery. To start, we illustrate our approach by building on the earlier work of Crowley [5] and we describe a framework that allows a more sophisticated analysis to take place. This is achieved in two steps. First, we identify interesting differential effects in a simplified version of Salsa20. Second, we identify key and IV choices that allow us to ensure that the behaviour of the genuine Salsa20 is reasonably well-approximated by the simplified version. This technique allows us to make a systematic research of possible input differences $\mathcal{ID}$'s and consequently to find $\mathcal{ID}$'s with optimal properties.

As mentioned, our observations are differential in nature. We will work with two copies of the state where X^0 is filled with the input (K, IV) and a second state Y^0 is initialized according to $Y^0 = X^0 \oplus \Delta^0$ where $\Delta^0 = (\Delta^0_0, \ldots, \Delta^0_{15})$ is the $\mathcal{ID}$. Note that the specifications of Salsa20 require that any $\mathcal{ID}$ must be zero in the diagonal words Δ^0_0, Δ^0_5, Δ^0_{10}, and Δ^0_{15}. After r rounds of Salsa20 the output difference $\mathcal{OD}$ is given[3] by $\Delta^r = X^r \oplus Y^r$.

3.1 A Linearised Version of Salsa20

In previous work, Crowley [5] identified a truncated differential over three rounds of Salsa20. Consider setting $\Delta^0_i = 0$ for $i \neq 9$ and $\Delta^0_9 = \texttt{0x80000000}$. Then the following truncated differential for the first three rounds holds with a theoretical probability 2^{-12}. In practice a variety of effects conspire to give an average probability of 2^{-9}.

[3] Note that due to the feedforward in Salsa20 that uses addition modulo 2^{32} this is not necessarily the same as the difference in the corresponding keystream.

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & \texttt{0x80000000} & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \xrightarrow[\text{col}]{\substack{\text{col}\\ \text{row}}} \begin{pmatrix} ? & ? & ? & \texttt{0x02002802} \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \end{pmatrix}$$

Given the behaviour exhibited in $x_3^3 \oplus y_3^3$ it is tempting to look for some impact in the next round. Yet, it is not clear how to proceed in a methodical manner.

To establish an appropriate framework for analysis, we introduce an alternative algorithm LinSalsa20. This is identical to Salsa20 except that all integer additions have been replaced by exclusive-or. The corresponding round functions are denoted LinColumnround and LinRowround. Assume that two initial states X^0 and $Y^0 = X^0 \oplus \Delta^0$ are iterated by LinSalsa20. Then since LinSalsa20 is completely linear in GF(2), the difference $\Delta^r = X^r \oplus Y^r$ coincides exactly with computing r iterations of Δ^0 with LinSalsa20. This computation does not require knowledge of the key and we refer to a differential path generated by LinSalsa20 as a *linear differential*. It is straightforward to see that there are many (admissible) input differences for which the output of LinSalsa20 is trivially non-random.

Proposition 1. *Consider an input* $\Delta_i^0 \in \{\texttt{0xFFFFFFFF}, \texttt{0x00000000}\}$ *for all words* $i = 0, \ldots, 15$. *Then, for any number of updates with* LinSalsa20, *one has* $\Delta_i^r \in \{\texttt{0xFFFFFFFF}, \texttt{0x00000000}\}$.

However we need to be more careful. While LinSalsa20 allows some straightforward analysis, the further the behaviour of LinSalsa20 is from the true Salsa20, the less useful it will be. Since a differential of large Hamming weight is likely to induce carries and hence non-linear behaviour to the genuine Salsa20, we will need a linear differential of low Hamming weight. Such a differential is intended to offer a reasonably good approximation to the same differential in genuine Salsa20. We will consider a linear differential to be of low weight if any computation involving active words in the difference only uses words of low Hamming weight ($\ll 16$). Let us consider Crowley's differential within this linear model.

Example 1. Consider an input difference with $\Delta_9^0 = \texttt{0x80000000}$ as the one non-zero word. The weight of differences for the first four rounds is as follows.

$$\begin{pmatrix} 0&0&0&0 \\ 0&0&0&0 \\ 0&1&0&0 \\ 0&0&0&0 \end{pmatrix} \xrightarrow{\text{col}} \begin{pmatrix} 0&2&0&0 \\ 0&3&0&0 \\ 0&1&0&0 \\ 0&1&0&0 \end{pmatrix} \xrightarrow{\text{row}} \begin{pmatrix} 4&2&2&2 \\ 7&10&3&6 \\ 1&3&4&1 \\ 0&1&1&2 \end{pmatrix} \xrightarrow{\text{col}} \begin{pmatrix} 9&19&6&5 \\ 3&13&5&5 \\ 4&11&11&7 \\ 1&16&2&10 \end{pmatrix}$$
$$\xrightarrow{\text{row}} \begin{pmatrix} 13&13&14&10 \\ 13&13&13&19 \\ 16&18&19&11 \\ \boxed{11}&17&20&15 \end{pmatrix}$$

The top line of this differential is as far as Crowley goes, but when using LinSalsa20 it appears we can go one round further. Indeed, one can identify a low-weight linear differential for word x_{12}^4, among others. Note that x_{12} is a right-to-diagonal word (with wrap) and is updated first in round four; the 16 in x_{13}^3 has no effect on x_{12}^4. □

The linear model can also be used to find longer differentials. A well-chosen multi-bit input may cause smaller diffusion than a single-bit input; non-zero bits can be placed in positions where they are annihilated in the update process. To illustrate, we focus again on a single column where the weight of the input (starting with the diagonal element) is $(0, 2, 1, 1)$. With a fixed relative position of the non-zero bits in this input, one can obtain an output after the first round of the form $(0, 1, 0, 0)$. The absolute position of the non-zero bits and the choice of column are free parameters and naturally leads to an identified sub-class of inputs. These all have the same properties in LinSalsa20.

Example 2. Consider an input difference with non-zero words $\Delta_2^0 =$ 0x00000100, $\Delta_6^0 =$ 0x00001000, and $\Delta_{14}^0 =$ 0x80080000.

$$\begin{pmatrix} 0&0&1&0 \\ 0&0&1&0 \\ 0&0&0&0 \\ 0&0&2&0 \end{pmatrix} \xrightarrow{\text{col}} \begin{pmatrix} 0&0&0&0 \\ 0&0&0&0 \\ 0&0&0&0 \\ 0&0&1&0 \end{pmatrix} \xrightarrow{\text{row}} \begin{pmatrix} 0&0&0&0 \\ 0&0&0&0 \\ 0&0&0&0 \\ 1&1&3&4 \end{pmatrix} \xrightarrow{\text{col}} \begin{pmatrix} 4&1&3&4 \\ 1&2&4&8 \\ 1&0&7&10 \\ 3&1&3&14 \end{pmatrix}$$

$$\xrightarrow{\text{row}} \begin{pmatrix} 13&1&6&7 \\ 11&14&5&7 \\ 7&4&14&5 \\ 14&21&18&17 \end{pmatrix} \xrightarrow{\text{col}} \begin{pmatrix} 13&16&17&17 \\ 6&16&19&23 \\ 14&\boxed{13}&18&15 \\ 18&16&15&15 \end{pmatrix}$$

One can identify a truncated low-weight linear differential for x_9^5 which is an out-of-diagonal word. Note that some words in the final array may have a lower Hamming weight, but their generation required computations using average-weighted words and so they are unlikely to be relevant to genuine Salsa20. □

3.2 Differentials in True Salsa20

In Section 3.1 we identified classes of inputs (with a required $\mathcal{ID}$) which gave low-weight truncated linear differentials after four and five rounds of LinSalsa20. For genuine (nonlinear) Salsa20, the same differentials might not behave in the same way and a differential trail will depend on the input. Therefore to find optimal $\mathcal{ID}$'s and inputs we will need to consider which conditions allow the non-linear differential trail to be closely approximated by the linear differential.

The only non-linear operation in Salsa20 is integer addition in the quarterround function, denoted $x_a + x_b$. Given an $\mathcal{ID}$ (Δ_a, Δ_b), the nonlinear $\mathcal{OD}$ corresponds to the XOR of $x_a + x_b$ and $(x_a \oplus \Delta_a) + (x_b \oplus \Delta_b)$. Thus, the nonlinear $\mathcal{OD}$ is identical to the linear $\mathcal{OD}$, if

$$(x_a + x_b) \oplus ((x_a \oplus \Delta_a) + (x_b \oplus \Delta_b)) = \Delta_a \oplus \Delta_b \ . \tag{1}$$

Each non-zero bit in Δ_a and Δ_b may cause integer addition to create or annihilate a sequence of carry bits. Hence we focus on low-weight trails to keep more control of such events. Note that a difference in the most significant bit is always linear.

In the following sections, we will be indirectly considering Eq. 1 when placing conditions on the inputs so that a differential in Salsa20 follows a linear differential in LinSalsa20 for some steps before diverging. Such conditions might be on the nonce, on the counter (conditions which can be satisfied by sampling a keystream in the appropriate way), or on the key (thereby establishing classes of weak keys). While many of these issues are complex, for instance, conditions aimed at linearising the $\mathcal{ID}$ must not conflict with the way the counter is likely to be incremented, some results are given in the subsequent sections.

3.3 Non-randomness in Four and Five Rounds of Salsa20

Consider the linear differential of Ex. 1 and set $\mathcal{ID}$ to be identical to that of [5]. By using LinSalsa20 we suspect a statistical imbalance in $x_{12}^4 \oplus y_{12}^4$. Given a set of N different pairs of (K, IV), where each pair takes the same fixed $\mathcal{ID}$, the distribution of the output difference for the N pairs can be analysed. However, we might consider a subset of the bits or even a single bit, and by examining each bit in $x_{12}^4 \oplus y_{12}^4$ one finds that bit 26 is heavily unbalanced[4]. This imbalance can be detected using a χ^2 test (see Appendix A) where a χ^2 score greater than some threshold is good evidence of non-randomness.

The behaviour of the differential is heavily key-dependent. The presence or absence of carries, on which Salsa20 relies, depends on the actual values of the operands. Thus some keys will dampen, and others amplify, the evolution of a differential. The imbalance in bit 26 is greater the closer Salsa20 is to LinSalsa20. A close inspection of the first round of the differential reveals that the first two additions, differentially speaking, act as XOR while the third does not. However, depending on how i_1 is incremented, we can establish conditions on the key to ensure that it does. Thus there are keys for which the imbalance in bit 26 is boosted. We refer to this as partially *linearising* the first round of the differential[5] and key conditions that achieve this are presented in Appendix B.

Example 3. Take N inputs (K, IV) where $IV = (v_0, v_1, i_0, i_1)$. The key K and nonce (v_0, v_1) are chosen at random though in the second experiment some bits of k_0 and k_6 are adapted. The counter (i_0, i_1) starts at zero and we sample the keystream so that the counter i_0 increments from 0 to $N-1$. For each input, we use values of i_1 to generate an associate input with $\mathcal{ID}$ $\Delta_9^0 = \texttt{0x80000000}$ (and zero otherwise). Compute the $\mathcal{OD}$ after four rounds of Salsa20 and evaluate the χ^2 statistic on bit 26 of Δ_{12}^4. In a χ^2 test on a single bit with threshold $T = 40$, the probability a uniform random source gives $\chi^2 > 40$ is around 2^{-32}, thus the probability of false alarm is 2^{-32}. For 100 experiments using random keys and nonces, the results are listed in Tab. 1. □

[4] In fact there are many unbalanced bits in the state of Salsa20 after four rounds.

[5] A more sophisticated set of conditions can be derived to linearise the entirety of the first round. However for clarity we restrict ourselves to the simpler case.

© Springer-Verlag Berlin Heidelberg 2006

Table 1. Non-randomness in four rounds of Salsa20.

	All Keys and Nonces		*Weak Key Class*	
N	*average* χ^2 *value*	% values > 40	*average* χ^2 *value*	% values > 40
2^{12}	33	20	51	34
2^{14}	123	41	192	46
2^{16}	315	46	656	68

Next we consider five rounds of Salsa20 and we use the differential of Ex. 2. The non-zero bits are located in column two. Word x_{14} is updated first by $x^1_{14} = x^0_{14} \oplus (x^0_{10} + x^0_6)_{\lll 7}$. A second state $y^0_i = x^0_i \oplus \Delta^0_i$ is updated in the same way and, according to Eq. 1, the difference of this first update will follow the linear differential if

$$(x^0_{10} + x^0_6) \oplus \big((x^0_{10} \oplus \Delta^0_{10}) + (x^0_6 \oplus \Delta^0_6)\big) = \Delta^0_{10} \oplus \Delta^0_6 \ . \qquad (2)$$

Notice that Δ^0_{10} is zero and that Δ^0_6 has a single non-zero bit in position 12. Further, $x^0_{10} = c_2$ and $x^0_6 = v_0$. Bits $12 \ldots 9$ of c_2 are defined as $(\ldots \mathtt{0110} \ldots)_2$. Consequently, if bits $11 \ldots 9$ of v_0 are chosen as $(\ldots \mathtt{000} \ldots)_2$, then no carry is produced from the right, and Eq. 1 is satisfied. Subsequently x_2 is updated and so provided the previous update followed the linear differential, the only non-zero bit in the difference will be in bit 31 and the linear trial will be followed. Updating x_6 is similar while updating x_{11} only involves zero differences.

Thus we have identified conditions on three bits of v_0, part of the nonce, so that the first round of genuine Salsa20 with the $\mathcal{ID}$ of Ex. 2 follows the linear trail. In fact, the $\mathcal{ID}$ of Ex. 2 turns out to be optimal, *i.e.* it seems to have minimum weight after two rounds of Salsa20; bitwise rotations of $\mathcal{ID}$ reduce the number of msb's while shifting the difference to another column shifts the input-condition to a key word instead of v_0. Without input conditions on v_0, the first round would follow the linear trail with a probability of about $\Pr = 0.175$.

Example 4. Take N inputs (K, IV) where $IV = (v_0, v_1, i_0, i_1)$. The key K and nonce (v_0, v_1) are chosen at random though in the second experiment bits 9–11 of v_0 are zero. The counter (i_0, i_1) starts at zero and we sample the keystream so that the counter i_0 increments from 0 to $N-1$. For each input, we use values of k_1, v_0, k_7 to generate an associate input with $\mathcal{ID}$ $\Delta^0_2 = \mathtt{0x00000100}$, $\Delta^0_6 = \mathtt{0x00001000}$, $\Delta^0_{14} = \mathtt{0x80080000}$ (and zero otherwise). Compute the $\mathcal{OD}$ after five rounds of Salsa20 and evaluate the χ^2 statistic on bit 1 of Δ^5_9. In a χ^2 test on a single bit with threshold $T = 40$, the probability a uniform random source gives $\chi^2 > 40$ is around 2^{-32}. For 100 experiments using random keys and nonces, the results are listed in Tab. 2. □

3.4 Non-randomness in Six and Seven Rounds of Salsa20

The results presented in Section 3.3 give statistical weaknesses, as measured by the χ^2 test on a single bit, over four rounds and five rounds of Salsa20. To create

Table 2. Non-randomness in five rounds of Salsa20.

	All Keys and Nonces		*Weak Nonce Class*	
N	*average* χ^2 *value*	% values > 40	*average* χ^2 *value*	% values > 40
2^{20}	5	4	27	26
2^{22}	16	11	105	73
2^{24}	78	17	383	89

these biased distributions, we used $\mathcal{ID}$'s of slightly different types. For the four round imbalance we use a non-zero difference in the counter values while for the five round imbalance we use non-zero differences in part of the key k_1 and k_7. We will comment on this later since it has an impact on the attacks we can mount.

Both statistical anomalies can be detected two rounds later. In both cases we intercept the required keystream and we guess the necessary key words to unwind the last two rounds of state update. Thus, for a single guess of the relevant words of key, the backwards computation is carried out over two rounds for N pairs of output, where each output was generated using the chosen input difference. The χ^2 statistic of the target bit of the target word is evaluated, and a χ^2 test with some threshold T applied. Our analysis tells us that a correct key guess will yield a significant χ^2 score. We assume that an incorrect key guess results in essentially random candidate values for the bit we test. Thus, a significantly large χ^2 value suggests that the key guess may be correct. The remaining key words can be searched exhaustively and the entire key guess verified against the keystream. If the χ^2 value for a key guess is not significant we move on to a new guess. The target word and bit as well as the key words to guess are given in Tab. 3.

Table 3. Key words to guess to partially unwind the last two rounds.

Differential	*Word*	*Bit*	*# Rounds*	*Key Words to Guess*
Example 3	Δ^4_{12}	26	Salsa20/6	k_3 k_4 k_5 k_6 k_7
Example 4	Δ^5_9	1	Salsa20/7	k_0 k_2 k_3 k_4 k_5 k_6

Clearly, the scale of the imbalance in the target bit is important to the success of this method. The closer Salsa20 behaves to LinSalsa20 then the greater the imbalance in the target bit, and the greater the χ^2 score we expect to observe. This helps an attacker in two ways:

1. If certain keys and IV's give a high χ^2 score, then a greater proportion of the keys from an identified set should be susceptible to attack.
2. Higher χ^2 scores permit less keystream or greater precision in an attack.

To begin to get a picture of how things might behave in practice, we have implemented a restricted version of this style of attack. In principle we could use

© Springer-Verlag Berlin Heidelberg 2006

the four round differential of Ex. 3 to attack six rounds of Salsa20. To keep the experiments tractable, however, we use the same differential to attack a restricted five-round version as a demonstration (*i.e.* we unwind one round only).

Example 5. We recover nine bits (bits 4 to 12) of k_3 under the assumption that k_5 has been correctly guessed. Over 100 random keys and nonces and N pairs, we give the success rate when assuming the correct key lies among the candidate values giving the three highest χ^2 values. We repeat the experiment for the weak key class identified in Ex. 3. For the weak key class we observe that the same proportion of keys can be recovered when using one quarter of the text, see Tab. 4. We recall that the weak keys only improve the differential propagation and that our attack is also working for other keys. □

Table 4. Demonstration of a key recovery attack on five rounds of Salsa20.

	All Keys and Nonces	*Weak Key Class*
N	*% success rate*	*% success rate*
2^{12}	20	28
2^{14}	29	41
2^{16}	44	54

As demonstrated in Ex. 5, at least in principle, our observations can be used in the way we intend. In the case of Salsa20/6 we estimate the work effort for a key-recovery attack to be around 2^{177} operations using 2^{16} pairs of keystream blocks sampled appropriately from the same keystream[6]. This is a crude estimate. Since such an attack requires guessing more key bits, more text may well be required. However, since the entirety of the target word can be recovered for any single key guess, using a single bit to test a key will miss much of the information available. Thus, it seems prudent to anticipate a final complexity close to these initial estimates. Under a related-key attack Salsa20/7 might be broken in around 2^{217} operations using 2^{24} pairs of keystream blocks taken from two sets of keystream. However, the practical validity of such an attack is debatable [3], so we merely observe that over seven of the 20 rounds in Salsa20, statistical imbalances can be detected.

4 Description of TSC-4

The stream cipher TSC-4 is specified in [12]. It consists of two states X and Y of 4×32 bits each, denoted $X = (x_0, x_1, x_2, x_3)^T$ and $Y = (y_0, y_1, y_2, y_3)^T$. Let $[x]_i$ denote bit i of a single 32 bit word x, then a bit-slice i of state X is

[6] We note in passing that we can recover the key for the 128-bit version of Salsa20/5 in 2^{81} operations using 2^{16} pairs of keystream blocks.

© Springer-Verlag Berlin Heidelberg 2006

defined as $([x_3]_i, [x_2]_i, [x_1]_i, [x_0]_i)$. We first describe the regular update function f, which updates the two states X and Y independently by single-cycle T-functions. In the case of state X, a 32-bit parameter α_X is computed as a function of X. It is defined by $\alpha_X = p \oplus (p + c_X) \oplus s$ with $p = x_0 \wedge x_1 \wedge x_2 \wedge x_3$ and $s = (x_0 + x_1 + x_2 + x_3) \lll 1$ and constant $c_X = \texttt{0x51291089}$. If $[\alpha_X]_i = 1$, then a fixed S-box S is applied to bit-slice i of X, and if $[\alpha_X]_i = 0$, then a fixed S-box S^6 is applied to bit-slice i of x (for all $i = 0, \ldots, 31$). The state Y is similarly updated where parameter α_Y has constant $c_Y = \texttt{0x12910895}$. Notice that the least significant bit-slice is always mapped by S. The S-boxes are defined as

$$\mathsf{S} = \{9, 2, 11, 15, 3, 0, 14, 4, 10, 13, 12, 5, 6, 8, 7, 1\}$$
$$\mathsf{S}^6 = \{6, 13, 8, 0, 5, 12, 1, 11, 4, 14, 3, 10, 15, 7, 2, 9\} \ .$$

The output function g produces a keystream byte z by combining some bytes of both states (using integer addition, `XOR`, shift and rotation), see [12] for more details.

Consider the initialisation function F of $\mathsf{TSC\text{-}4}$. To start, the internal state of 256 bits is loaded with $K = (k_0, \ldots, k_9)$ and $IV = (i_0, \ldots, i_9)$ each of 10×8 bits (a single 32-bit word is denoted as a concatenation of four 8-bit words).

$$X = \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} k_3 & k_2 & k_1 & k_0 \\ k_7 & k_6 & k_5 & k_4 \\ i_3 & i_2 & i_1 & i_0 \\ i_7 & i_6 & i_5 & i_4 \end{pmatrix}, \quad Y = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} i_1 & i_0 & i_9 & i_8 \\ i_5 & i_4 & i_3 & i_2 \\ k_1 & k_0 & k_9 & k_8 \\ k_5 & k_4 & k_3 & k_2 \end{pmatrix}$$

A single round of the initialisation function (denoted as a *warm-up* update) consists of a regular update and some additional operations: A byte $z = \mathsf{g}(X)$ is produced, x_1 and y_0 are rotated to the left by eight positions, and then byte z is `XOR`-ed to the 8 least significant bits of x_1 and y_0. The specifications of $\mathsf{TSC\text{-}4}$ propose $r = 8$ rounds.

5 Analysis of TSC-4

In [10,11], predecessors of $\mathsf{TSC\text{-}4}$ have been attacked by exploiting a bit-flip bias for multiple applications of the state update function f. This bias still exists for regular updates of $\mathsf{TSC\text{-}4}$, but the strong filter function g prevents from an attack. In this section, we disregard the details of the filter function and investigate the statistical properties of multiple warm-up updates of $\mathsf{TSC\text{-}4}$: While the regular updates have some guaranteed properties, the warm-up updates use additional *ad hoc* operations that are designed to accelerate diffusion. Notice that our analysis is embedded in a more general context: we actually consider the initialisation function F of $\mathsf{TSC\text{-}4}$ and try to detect some non-random behaviour in a set of outputs (*i.e.* in the $\mathsf{TSC\text{-}4}$ initial states) that are produced by a set of well-chosen inputs (*i.e.* in the IV's).

5.1 Statistical Model of Initialisation

We investigate the statistical properties of the initialisation process. In our simple statistical model, we assume that α (with exception of the lsb) and the feedback z are uniformly randomly. Consider a single bit-slice i (not the least significant one) in the state X, then our assumptions imply for each round:

1. Bit-slice i is mapped uniformly randomly by S or by S^6.
2. After application of the S-box, bit 1 of bit-slice i is chosen uniformly randomly.

With a fixed input $w \in \{0, \ldots, 15\}$, these two steps are repeated for r rounds, so we can analyse the distribution of the output $v \in \{0, \ldots, 15\}$. Within this model, the distribution can be computed exactly in 2^{2r} steps. The other cases (*i.e.* the least-significant bit-slice and the state Y) are treated similarly. The bias of the distribution is measured with the *Euclidean Squared Distance* $\varepsilon^2 := \sum \varepsilon_v^2$ with $\varepsilon_v := \Pr(v) - 1/16$, where $\Pr(v)$ denotes the probability for output v (given some fixed parameters). In Tab. 5, the bias ε^2 is shown for different parameters. To simplify the presentation we compute ε^2 for all inputs w and show the *average* values only.

Table 5. Average bias ε^2 in the statistical model for $r = 6, \ldots, 12$ rounds, and for different bit-slices.

	lsb in X	lsb in Y	non-lsb in X	non-lsb in Y
$r = 6$	7.1×10^{-3}	4.2×10^{-3}	7.7×10^{-5}	1.8×10^{-5}
$r = 8$	9.7×10^{-4}	2.1×10^{-4}	4.5×10^{-6}	4.6×10^{-7}
$r = 10$	1.3×10^{-4}	7.6×10^{-6}	2.1×10^{-7}	9.8×10^{-9}
$r = 12$	2.3×10^{-5}	1.0×10^{-6}	5.5×10^{-9}	2.1×10^{-10}

As expected, the average bias is decreasing with the number of rounds r. In the case of the least-significant bit-slice in the state X, it is reduced by a factor of about 2.6 with each additional round. Interestingly, the position of the random bit (step 2) has a notable influence on the distribution and diffusion is better for state Y. And, as expected, diffusion is better for bit-slices which are not on the least-significant position (intuitively a combination of S and S^6 results in larger diffusion than using S only).

5.2 Experimental Results

Now we attempt to detect the bias of the previous subsection in the genuine initialisation function $\mathsf{F}(K, IV)$ of $\mathsf{TSC\text{-}4}$. We need N different inputs (K, IV) where the value of a fixed bit-slice i is the same for all inputs. Each bit-slice consists of two key bits and two IV bits. Consequently, bit-slice i is the same for all inputs, if the key is fixed (and unknown), and if the IV bits of bit-slice i are fixed (though the other IV bits can be varied). The N outputs can then be used to evaluate a χ^2 statistic on bit-slice i. Provided that the assumptions on the model of the previous section are valid, bit-slice $i = 0$ of the state X is expected to have maximum bias. Here is an example for $r = 8$ rounds.

Example 6. Take N different inputs (K, IV) where $IV = (i_0, \ldots, i_9)$. The key is fixed, IV bytes $i_0, i_1 \ldots, i_7$ are zero, and i_8, i_9 increments from 0 to $N - 1$. Compute all N outputs after $r = 8$ rounds of $\mathsf{F}(K, IV)$ and evaluate the χ^2 statistic on the least-significant bit-slice in the state X of the output. In a χ^2 test on 4 bits with threshold $T = 80$, the probability a uniform random source gives $\chi^2 > 80$ is around 2^{-34}. For 100 experiments using random keys, the results are listed in Tab. 6. $\square$

Table 6. Average χ^2 statistic in the experiment for $r = 8$ rounds and a varying number of samples.

	All Keys	
N	*average* χ^2 *value*	% values > 80
2^{10}	40	3
2^{12}	119	67
2^{14}	421	100

For all three choices of N, the assigned bias $\varepsilon^2 = (\chi^2 - 15)/16N$ becomes about $2^{-9.2}$ (see Appendix A). This is in good agreement with the model of Section 5.1, which predicts an average bias of $\varepsilon^2 = 2^{-9.8}$ in this setup[7]. Of course, the initial state cannot be accessed by an attacker, so the χ^2 test has perhaps a certificational character. However, the setup of Ex. 6 does not require any key bit to be known, and the number of samples N is very small. Consequently, this non-randomness may be a basis for future attacks that includes analysis of the filter function g.

The non-randomness is not limited to the least significant bit-slice. A notable example is $i = 8$ (and with other parameters as in Ex. 6), which results in an average value of $\chi^2 = 45$ for $N = 2^{10}$. This is a consequence of the specific setup in Ex. 6 where bit-slices $i = 8, 9 \ldots$ of X after the first round are the same for all N states and so the effective number of rounds is only $r - 1$ (in addition, the biased bit 1 of bit-slice 0 is rotated into bit-slice 8).

The above experiment with $i = 8$ was carried out for a varying number of rounds, see Fig. 2. In order to measure χ^2 values for a larger number of rounds, we increased the number of samples to $N = 2^{18}$. This was done by choosing zero IV bytes $i_0, i_1, \ldots, i_5$ and counting up i_6, i_7, i_8, i_9 from 0 to $N-1$. Supplementary experiments revealed that ε^2 is approximately constant for different values of N,

[7] Notice that two input bits of bit-slice $i = 0$ are always zero in the setup of Ex. 6. This has a small influence on the modeled bias in Tab. 5.

© Springer-Verlag Berlin Heidelberg 2006

r	χ^2	ε^2
6	47593	1.1×10^{-2}
8	6067	1.4×10^{-3}
10	1437	3.4×10^{-4}
12	260	5.8×10^{-5}
14	44	6.8×10^{-6}

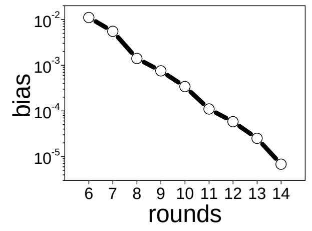

Fig. 2. Average χ^2 and the assigned bias ε^2 for $r = 6, \ldots, 14$ rounds (where the bias ε^2 is plotted on the right).

hence ε^2 is a good measure for the diffusion of F. The bias ε^2 in terms of r can be approximated by an exponential decay and in one round ε^2 is reduced by a factor of about 2.5. By extrapolation, we expect that about $r = 32$ rounds would be necessary to obtain a bias of $\varepsilon^2 = 2^{-40}$. In an extended experiment one could also measure the effectiveness of the combined initialisation function F and update function f^t. For example, with $r = 8$, $t = 50$ and $N = 2^{18}$, we observed an average value of $\chi^2 = 32$ when using the same setup as previously. However we did not observe a bias in the keystream.

6 Conclusions

In this paper we considered the way the key and the initialisation information is used in two Phase 2 candidates in eSTREAM. In the case of TSC-4, the initial cipher state is derived using eight applications of a warm-up function. Non-randomness over all eight iterations can be detected in the initial state with about 1000 inputs. Each additional round increases the data requirements by a factor of about 2.5 and this non-randomness requires the attacker to choose IV bits only. However no bias in the keystream of TSC-4 resulting from this non-randomness has yet been detected, so it remains to be seen if our observations can form the basis for an attack in the future. As the rating of *Focus* candidate in eSTREAM Phase 2 testifies, Salsa20 is widely viewed as a very promising proposal. Nothing in this paper affects the security of the full version of the cipher. However we expect that the key can be recovered from five rounds of 128-bit Salsa20 with around 2^{81} operations and six rounds of 256-bit Salsa20 with around 2^{177} operations. Both attacks would require very moderate amounts of text. If we allow related-key attacks then the security of seven rounds of 256-bit Salsa20 might be in question with around 2^{217} operations. However, given divided opinions on such an attack model, we prefer to observe that a statistical weakness has been observed over seven rounds. While we anticipate some progress, we are doubtful that many more rounds can be attacked using the methods of this paper. Thus Salsa20 still appears to be a conservative design. Given our results, however, we are doubtful that Salsa20/8 will offer adequate security in the future, though Salsa20/12 could turn out to be a well-balanced proposal.

Acknowledgments

The first and second author are supported in part by grant 5005-67322 of NCCR-MICS (a center of the Swiss National Science Foundation). The second author also receives partial funding through GEBERT RÜF STIFTUNG. Other authors are supported by the French Ministry of Research RNRT X-CRYPT project and by the European Commission through the IST Program under Contract IST-2002-507932 ECRYPT.

References

1. D.J. Bernstein. Salsa20. In *eSTREAM, ECRYPT Stream Cipher Project*, Report 2005/025.
2. D.J. Bernstein. Salsa20/8 and Salsa20/12. In *eSTREAM, ECRYPT Stream Cipher Project*, Report 2006/007.
3. D.J. Bernstein. Related-key attacks: who cares? In *eSTREAM discussion forum*, June 22, 2005. `http://www.ecrypt.eu.org/stream/phorum/read.php?1,23`.
4. A. Biryukov. A New 128 Bit Key Stream Cipher: LEX. In *eSTREAM, ECRYPT Stream Cipher Project*, Report 2005/013.
5. P. Crowley. Truncated Differential Cryptanalysis of Five Rounds of Salsa20. In *eSTREAM, ECRYPT Stream Cipher Project*, Report 2005/073.
6. J. Daemen, R. Goverts, and J. Vandewalle. Resynchronization Weaknesses in Synchronous Stream Ciphers. In *Advances in Cryptology - EUROCRYPT 1993*, LNCS 765, pages 159–167. Springer-Verlag, 1993.
7. M. Dichtl and M. Schafheutle. Linearity Properties of the SOBER-t32 Key Loading. In *Fast Software Encryption 2002*, LNCS 765, pages 159–167. Springer-Verlag, 1993.
8. P. Ekdahl and T. Johansson. Another Attack on A5/1. In *IEEE Transactions on Information Theory*, volume 49/1, pages 284–289, 2003.
9. S.R. Fluhrer, I. Mantin, and A. Shamir. Weaknesses in the Key Scheduling Algorithm of RC4. In *Selected Areas in Cryptography 2001*, LNCS 2259, pages 1–24. Springer-Verlag, 2001.
10. S. Künzli, P. Junod, and W. Meier. Distinguishing Attacks on T-functions. In *Progress in Cryptology - Mycrypt 2005*, LNCS 3715, pages 2–15. Springer-Verlag, 2005.
11. F. Muller and T. Peyrin. Linear Cryptanalysis of the TSC Family of Stream Ciphers. In *Advances in Cryptology - ASIACRYPT 2005*, LNCS 3788, pages 373–394. Springer-Verlag, 2005.
12. D. Moon, D. Kwon, D. Han, J. Lee, G.H. Ryu, D.W. Lee, Y. Yeom, and S. Chee. T-function Based Streamcipher TSC-4. In *eSTREAM, ECRYPT Stream Cipher Project*, Report 2006/024.

A The χ^2 Test

Let $X := X_1, X_2, \ldots, X_N$ denote N *i.i.d.* random variables where each $X_i \in \{x_0, \ldots, x_m\}$ and with unknown distribution. A χ^2 test is applied on the observation X, in order to decide if the observation is consistent with the hypothesis that X_i have distribution D_U. Let N_i be the number of observations x_i in X, and E_i the expectation for x_i under distribution D_U. Then, the χ^2 statistic is a random variable defined by

$$\chi^2 := \sum_{i=1}^{m} \frac{(N_i - E_i)^2}{E_i} \ . \tag{3}$$

In the case of a uniform distribution D_U, one has $E_i = N/m$. The χ^2 statistic (for large N) is then compared with the threshold of the $\chi^2_{\alpha,m-1}$ distribution having $m-1$ degrees of freedom and significance level α. Consequently, a χ^2 test can be defined by a threshold T, where the hypothesis is accepted if $\chi^2(X) < T$.

If X has uniform distribution D_U, the expectation of χ^2 becomes $E_U(\chi^2) = m-1$. If X has another distribution D_X (which is assumed to be close to D_U), then the expectation of χ^2 becomes about $E_X(\chi^2) = (c+1)m - 1$, where $c := N\varepsilon^2$ and $\varepsilon^2 := \sum \varepsilon_i^2$ with probability bias $\varepsilon_i := \Pr_X(x_i) - \Pr_U(x_i)$. Notice that $E_X(\chi^2)$ differs from $E_U(\chi^2)$ significantly, if $c = \mathcal{O}(1)$. Consequently, about $N = \mathcal{O}(1/\varepsilon^2)$ samples are required to distinguish a source with distribution D_X from a source with distribution D_U.

B Weak Key Conditions for Example 3

The key conditions for the weak key class of Ex. 3 are on k_0 and k_6. First set the following bits of k_0 to the values shown:

bit number:	0	1	20	21	22	23
value:	0	1	0	0	1	1

Next set bit 7 of k_6 equal to bit 7 of T where $c_1 =$ `0x3320646E` and

$$T = (((k_0 + c_1) \lll 7) + c_1) \lll 9.$$

Note that all these conditions are randomly satisfied with a probability of 2^{-7}.