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.

Keywords: 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, 9, 8]). Good mixing of the secret key is similarly required and there should not be any identifiable subsets of keys that have a traceable

R. Barua and T. Lange (Eds.): INDOCRYPT 2006, LNCS 4329, pp. 2–16, 2006.

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 $\mathsf{quarterround}(y_0, y_1, y_2, y_3) = (z_0, z_1, z_2, z_3)$.

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].