Salsa20 Cryptanalysis: New Moves and Revisiting Old Styles

Subhamoy Maitra, Goutam Paul, Willi Meier

Salsa20 Cryptanalysis: New Moves and Revisiting Old Styles*

Subhamoy Maitra[1], Goutam Paul[1], Willi Meier[2]

[1] Indian Statistical Institute, Kolkata, India, {subho,goutam.paul}@isical.ac.in
[2] FHNW, Windisch, Switzerland, willi.meier@fhnw.ch

Abstract. In this paper, we revisit some existing techniques in Salsa20 cryptanalysis, and provide some new ideas as well. As a new result, we explain how a valid initial state can be obtained from a Salsa20 state after one round. This helps in studying the non-randomness of Salsa20 after 5 rounds. In particular, it can be seen that the 5-round bias reported by Fischer et al. (Indocrypt 2006) is a special case of our analysis. Towards improving the existing results, we revisit the idea of Probabilistic Neutral Bit (PNB) and how a proper choice of certain parameters reduce the complexity of the existing attacks. For cryptanalysis against 8-round Salsa20, we could achieve the key search complexity of $2^{247.2}$ compared to the earlier results of 2^{251} (FSE 2008) and 2^{250} (ICISC 2012).

Keywords: Stream Cipher, Salsa20, Salsa20/12, Non-Randomness, Round Reversal, Probabilistic Neutral Bit (PNB), ARX Cipher.

1 Introduction

Salsa20 [2] was designed by Bernstein in 2005 as a candidate for eStream [9] and Salsa20/12 has been accepted in the eStream software portfolio. It has generated serious attention in the domain of cryptanalysis and quite a few results have been published in this direction [3, 4, 8, 1, 5, 7, 6] that show weaknesses of this cipher in reduced rounds. The main ideas in all these papers are circled around the following two issues. (i) Put some input difference at the initial state and then try to obtain certain bias in some output differential. (ii) Once you can move forward a few rounds with the above strategy, try to come back a few rounds from a final state after a certain rounds obtaining further non-randomness. Combining the above ideas, one may find some non-randomness in Salsa20 for forward plus backward many rounds. The first result on Salsa20 cryptanalysis by [3] used this idea to move forward 3 rounds from the initial state and to come back 2 rounds from the final state for an attack on 5-round Salsa20. Later in [1], biased differentials till 4 forward rounds have been exploited and then the attack considered moving backwards from the final state for 4 rounds to obtain an 8-round attack on Salsa20.

* The reader may refer to http://eprint.iacr.org/2015/217 for the extended version of this paper.

Before proceeding further, let us briefly explain the structure of Salsa20 stream cipher. This cipher considers 16 words, each of 32 bits. We may refer to this as the Salsa state that can be written in 4×4 matrix format as follows:

$$X = \begin{pmatrix} x_0 & x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 & x_7 \\ x_8 & x_9 & x_{10} & x_{11} \\ x_{12} & x_{13} & x_{14} & x_{15} \end{pmatrix} = \begin{pmatrix} c_0 & k_0 & k_1 & k_2 \\ k_3 & c_1 & v_0 & v_1 \\ t_0 & t_1 & c_2 & k_4 \\ k_5 & k_6 & k_7 & c_3 \end{pmatrix}.$$

The rightmost matrix shows the initial state, that takes four predefined constants $c_0, \ldots, c_3$, 256-bit key $k_0, \ldots, k_7$, 64-bit nonce v_0, v_1 and 64-bit counter t_0, t_1.

The basic nonlinear operation of Salsa20 is the quarterround function. Each $\mathsf{quarterround}(a, b, c, d)$ consists of four ARX rounds, each of which comprises of one addition (A), one cyclic left rotation (R) and one XOR (X) operation as given below.

$$\left.\begin{aligned} b &= b \oplus ((a \ + \ d) \lll 7), \\ c &= c \oplus ((b \ + \ a) \lll 9), \\ d &= d \oplus ((c \ + \ b) \lll 13), \\ a &= a \oplus ((d \ + \ c) \lll 18). \end{aligned}\right\} \tag{1}$$

Each columnround works as four quarterrounds on each of the four columns of the state matrix and each rowround works as four quarterrounds on each of the four rows of the state matrix. In Salsa20 (one can call it Salsa20/20), ten times the columnround and ten times the rowround are applied alternatively on the initial state. One may note that this can be considered as application of the columnround and transpose of the state matrix [6] twenty times. This helps in understanding the cipher better as in this case every round of Salsa20 becomes identical. To be precise, in each round, we first apply quarterround on all the four columns in the following order: $\mathsf{quarterround}(x_0, x_4, x_8, x_{12})$, $\mathsf{quarterround}(x_5, x_9, x_{13}, x_1)$, $\mathsf{quarterround}(x_{10}, x_{14}, x_2, x_6)$, and $\mathsf{quarterround}(x_{15}, x_3, x_7, x_{11})$, and then a $\mathsf{transpose}(X)$ as considering it as a 4×4 matrix.

By $X^{(r)}$, we mean that r rounds have been applied on the initial state X. Hence $X^{(0)}$ is the same as the initial state X. Finally, after R rounds we have $X^{(R)}$. Then a keystream block of 16 words or 512 bits is obtained as $Z = X + X^{(R)}$. For Salsa20, $R = 20$. However, the one accepted in eStream [9] software portfolio is Salsa20/12, where $R = 12$. Naturally, more rounds will provide better security and less rounds will provide higher speed.

One may note that each Salsa20 round is reversible as the state-transition operations are reversible. In other words, if $X^{(r+1)} = \mathsf{round}(X^{(r)})$, then $X^{(r)} = \mathsf{reverseround}(X^{(r+1)})$, where reverseround is the inverse of round and consists of first transposing the state and then applying the inverse of quarterround for each column as follows.

$$\left.\begin{aligned} a &= a \oplus ((d \ + \ c) \lll 18), \\ d &= d \oplus ((c \ + \ b) \lll 13), \\ c &= c \oplus ((b \ + \ a) \lll 9), \\ b &= b \oplus ((a \ + \ d) \lll 7). \end{aligned}\right\} \tag{2}$$

Consider that one obtains a state $X^{(1)}$ after one round of Salsa20. Now to know whether it is a valid state after one round, one needs to come back by one reverse round and then check whether the constants in the diagonal elements are indeed the specified ones. This is the constraint for 256-bit Salsa20. However, for 128-bit Salsa20, one needs to have another constraint related to the key words apart from matching the constants. That is, we need to have $k_i = k_{i+4}$, for $0 \leq i \leq 3$.

Let us now present a few more notations. We have already denoted x_i as the i-th word of the matrix X. By $x_{i,j}$, we mean the j-th bit (0-th bit is the least significant bit) of x_i. Given two states $X^{(r)}, X'^{(r)}$, we denote $\Delta_i^{(r)} = x_i^{(r)} \oplus x_i'^{(r)}$. By $\Delta_{i,j}^{(r)} = x_{i,j}^{(r)} \oplus x_{i,j}'^{(r)}$, we mean the difference between two states at the j-th bit of the i-th word after r many rounds. That is, '$\Delta_{7,31}^{(0)} = 1$' means that we have two initial states $X^{(0)}, X'^{(0)}$ that differ at the 31-st (most significant) bit of the 7-th word (a nonce word) and they are same at all the other bits of the complete state. In such a case, one can obtain significant biases after 4 rounds. That is, in general, we are interested to input a difference at the initial state (call it Input Differential or $\mathcal{ID}$) and then try to obtain some bias corresponding to combinations of some output bits (call it Output Differential or $\mathcal{OD}$). Most naturally, one can compute $\Pr(\Delta_{p,q}^{(r)} = 1 | \Delta_{i,j}^{(0)} = 1) = \frac{1}{2}(1 + \epsilon_d)$, where the probability is estimated for a fixed key and all the possible choices of nonce and counter words v_0, v_1 and t_0, t_1 respectively, other than the constraints imposed due to the input differences on keys or nonces/counters. The term ϵ_d is a measure of the amount of the bias away from the probability of a random event and is naturally referred as the *bias* in the literature.

Now we present the advantage of coming back one round. Consider that one likes to put some input differences in a state to obtain some output differential after a few rounds. Let $X^{(1)}, X'^{(1)}$ be the two initial states that differ in a few places and we obtain some bias $\Pr(\Delta_{p,q}^{(r)} = 1 | \Delta_{i,j}^{(1)} = 1) = \frac{1}{2}(1 + \epsilon_d)$. That is, actually we consider $(r-1)$ rounds here. If it is possible to get back one round each from $X^{(1)}, X'^{(1)}$ such that $X^{(0)}, X'^{(0)}$ are valid initial states with differentials $\Delta_{i_1,j_1}^{(0)}, \Delta_{i_2,j_2}^{(0)}, \ldots$ etc, then we can consider the situation $\Pr(\Delta_{p,q}^{(r)} = 1 | \Delta_{i_1,j_1}^{(0)} = 1, \Delta_{i_2,j_2}^{(0)} = 1, \ldots) = \frac{1}{2}(1 + \epsilon_d)$. That is, we can exploit the bias as if it is after r rounds. This helps in analysing Salsa20 by one additional round.

In Section 2, we explain how one can come back to a valid initial state from a Salsa20 state after one round. This helps in interpreting many four round biases as five round ones and provides a generalized framework for the 5-round biased differential pointed out in [4]. In Section 3, we revisit the cryptanalysis of Salsa20/8 as described in [1]. Interestingly, we note that with experiments the median of certain biases are 4 times more than what observed in [1]. Further, we look at the trade-off between carefully choosing more PNBs at the cost of accepting less probability for distinguishing the correct key from the wrong keys. In the process, we show that the same 36 PNBs as considered in [1] and 5 more PNBs (so total 36+5=41) can be exploited to obtain a key recovery attack with key search complexity of $2^{247.2}$. The earlier results in this direction were 2^{251} [1] and 2^{250} [7].

2 Reversing one round of Salsa20

There are several state of the art works that studied biases in Salsa20. The most significant biases, by considering some output differential given some input difference, could be obtained after 3, 4 and 5 rounds [3, 4, 8, 1, 5, 7, 6] and these biases have been exploited suitably to obtain several cryptanalytic and non-randomness results in Salsa20 till 8 rounds. One of the most significant biases after 5 rounds has been reported in [4, Section 3.3]. They considered the differentials $\Delta_2^{(0)} = 0x00000100$, $\Delta_6^{(0)} = 0x00001000$, $\Delta_{14}^{(0)} = 0x80080000$. This shows an output bias after 5 rounds as follows:

$$\Pr(\Delta_{6,1}^{(5)} = 1 | \Delta_2^{(0)} = 0x00000100, \Delta_6^{(0)} = 0x00001000, \Delta_{14}^{(0)} = 0x80080000) \approx \frac{1}{2}(1 + 0.000772). \quad (3)$$

Note that this probability is less than $\frac{1}{2} - \frac{1}{2^{12}}$ and thus, the negative bias in absolute terms is more than $\frac{1}{2^{11}}$. The experimental data we present here is after 2^{30} runs which is enough to assess these biases with high confidence.

However, one observation here is very interesting, which shows that after one round, in certain cases, this generates a differential in only one bit, namely, $\Delta_{11}^{(1)} = 0x80000000$. At the same time, there are many cases where there exist other differentials in addition to $\Delta_{11}^{(1)} = 0x80000000$. This provides the main motivation of our study. We note that if one can start with some differentials such that after one round the differential is only at a single bit, then we will have sharper bias.

We look at the problem from another direction. Consider the input difference $\Delta_{11}^{(1)} = 0x80000000$. In this case,

$$\Pr(\Delta_{6,1}^{(5)} = 1 | \Delta_{11}^{(1)} = 0x80000000) \approx \frac{1}{2}(1 + 0.004274), \quad (4)$$

which is a much sharper bias. At the same time, one may note that if one comes back by reversing one round, this keeps both the states valid and the differentials are such that $\Delta_2^{(0)} = 0x00000100, \Delta_6^{(0)} = 0x00001000, \Delta_{14}^{(0)} = 0x8??80000)$ in most of the cases. We will get into detailed proof later and also the characterization of ? positions, but if one starts with such differentials, then we obtain a bias much sharper than what described in [4, Section 3.3].

In fact, noting the pattern by going back in reverse direction by one round, with this idea we could immediately improve the bias mentioned in (3) from [4] as

$$\Pr(\Delta_{6,1}^{(5)} = 1 | \Delta_2^{(0)} = 0x00000100, \Delta_6^{(0)} = 0x00001000, \Delta_{14}^{(0)} = 0x80180000) \approx \frac{1}{2}(1 + 0.001884). \quad (5)$$

We have included one additional bit of differential in the word 14 to obtain a better bias over [4].

Before proceeding further, let us describe the differential patterns. Given two states $X^{(r)}, X'^{(r)}$, we represent the differential state matrix as

$$\Delta^{(r)} = \begin{pmatrix} \Delta_0^{(r)} & \Delta_1^{(r)} & \Delta_2^{(r)} & \Delta_3^{(r)} \\ \Delta_4^{(r)} & \Delta_5^{(r)} & \Delta_6^{(r)} & \Delta_7^{(r)} \\ \Delta_8^{(r)} & \Delta_9^{(r)} & \Delta_{10}^{(r)} & \Delta_{11}^{(r)} \\ \Delta_{12}^{(r)} & \Delta_{13}^{(r)} & \Delta_{14}^{(r)} & \Delta_{15}^{(r)} \end{pmatrix},$$

where $\Delta_i^{(r)} = x_i^{(r)} \oplus x_i'^{(r)}$. As described in [4], we actually have the following scenario from the initial state to the state after application of one forward round.

$$\begin{pmatrix} 0\ 0\ 0x00000100\ 0 \\ 0\ 0\ 0x00001000\ 0 \\ 0\ 0 \quad 0 \quad 0 \\ 0\ 0\ 0x80080000\ 0 \end{pmatrix} \Rightarrow \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0x???????? & 0x???????? & 0x???????? & 0x80000000 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

Had one managed the unknown places to zero in the differential pattern after one round, the bias described in [4] would have been much sharper. In this direction, we look at this by reversing one round.

$$\begin{pmatrix} 0\ 0\ 0x00000100\ 0 \\ 0\ 0\ 0x00001000\ 0 \\ 0\ 0 \quad 0 \quad 0 \\ 0\ 0\ 0x???80000\ 0 \end{pmatrix} \Leftarrow \begin{pmatrix} 0\ 0\ 0 \quad 0 \\ 0\ 0\ 0 \quad 0 \\ 0\ 0\ 0\ 0x80000000 \\ 0\ 0\ 0 \quad 0 \end{pmatrix}.$$

Thus one important question is to characterize the bits in the 14-th word. Next, we present the main theoretical result in this direction.

Theorem 1. *Let $X^{(0)}$ be an arbitrary valid initial state. Suppose, after the first round, we modify $X^{(1)} = \mathsf{round}(X^{(0)})$ to $X'^{(1)}$ with the differential $\Delta_{11,31}^{(1)}$. Then $\mathsf{reverseround}(X'^{(1)})$ yields a valid initial state $X'^{(0)}$ with the differential*

$$\Delta^{(0)} = \begin{pmatrix} 0\ 0\ 0x00000100\ 0 \\ 0\ 0\ 0x00001000\ 0 \\ 0\ 0 \quad 0 \quad 0 \\ 0\ 0\ 0x???80000\ 0 \end{pmatrix}.$$

Proof. The first step in $\mathsf{reverseround}(X^{(1)})$ is $\mathsf{transpose}(X^{(1)})$, which changes the differential $\Delta_{11,31}^{(1)}$ to the differential $\Delta_{14,31}^{(1)}$. In the next step, reverse of quarterround would be applied to each column independently. Since position 14 appears in the 3rd column, let us focus on the effect of the inverse of quarterround on this column. All other columns would have 0 differentials after the inverse. From Equations (2), we can write the relevant equations as follows.

$$\left.\begin{aligned} x_{10}^{(0)} &= x_{10}^{(1)} \oplus ((x_2^{(1)} + x_6^{(1)}) \lll 18), \\ x_6^{(0)} &= x_6^{(1)} \oplus ((x_{14}^{(1)} + x_2^{(1)}) \lll 13), \\ x_2^{(0)} &= x_2^{(1)} \oplus ((x_{10}^{(0)} + x_{14}^{(1)}) \lll 9), \\ x_{14}^{(0)} &= x_{14}^{(1)} \oplus ((x_6^{(0)} + x_{10}^{(0)}) \lll 7). \end{aligned}\right\} \tag{6}$$

The same equations hold for x' as well. Now, let us analyze the differentials.

In the first equation, all the words of $x^{(1)}$ and $x'^{(1)}$ are identical. Hence, $\Delta_{10}^{(0)} = 0$. This proves that the state $X'^{(0)}$ is a valid initial state.

In the second equation, $x^{(1)}$ and $x'^{(1)}$ differ only in the MSB of the word 14, which causes a differential only in the MSB of the sum. After 13-bit left rotation, this differential moves to the bit position 12 of the term to the right of '$\oplus$'. Hence, $\Delta_6^{(0)} = 0x00001000$.

In the third equation, again $x^{(1)}$ and $x'^{(1)}$ differ only in the 31st bit of the word 14, which causes a differential only in the 31st bit of the sum. After 9-bit left rotation, this differential moves to the bit position 8 of the term to the right of '$\oplus$'. Hence, $\Delta_2^{(0)} = 0x00000100$.

In the fourth equation, the sum will have a differential in the bit position 12 due to $\Delta_{6,12}^{(0)} = 1$. There may be differentials to the left of the bit position 12, due to carry propagation. After 7-bit left rotation, the differential in the bit position 12 moves to the bit position 19 of the term to the right of '$\oplus$'. Hence, $\Delta_{14}^{(0)} = 0x???80000$. □

The value of '???' in $\Delta_{14}^{(0)}$ depends on $x_6^{(0)}$, which is nothing but the nonce v_0. Again, both $v_0 = x_6^{(0)}$ and $v_0' = x_6'^{(0)}$ (that differ in bit position 12 only) is added to the same constant $c_2 = x_{10}^{(0)} = x_{10}'^{(0)} =$ 0x79622d36, which has 3 consecutive 0s in the bit positions 14, 15, 16 and again in the bit positions 18, 19, 20 (these positions move to the left by 7 positions after the rotations). So in most cases, the effect of the carry is absorbed before it reaches from position 19 (i.e., 12 rotated by 7 positions) to position 31. Thus, the most frequent value of $\Delta_{14}^{(0)}$ is observed to be 0x8??80000.

Though there are multiple possible values of $\Delta_{14}^{(0)}$ for $\Delta_{11}^{(1)} = 0x80000000$, the value of $\Delta_{14}^{(0)}$ is uniquely determined for a given nonce. We can explicitly write

$$\Delta_{14}^{(0)} = 0x80000000 \oplus [(c_2 + v_0) \lll 7] \oplus [(c_2 + (v_0 \oplus 0x00001000)) \lll 7]. \quad (7)$$

With all these constraints on the initial state, one can obtain the bias as described in (4).

Let us discuss another example here. We consider the $\mathcal{ID}$ at the 31st bit of the 6th word and the $\mathcal{OD}$ at the 1st bit of the 1st word. By Theorem 2, once $\Delta_{6,31}^{(1)}$ is applied to a valid state after one round, the resulting state when reversed, yields another valid initial state. Similar to Theorem 1, one can show that the differential matrix goes through the following transition.

$$\begin{pmatrix} 0 & 0x00001000 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0x???80000 & 0 & 0 \\ 0 & 0x00000100 & 0 & 0 \end{pmatrix} \Leftarrow \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0x80000000 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

Similar to Equation (7), the value of $\Delta_9^{(0)}$ is uniquely determined for a given key as follows

$$\Delta_9^{(0)} = 0x80000000 \oplus [(c_1 + k_0) \lll 7] \oplus [(c_1 + (k_0 \oplus 0x00001000)) \lll 7].$$

Let S_d be the set of such valid input differences. We then map four-round biases with the $\mathcal{ID}$-$\mathcal{OD}$ pair $(\Delta^{(0)}_{6,31}, \Delta^{(4)}_{1,1})$ to five-round biases with the $\mathcal{ID}$-$\mathcal{OD}$ pair $(\Delta^{(0)}, \Delta^{(5)}_{1,1})$, where $\Delta^{(0)} \in S_d$. In this case, $\frac{1}{2}(1+\epsilon_d) = 0.501552$ and so $\epsilon_d = 0.003104$.

If we introduce an arbitrary differential after one round to convert $X^{(1)}$ to $X'^{(1)}$, then reversing $X'^{(1)}$ does not always yield a valid initial state. In the next result, that is a generalization of Theorem 1, we study the positions of those differentials.

Theorem 2. *Let $X^{(0)}$ be an arbitrary valid initial state. Suppose, after the first round, we complement any combination of the 128 bits of the words x_1, x_6, x_{11} and x_{12} of $X^{(1)}$ to obtain $X'^{(1)}$. Then $X'^{(0)} = \mathsf{reverseround}(X'^{(1)})$ is always a valid initial state.*

This idea helps us in translating the biases after 4 rounds to that of 5 rounds by reversing one round. One should also note that the reversal of one round is independent of the exactly chosen values of the constants c_0, c_1, c_2, c_3 in Salsa20.

3 Revisiting PNBs: improved cryptanalysis of Salsa 20/8

The cryptanalysis on Salsa20 that we discuss now is a known plaintext-only attack. The 512-bit key stream of Salsa20/R is $X + X^{(R)}$. In the known plaintext model, $X + X^{(R)}$ is completely available to the attacker. However, the 256 key bits are not available, thus only the rest 256 bits of X are known. The motivation is to guess the 256 key bits of Salsa20/R with a key search complexity less than 2^{256}, the complexity for exhaustive search.

The concept of Probabilistic Neutral Bits (PNBs) was exploited against Salsa20 in [1] in this framework. Before describing our parameters for an improved attack complexity, we first provide a brief and simple description of the generic attack using PNBs on Salsa20/R. For a detailed understanding and formal definitions in this area, one may refer to [1, Section 3].

Suppose X, X' are two valid initial states with a given $\mathcal{ID}$ $\Delta^{(0)}_{i,j} = 1$, for which we observe a high bias ϵ_d in an $\mathcal{OD}$ $\Delta^{(r)}_{p,q}$ after $r < R$ Salsa rounds. Thus, $\Pr(\Delta^{(r)}_{p,q} = 1 | \Delta^{(0)}_{i,j} = 1) = \frac{1}{2}(1+\epsilon_d)$, where $\Delta^{(r)} = X^{(r)} \oplus X'^{(r)}$. The two keystream blocks after R rounds are given by $Z = X + X^{(R)}$ and $Z' = X' + X'^{(R)}$.

Suppose, in both X and X', we complement a particular key bit position k to yield the states $\overline{X}$ and $\overline{X'}$ respectively. Next, we reverse the states $Z - \overline{X}$ and $Z' - \overline{X'}$ by $R - r$ rounds to yield the states Y and Y' respectively. Let $\Gamma_{p,q} = Y_{p,q} \oplus Y'_{p,q}$. If the bias in the event $(\Delta^{(r)}_{p,q} \oplus \Gamma_{p,q} = 0 | \Delta^{(0)}_{i,j} = 1)$ is high, i.e., $\Delta^{(r)}_{p,q} = \Gamma_{p,q}$ with high probability, given the $\mathcal{ID}$, then we call the key bit k a *probabilistic neutral bit* (PNB). If $\Pr(\Delta^{(r)}_{p,q} \oplus \Gamma_{p,q} = 0 | \Delta^{(0)}_{i,j} = 1) = \frac{1}{2}(1+\gamma_k)$, then γ_k is called the *neutrality measure* of the key bit. One should run this experiment for each key bit several times over randomly chosen nonces and counters and we experiment this for 2^{24} samples corresponding to each key bit. By repeating this

for all the 256 key bits, a subset of the key bits is identified, which are called the PNBs. Typically, a threshold probability $\frac{1}{2}(1+\gamma)$ is chosen to filter the PNBs. In other words, if $\gamma_k \geq \gamma$, then the key bit k is included in the set of the PNBs. Suppose the size of this subset is n and therefore the number of non-PNB bits is $m = 256 - n$. The main idea behind the key recovery is to search these two sets separately.

After the set of PNBs is determined, the actual attack considers search over the key bits which are not PNBs and by considering a distinguisher it is possible to determine when the correct keys can be identified. The attack algorithm is clearly described in [1]. However, for practical simulations, it is not possible to complete this attack as the complexity is very high. Nevertheless, we need to obtain certain biases properly to estimate the complexity of the attack and that can be achieved by trying out the following.

While studying the PNBs, in X and X', we complemented a particular key bit position k to yield the states $\overline{X}$ and $\overline{X'}$ respectively. However, during the key search, we assign random values to all the PNBs and keep the correct values of the keys in other places. That is, we assign correct key values to the m non-PNB key bits and assign random binary values to the n PNB key bits in both X and X' to yield the states $\hat{X}$ and $\hat{X}'$ respectively. Then we reverse the states $Z - \hat{X}$ and $Z' - \hat{X}'$ by $R - r$ rounds to yield the states $\hat{Y}$ and $\hat{Y}'$ respectively. Let $\hat{\Gamma}_{p,q} = \hat{Y}_{p,q} \oplus \hat{Y}'_{p,q}$ and $\Pr(\hat{\Gamma}_{p,q} = 1 | \Delta^{(0)}_{i,j} = 1) = \frac{1}{2}(1+\hat{\epsilon})$. A higher value of $\hat{\epsilon}$ means that the PNBs have been chosen properly and even without knowing the n PNBs it is possible to identify the rest of the key bits which are not PNBs.

Next, we assign random binary values to all the 256 key bits in both X and X' to yield the states $\widetilde{X}$ and $\widetilde{X}'$ respectively. Then we reverse the states $Z - \widetilde{X}$ and $Z' - \widetilde{X}'$ by $R - r$ rounds to yield the states $\widetilde{Y}$ and $\widetilde{Y}'$ respectively. Let $\widetilde{\Gamma}_{p,q} = \widetilde{Y}_{p,q} \oplus \widetilde{Y}'_{p,q}$ and $\Pr(\widetilde{\Gamma}_{p,q} = 1 | \Delta^{(0)}_{i,j} = 1) = \frac{1}{2}(1+\widetilde{\epsilon})$.

In actual key recovery attack, if the biases $\hat{\epsilon}$ and $\widetilde{\epsilon}$ can be efficiently distinguished, i.e., if the gap between the biases is significant with $\widetilde{\epsilon} \approx 0$ (as it corresponds to a random event and should not have any bias), then we can conclude that the assignment $\hat{X}$ yields the correct values for the non-PNB bits. This is what that needs to be experimented while choosing the set of PNBs. This also provides the estimate of $\hat{\epsilon}$ that will be involved in obtaining the complexity. To follow the same notation as in [1], we denote $\epsilon = \hat{\epsilon}$ in following discussion.

In [1], the bias in the event $(\hat{\Gamma}_{p,q} = \Delta^{(r)}_{p,q})$ is denoted by ϵ_a. In [1], the estimation of this bias is as follows. The key is fixed and one can vary the nonces and the counters to calculate one ϵ_a. Then one considers many randomly chosen keys to obtain a set of ϵ_a's and finally computes the median ϵ^*_a's from this set. Similarly, the median ϵ^*_d is estimated from the values of several ϵ_d's corresponding to different keys. Finally one can estimate ϵ^* as the median value of ϵ's. The idea of using median is that, one can guarantee that the estimated probabilities will work for at least half of the keys. We will also use the same measure for our experiments.

Following [1], if the number of samples used is N and if the probability of false alarm is $P_{fa} = 2^{-\alpha}$, the complexity of the attack is then given by

$2^m(N + 2^n P_{fa}) = 2^m N + 2^{256-\alpha}$, where the required number of samples is $N \approx \left(\frac{\sqrt{\alpha \log 4}+3\sqrt{1-(\epsilon^*)^2}}{\epsilon^*}\right)^2$, for probability of non-detection $P_{nd} = 1.3 \times 10^{-3}$.

3.1 Revisiting the complexity of attack in [1]

In [1], for $R = 8$, for the forward rounds, $r = 4$ and the $\mathcal{ID}$-$\mathcal{OD}$ pair had been $(\Delta^{(0)}_{7,31}, \Delta^{(4)}_{1,14})$. In this case, $\frac{1}{2}(1+\epsilon^*_d) = 0.5657$, i.e., $\epsilon^*_d = 0.1314$. Taking $\gamma = 0.12$, $n = 36$ PNBs have been reported in [1] and they estimated $\epsilon^* = 0.00015$ as the median bias over all keys.

In this case, $m = 256 - 36 = 220$. The authors of [1] used the following result for estimating the complexity. Taking $\alpha = 8$, one obtains $N \approx 2^{30.73} \approx 2^{31}$ and thus, the total complexity becomes $2^{220}(2^{31} + 2^{36} \cdot 2^{-8})$, which is approximately 2^{251} [1]. In [7], this complexity has been improved by additionally considering two-step Column Chaining Distinguishers (CCD) that provides the reduced complexity of 2^{250}.

However, we have got much improved results than [1] with our experiments. Note that the exact experimental figures related to the number of samples are not available in the paper [1]. We have checked with total 2^{36} samples as follows. We take 2^{10} randomly chosen keys and for each of them we take 2^{26} randomly chosen counters and nonces. For each of these keys, we take the average data ϵ and then we consider the median of those ϵ's which comes to be $\epsilon^* = 0.00060$. Surprisingly, we obtain 4 times more median bias than what observed in [1] for both ϵ^*_a and ϵ^*. With this estimate, keeping all the other parameters same, the complexity of [1] can be revised as follows. In this case, we obtain $N \approx 2^{26.73}$ and thus, the total complexity becomes $2^{220}(2^{26.73} + 2^{36} \cdot 2^{-8}) \approx 2^{246.73} + 2^{248}$.

We can then reduce the complexity further by playing around with α. The key idea is that the larger of the terms $2^m N$ and $2^{256-\alpha}$ dominates the complexity expression. We can take $\alpha = 12.82$ and in that case we obtain the complexity as $2^{220}(2^{27.11} + 2^{23.18}) \approx 2^{220} \cdot 2^{27.2} = 2^{247.2}$. Note that during this optimization process, we have kept P_{nd} fixed at 1.3×10^{-3}, same as in [1].

3.2 Adding more PNBs

For our study, we look at the PNBs more carefully. We use the same $\mathcal{ID}$-$\mathcal{OD}$ pair to be $(\Delta^{(0)}_{7,31}, \Delta^{(4)}_{1,14})$. Lowering γ, one can find many more PNBs. However, not all of them may be good candidates, as the bias ϵ^* gets reduced too. Taking $\gamma = 0.0488$ (but we do not consider all the key bits having $\gamma_k \geq \gamma$ and only consider them selectively; we describe this in detail little later), and 2^{24} samples for each key bit, we found the optimized list of PNBs to include 5 additional key bits other than the 36 key bits mentioned in [1]. Thus, in our analysis, $n = 41$ and $m = 256 - 41 = 215$. After the PNBs are identified, we used 2^{43} samples. These are 2^7 different keys and for each key we take an average of 2^{36} randomly chosen counters and nonces. Note that in this case we need more samples than as in Section 3.1. This is because in this case the biases are much less. Our

experiment shows that the median $\epsilon^* = 0.000106$. For $\alpha = 12.82$, we obtain the complexity as $2^{215}(2^{32.11} + 2^{28.18}) \approx 2^{215} \cdot 2^{32.2} = 2^{247.2}$. This is similar to the complexity obtained in Section 3.1.

Using our idea of biases after 5 rounds and coming back 4 rounds from the 9th round, one can exploit similar ideas of PNBs as in Section 3 for obtaining non-randomness in related-key scenario of Salsa20/9. However, we could only obtain a complexity slightly less than the exhaustive search and one may need to explore it further towards significant results. Finally, it should be mentioned that our works, as well as all the existing results in Salsa20 cryptanalysis, are on reduced rounds and they do not pose any security threat against Salsa20/12, which is in the eStream software portfolio.

Acknowledgment The third author has been supported in part by the Research Council KU Leuven: a senior postdoctoral scholarship SF/14/010 linked to the GOA TENSE (GOA/11/007).

References

1. J. -P. Aumasson, S. Fischer, S. Khazaei, W. Meier, and C. Rechberger. New Features of Latin Dances: Analysis of Salsa, ChaCha, and Rumba. FSE 2008, LNCS 5086, pp. 470-488, Springer.
2. D. J. Bernstein. Snuffle 2005: the Salsa20 encryption function `http://cr.yp.to/snuffle.html`
3. P. Crowley. Truncated differential cryptanalysis of five rounds of Salsa20. SASC 2006. `http://www.ciphergoth.org/crypto/salsa20/`
4. S. Fischer, W. Meier, C. Berbain, J. -F. Biasse, M. J. B. Robshaw. Non-randomness in eSTREAM candidates Salsa20 and TSC-4. INDOCRYPT 2006. LNCS 4329, pp. 2-16, Springer.
5. T. Ishiguro, S. Kiyomoto, and Y. Miyake. Latin Dances Revisited: New Analytic Results of Salsa20 and ChaCha. ICICS 2011, LNCS 7043, pp. 255-266, Springer. See corrections in `http://eprint.iacr.org/2012/065`
6. N. Mouha and B. Preneel. Towards Finding Optimal Differential Characteristics for ARX: Application to Salsa20. Cryptology ePrint Archive: Report 2013/328, `http://eprint.iacr.org/2013/328`
7. Z. Shi, B. Zhang, D. Feng and W. Wu. Improved Key Recovery Attacks on Reduced-Round Salsa20 and ChaCha. ICISC 2012, LNCS 7839, pp. 337-351, Springer.
8. Y. Tsunoo, T. Saito, H. Kubo, T. Suzaki, and Hiroki Nakashima. Differential Cryptanalysis of Salsa20/8. SASC 2007. `http://www.ecrypt.eu.org/stream/papersdir/2007/010.pdf`
9. The ECRYPT Stream Cipher Project. eSTREAM Portfolio of Stream Ciphers. `http://www.ecrypt.eu.org/stream/`