New Features of Latin Dances:
Analysis of Salsa, ChaCha, and Rumba

Jean-Philippe Aumasson[1], Simon Fischer[1], Shahram Khazaei[2], Willi Meier[1], and Christian Rechberger[3]

[1] FHNW, Windisch, Switzerland
[2] EPFL, Lausanne, Switzerland
[3] IAIK, Graz, Austria

Abstract. The stream cipher Salsa20 was introduced by Bernstein in 2005 as a candidate in the eSTREAM project, accompanied by the reduced versions Salsa20/8 and Salsa20/12. ChaCha is a variant of Salsa20 aiming at bringing better diffusion for similar performance. Variants of Salsa20 with up to 7 rounds (instead of 20) have been broken by differential cryptanalysis, while ChaCha has not been analyzed yet. We introduce a novel method for differential cryptanalysis of Salsa20 and ChaCha, inspired by correlation attacks and related to the notion of neutral bits. This is the first application of neutral bits in stream cipher cryptanalysis. It allows us to break the 256-bit version of Salsa20/8, to bring faster attacks on the 7-round variant, and to break 6- and 7-round ChaCha. In a second part, we analyze the compression function Rumba, built as the XOR of four Salsa20 instances and returning a 512-bit output. We find collision and preimage attacks for two simplified variants, then we discuss differential attacks on the original version, and exploit a high-probability differential to reduce complexity of collision search from 2^{256} to 2^{79} for 3-round Rumba. To prove the correctness of our approach we provide examples of collisions and near-collisions on simplified versions.

1 Introduction

Salsa20 [5] is a stream cipher introduced by Bernstein in 2005 as a candidate in the eSTREAM project [12], that has been selected in April 2007 for the third and ultimate phase of the competition. Three independent cryptanalyses were published [11,13,16], reporting key-recovery attacks for reduced versions with up to 7 rounds, while Salsa20 has a total of 20 rounds. Bernstein also submitted to public evaluation the 8- and 12-round variants Salsa20/8 and Salsa20/12 [6], though they are not formal eSTREAM candidates. Later he introduced ChaCha [3,4,8], a variant of Salsa20 that aims at bringing faster diffusion without slowing down encryption.

The compression function Rumba [7] was presented in 2007 in the context of a study of generalized birthday attacks [17] applied to incremental hashing [2], as the component of a hypothetical iterated hashing scheme. Rumba maps a 1536-bit value to a 512-bit (intermediate) digest, and Bernstein only conjectures collision resistance for this function, letting a further convenient operating mode provide extra security properties as pseudo-randomness.

Related Work. Variants of Salsa20 up to 7 rounds have been broken by differential cryptanalysis, exploiting a truncated differential over 3 or 4 rounds. The knowledge of less than 256 key bits can be sufficient for observing a difference in the state after three or four rounds, given a block of keystream of up to seven rounds of Salsa20. In 2005, Crowley [11] reported a 3-round differential, and built upon this an attack on Salsa20/5 within claimed 2^{165} trials. In 2006, Fischer et al. [13] exploited a 4-round differential to attack Salsa20/6 within claimed 2^{177} trials. In 2007, Tsunoo et al. [16] attacked Salsa20/7 within about 2^{190} trials, still exploiting a 4-round differential, and also claimed a break of Salsa20/8. However, the latter attack is effectively slower than brute force, cf. §3.5. Tsunoo et al. notably improve from previous attacks by reducing the guesses to certain bits—rather than guessing whole key words—using nonlinear approximation of integer addition. Eventually, no attack on ChaCha or Rumba has been published so far.

Contribution. We introduce a novel method for attacking Salsa20 and ChaCha (and potentially other ciphers) inspired from correlation attacks, and from the notion of neutral bit, introduced by Biham and Chen [9] for attacking SHA-0. More precisely, we use an empirical measure of the correlation between certain key bits of the state and the bias observed after working a few rounds backward, in order to split key bits into two subsets: the extremely relevant key bits to be subjected to an exhaustive search and filtered by observations of a biased output-difference value,and the less significant key bits ultimately determined by exhaustive search. To the best of our knowledge, this is the first time that neutral bits are used for the analysis of stream ciphers. Our results are summarized in Tab. 1. We present the first key-recovery attack for the 256-bit version of Salsa20/8, improve the previous attack on 7-round Salsa20 by a factor 2^{39}, and present attacks on ChaCha up to 7 rounds. The 128-bit versions are also investigated. In a second part, we first show collision and preimage attacks for simplified versions of Rumba, then we present a differential analysis of the original version using the methods of linearization and neutral bits: our main result is a collision attack for 3-round Rumba running in about 2^{79} trials (compared to 2^{256} with a birthday attack). We also give examples of near-collisions over three and four rounds.

Table 1. Complexity of the best attacks known, with success probability 1/2.

	Salsa20/7	Salsa20/8	ChaCha6	ChaCha7	Rumba3
Before	2^{190}	2^{255}	2^{255}	2^{255}	2^{256}
Now	2^{151}	2^{251}	2^{139}	2^{248}	2^{79}

Road Map. We first recall the definitions of Salsa20, ChaCha, and Rumba in §2, then §3 describes our attacks on Salsa20 and ChaCha, and §4 presents our cryptanalysis of Rumba. The appendices give the sets of constant values, and some parameters necessary to reproduce our attacks.

2 Specification of Primitives

In this section, we give a concise description of the stream ciphers Salsa20 and ChaCha, and of the compression function Rumba.

2.1 Salsa20

The stream cipher Salsa20 operates on 32-bit words, takes as input a 256-bit key $k = (k_0, k_1, \ldots, k_7)$ and a 64-bit nonce $v = (v_0, v_1)$, and produces a sequence of 512-bit keystream blocks. The i-th block is the output of the *Salsa20 function*, that takes as input the key, the nonce, and a 64-bit counter $t = (t_0, t_1)$ corresponding to the integer i. This function acts on the 4×4 matrix of 32-bit words written as

$$X = \begin{pmatrix} x_0 & x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 & x_7 \\ x_8 & x_9 & x_{10} & x_{11} \\ x_{12} & x_{13} & x_{14} & x_{15} \end{pmatrix} = \begin{pmatrix} c_0 & k_0 & k_1 & k_2 \\ k_3 & c_1 & v_0 & v_1 \\ t_0 & t_1 & c_2 & k_4 \\ k_5 & k_6 & k_7 & c_3 \end{pmatrix} . \quad (1)$$

The c_i's are predefined constants (see Appendix A). There is also a mode for a 128-bit key k', where the 256 key bits in the matrix are filled with $k = k' \| k'$. If not mentioned otherwise, we focus on the 256-bit version. A keystream block Z is then defined as

$$Z = X + X^{20} , \quad (2)$$

where "+" symbolizes wordwise integer addition, and where $X^r = \mathsf{Round}^r(X)$ with the round function Round of Salsa20. The round function is based on the following nonlinear operation (also called the $\mathsf{quarterround}$ function), which transforms a vector (x_0, x_1, x_2, x_3) to (z_0, z_1, z_2, z_3) by sequentially computing

$$\begin{aligned} z_1 &= x_1 \oplus \big[(x_3 + x_0) \lll 7\big] \\ z_2 &= x_2 \oplus \big[(x_0 + z_1) \lll 9\big] \\ z_3 &= x_3 \oplus \big[(z_1 + z_2) \lll 13\big] \\ z_0 &= x_0 \oplus \big[(z_2 + z_3) \lll 18\big] . \end{aligned} \quad (3)$$

In odd numbers of rounds (which are called $\mathsf{columnrounds}$ in the original specification of Salsa20), the nonlinear operation is applied to the columns (x_0, x_4, x_8, x_{12}), (x_5, x_9, x_{13}, x_1), $(x_{10}, x_{14}, x_2, x_6)$, $(x_{15}, x_3, x_7, x_{11})$. In even numbers of rounds (which are also called the $\mathsf{rowrounds}$), the nonlinear operation is applied to the rows (x_0, x_1, x_2, x_3), (x_5, x_6, x_7, x_4), $(x_{10}, x_{11}, x_8, x_9)$, $(x_{15}, x_{12}, x_{13}, x_{14})$. We write Salsa20/R for R-round variants, i.e. with $Z = X + X^R$. Note that the r-round inverse $X^{-r} = \mathsf{Round}^{-r}(X)$ is defined differently whether it inverts after an odd or and even number of rounds.

2.2 ChaCha

ChaCha is similar to Salsa20 with the following modifications

1. The input words are placed differently in the initial matrix:

$$X = \begin{pmatrix} x_0 & x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 & x_7 \\ x_8 & x_9 & x_{10} & x_{11} \\ x_{12} & x_{13} & x_{14} & x_{15} \end{pmatrix} = \begin{pmatrix} c_0 & c_1 & c_2 & c_3 \\ k_0 & k_1 & k_2 & k_3 \\ k_4 & k_5 & k_6 & k_7 \\ t_0 & t_1 & v_0 & v_1 \end{pmatrix} . \quad (4)$$

2. The nonlinear operation of Round transforms a vector (x_0, x_1, x_2, x_3) to (z_0, z_1, z_2, z_3) by sequentially computing

$$\begin{array}{ll} b_0 = x_0 + x_1, & b_3 = (x_3 \oplus b_0) \lll 16 \\ b_2 = x_2 + b_3, & b_1 = (x_1 \oplus b_2) \lll 12 \\ z_0 = b_0 + b_1, & z_3 = (b_3 \oplus z_0) \lll 8 \\ z_2 = b_2 + z_3, & z_1 = (b_1 \oplus z_2) \lll 7 . \end{array} \quad (5)$$

3. The round function is defined differently: in odd numbers of rounds, the nonlinear operation is applied to the columns (x_0, x_4, x_8, x_{12}), (x_1, x_5, x_9, x_{13}), $(x_2, x_6, x_{10}, x_{14})$, $(x_3, x_7, x_{11}, x_{15})$, and in even numbers of rounds, the nonlinear operation is applied to the diagonals $(x_0, x_5, x_{10}, x_{15})$, $(x_1, x_6, x_{11}, x_{12})$, (x_2, x_7, x_8, x_{13}), (x_3, x_4, x_9, x_{14}), see [3] for details.

As for Salsa20, the round function of ChaCha is trivially invertible. R-round variants are denoted ChaChaR. The core function of ChaCha suggests that *"the big advantage of ChaCha over Salsa20 is the diffusion, which at least at first glance looks considerably faster"* [4].

2.3 Rumba

Rumba is a *compression function* built on Salsa20, mapping a 1536-bit message to a 512-bit value. The input M is parsed as four 384-bit chunks $M_0, \ldots, M_3$, and Rumba's output is

$$\begin{aligned} \mathrm{Rumba}(M) &= F_0(M_0) \oplus F_1(M_1) \oplus F_2(M_2) \oplus F_3(M_3) \\ &= (X_0 + X_0^{20}) \oplus (X_1 + X_1^{20}) \oplus (X_2 + X_2^{20}) \oplus (X_3 + X_3^{20}) , \end{aligned} \quad (6)$$

where each F_i is an instance of the function Salsa20 with distinct diagonal constants (see Appendix A). The 384-bit input chunk M_i along with the corresponding 128-bit diagonal constants are then used to fill up the corresponding input matrix X_i. A single word j of X_i is denoted $x_{i,j}$. Note that the functions F_i include the feedforward operation of Salsa20. RumbaR stands for R-round variant.

3 Differential Analysis of Salsa20 and ChaCha

This section introduces differential attacks based on a new technique called *probabilistic neutral bits* (shortcut PNB's). To apply it to Salsa20 and ChaCha, we first identify optimal choices of truncated differentials, then we describe a general framework for probabilistic backwards computation, and introduce the notion of PNB's along with a method to find them. Then, we outline the overall attack, and present concrete attacks for Salsa20/7, Salsa20/8, ChaCha6, and ChaCha7. Eventually, we discuss our attack scenarios and possibilities of improvements.

3.1 Choosing a Differential

Let x_i be the i-th word of the matrix-state X, and x'_i an associated word with the difference $\Delta_i^0 = x_i \oplus x'_i$. The j-th bit of x_i is denoted $[x_i]_j$. We use (truncated) input/output differentials for the input X, with a single-bit input-difference $[\Delta_i^0]_j = 1$ in the nonce, and consider a single-bit output-difference $[\Delta_p^r]_q$ after r rounds in X^r. Such a differential is denoted $([\Delta_p^r]_q \mid [\Delta_i^0]_j)$. For a fixed key, the bias ε_d of the output-difference is defined by

$$\Pr_{v,t}\{[\Delta_p^r]_q = 1 \mid [\Delta_i^0]_j\} = \frac{1}{2}(1+\varepsilon_d) \ , \qquad (7)$$

where the probability holds over all nonces and counters. Furthermore, considering key as a random variable, we denote the median value of of ε_d by $\varepsilon_d^\star$. Hence, for half of the keys this differential will have a bias of at least $\varepsilon_d^\star$. Note that our statistical model considers a (uniformly) random value of the counter. In the following, we use the shortcuts $\mathcal{ID}$ and $\mathcal{OD}$ for input- and output-difference.

3.2 Probabilistic Backwards Computation

In the following, assume that the differential $([\Delta_p^r]_q \mid [\Delta_i^0]_j)$ of bias ε_d is fixed, and the corresponding outputs Z and Z' are observed for nonce v, counter t and key k. Having k, v and t, one can invert the operations in $Z = X + X^R$ and $Z' = X' + (X')^R$ in order to access to the r-round forward differential (with $r < R$) from the backward direction thanks to the relations $X^r = (Z-X)^{r-R}$ and $(X')^r = (Z' - X')^{r-R}$. More specifically, define $f(k, v, t, Z, Z')$ as the function which returns the q-th LSB of the word number p of the matrix $(Z-X)^{r-R} \oplus (Z' - X')^{r-R}$, hence $f(k, v, t, Z, Z') = [\Delta_p^r]_q$. Given enough output block pairs with the presumed difference in the input, one can verify the correctness of a guessed candidate $\hat{k}$ for the key k by evaluating the bias of the function f. More precisely, we have $\Pr\{f(\hat{k}, v, t, Z, Z') = 1\} = \frac{1}{2}(1+\varepsilon_d)$ conditioned on $\hat{k} = k$, whereas for (almost all) $\hat{k} \neq k$ we expect f be unbiased i.e. $\Pr\{f(\hat{k}, v, t, Z, Z') = 1\} = \frac{1}{2}$. The classical way of finding the correct key requires exhaustive search over all possible 2^{256} guesses $\hat{k}$. However, we can search only over a subkey of $m = 256 - n$ bits, provided that an approximation g of f which effectively depends on m key bits is available. More formally, let $\bar{k}$ correspond to the subkey of m bits of the key k and let f be correlated to g with bias ε_a i.e.:

$$\Pr_{v,t}\{f(k,v,t,Z,Z') = g(\bar{k},v,t,Z,Z')\} = \frac{1}{2}(1+\varepsilon_a) \ . \tag{8}$$

Note that deterministic backwards computation (i.e. $\bar{k} = k$ with $f = g$) is a special case with $\varepsilon_a = 1$. Denote the bias of g by ε, i.e. $\Pr\{g(\bar{k},v,t,Z,Z') = 1\} = \frac{1}{2}(1+\varepsilon)$. Under some reasonable independency assumptions, the equality $\varepsilon = \varepsilon_d \cdot \varepsilon_a$ holds. Again, we denote $\varepsilon^\star$ the median bias over all keys (we verified in experiments that $\varepsilon^\star$ can be well estimated by the median of $\varepsilon_d \cdot \varepsilon_a$). Here, one can verify the correctness of a guessed candidate $\hat{\bar{k}}$ for the subkey $\bar{k}$ by evaluating the bias of the function g based on the fact that we have $\Pr\{g(\hat{\bar{k}},v,t,Z,Z') = 1\} = \frac{1}{2}(1+\varepsilon)$ for $\hat{\bar{k}} = \bar{k}$, whereas $\Pr\{g(\hat{\bar{k}},v,t,Z,Z') = 1\} = \frac{1}{2}$ for $\hat{\bar{k}} \neq \bar{k}$. This way we are facing an exhaustive search over 2^m subkey candidates opposed to the original 2^{256} key candidates which can potentially lead to a faster attack. We stress that the price which we pay is a higher data complexity, see §3.4 for more details.

3.3 Probabilistic Neutral Bits

Our new view of the problem, described in §3.2, demands efficient ways for finding suitable approximations $g(\bar{k},W)$ of a given function $f(k,W)$ where W is a known parameter; in our case, it is $W = (v,t,Z,Z')$. In a probabilistic model one can consider W as a uniformly distributed random variable. Finding such approximations in general is an interesting open problem. In this section we introduce a generalized concept of neutral bits [9] called *probabilistic neutral bits* (PNB's). This will help us to find suitable approximations in the case that the Boolean function f does not properly mix its input bits. Generally speaking, PNB's allows us to divide the key bits into two groups: *significant key bits* (of size m) and *non-significant key bits* (of size n). In order to identify these two sets we focus on the amount of influence which each bit of the key has on the output of f. Here is a formal definition of a suitable measure:

Definition 1. *The neutrality measure of the key bit k_i with respect to the function $f(k,W)$ is defined as γ_i, where $\Pr = \frac{1}{2}(1+\gamma_i)$ is the probability (over all k and W) that complementing the key bit k_i does not change the output of $f(k,W)$.*

Singular cases of the neutrality measure are:

- $\gamma_i = 1$: $f(k,W)$ does not depend on i-th key bit (i.e. it is a neutral bit).
- $\gamma_i = 0$: $f(k,W)$ is statistically independent of the i-th key bit (i.e. it is a significant bit).
- $\gamma_i = -1$: $f(k,W)$ linearly depends on the i-th key bit.

In practice, we set a threshold γ and put all key bits with $\gamma_i \leq \gamma$ in the set of significant key bits. The less significant key bits we get, the faster the attack will be, provided that the bias ε_a (see Eq. 8) remains non-negligible. Having found significant and non-significant key bits, we simply let $\bar{k}$ be the significant key bits and define $g(\bar{k}, W)$ as $f(k, W)$ with non-significant key bits being set to a fixed value (e.g. all zero). Note that, contrary to the mutual interaction between neutral bits in [9], here we have directly combined several PNB's without altering their probabilistic quality. This can be justified as the bias ε_a smoothly decreases while we increase the threshold γ.

Remark 1. Tsunoo et al. [16] used nonlinear approximations of integer addition to identify the dependency of key bits, whereas the independent key bits—with respect to nonlinear approximation of some order—are fixed. This can be seen as a special case of our method.

3.4 Complexity Estimation

Here we sketch the full attack described in the previous subsections, then study its computational cost. The attack is split up into a precomputation stage, and a stage of effective attack; note that precomputation is not specific to a key or a counter.

Precomputation

1. Find a high-probability r-round differential with $\mathcal{ID}$ in the nonce or counter.
2. Choose a threshold γ.
3. Construct the function f defined in §3.2.
4. Empirically estimate the neutrality measure γ_i of each key bit for f.
5. Put all those key bits with $\gamma_i < \gamma$ in the significant key bits set (of size $m = 256 - n$).
6. Construct the function g using f by assigning a fixed value to the non-significant key bits, see §3.2 and §3.3.
7. Estimate the median bias $\varepsilon^\star$ by empirically measuring the bias of g using many randomly chosen keys, see §3.2.
8. Estimate the data and time complexity of the attack, see the following.

The cost of this precomputation phase is negligible compared to the effective attack (to be explained later). The r-round differential and the threshold γ should be chosen such that the resulting time complexity is optimal. This will be addressed later in this section. At step 1, we require the difference to be in the nonce or in the counter, assuming that both variables are user-controlled inputs. We exclude a difference in the key in a *related-key* attack due to the disputable attack model. Previous attacks on Salsa20 use the rough estimate of $N = \varepsilon^{-2}$ samples, in order to identify the correct subkey in a large search space. However this estimate is incorrect: this is the number of samples necessary to identify a *single* random unknown bit from either a uniform source or from a non-uniform source with ε, which is a different problem of hypothesis testing. In our case, we have a set of 2^m sequences of random variables with $2^m - 1$ of them verifying the null hypothesis H_0, and a single one verifying the alternative hypothesis H_1. For a realization a of the corresponding random variable A, the decision rule $\mathcal{D}(a) = i$ to accept H_i can lead to two types of errors:

1. Non-detection: $\mathcal{D}(a) = 0$ and $A \in H_1$. The probability of this event is p_{nd}.
2. False alarm: $\mathcal{D}(a) = 1$ and $A \in H_0$. The probability of this event is p_{fa}.

The Neyman-Pearson decision theory gives results to estimate the number of samples N required to get some bounds on the probabilities. It can be shown that

$$N \approx \left(\frac{\sqrt{\alpha \log 4} + 3\sqrt{1-\varepsilon^2}}{\varepsilon} \right)^2 \tag{9}$$

samples suffices to achieve $p_{\text{nd}} = 1.3 \times 10^{-3}$ and $p_{\text{fa}} = 2^{-\alpha}$. Calculus details and the construction of the optimal distinguisher can be found in [15], see also [1] for more general results on distributions' distinguishability. In our case the value of ε is key dependent, so we use the median bias $\varepsilon^\star$ in place of ε in Eq. 9, resulting in a success probability of at least $\frac{1}{2}(1-p_{\text{nd}}) \approx \frac{1}{2}$ for our attack. Having determined the required number of samples N and the optimal distinguisher, we can now present the effective (or online) attack.

Effective attack

1. For an unknown key, collect N pairs of keystream blocks where each pair is produced by states with a random nonce and counter (satisfying the relevant $\mathcal{ID}$).
2. For each choice of the subkey (i.e. the m significant key bits) do:
   (a) Compute the bias of g using the N keystream block pairs.
   (b) If the optimal distinguisher legitimates the subkeys candidate as a (possibly) correct one, perform an additional exhaustive search over the n non-significant key bits in order to check the correctness of this filtered subkey and to find the non-significant key bits.
   (c) Stop if the right key is found, and output the recovered key.

Let us now discuss the time complexity of our attack. Step 2 is repeated for all 2^m subkey candidates. For each subkey, step (a) is always executed which has complexity[4] of N. However, the search part of step (b) is performed only with probability $p_{\text{fa}} = 2^{-\alpha}$ which brings an additional cost of 2^n in case a subkey passes the optimal distinguisher's filter. Therefore the complexity of step (b) is $2^n p_{\text{fa}}$, showing a total complexity of $2^m(N + 2^n p_{\text{fa}}) = 2^m N + 2^{256-\alpha}$ for the effective attack. In practice, α (and hence N) is chosen such that it minimizes $2^m N + 2^{256-\alpha}$. Note that the potential improvement from key ranking techniques is not considered here, see e.g. [14]. The data complexity of our attack is N keystream block pairs.

Remark 2. It is reasonable to assume that a false subkey, which is close to the correct subkey, may introduce a non-negligible bias. In general, this results in an increased value of p_{fa}. If many significant key bits have neutrality measure close to zero, then the increase is expected to be small, but the precise practical impact of this observation is unknown to the authors.

[4] More precisely the complexity is about $2(R-r)/RN$ times the required time for producing one keystream block.

3.5 Experimental Results

We used automatized search to identify optimal differentials for the reduced-round versions Salsa20/7, Salsa20/8, ChaCha6, and ChaCha7. This search is based on the following observation: The number n of PNB's for some fixed threshold γ mostly depends on the $\mathcal{OD}$, but not on the $\mathcal{ID}$. Consequently, for each of the 512 single-bit $\mathcal{OD}$'s, we can assign the $\mathcal{ID}$ with maximum bias ε_d, and estimate time complexity of the attack. Below we only present the differentials leading to the best attacks. The threshold γ is also an important parameter: Given a fixed differential, time complexity of the attack is minimal for some optimal value of γ. However, this optimum may be reached for quite small γ, such that n is large and $|\varepsilon_a^\star|$ small. We use at most 2^{24} random nonces and counters for each of the 2^{10} random keys, so we can only measure a bias of about $|\varepsilon_a^\star| > c \cdot 2^{-12}$ (where $c \approx 10$ for a reasonable estimation error). In our experiments, the optimum is not reached with these computational possibilities (see e.g. Tab. 2), and we note that the described complexities may be improved by choosing a smaller γ.

Attack on 256-bit Salsa20/7. We use the differential $([\Delta_1^4]_{14} \mid [\Delta_7^0]_{31})$ with $|\varepsilon_d^\star| = 0.131$. The $\mathcal{OD}$ is observed after working three rounds backward from a 7-round keystream block. To illustrate the role of the threshold γ, we present in Tab. 2 complexity estimates along with the number n of PNB's, the values of $|\varepsilon_d^\star|$ and $|\varepsilon^\star|$, and the optimal values of α for several threshold values. For $\gamma = 0.4$, the attack runs in time 2^{151} and data 2^{26}. The previous best attack in [16] required about 2^{190} trials and 2^{12} data.

Table 2. Different parameters for our attack on 256-bit Salsa20/7.

γ	n	$\lvert\varepsilon_a^\star\rvert$	$\lvert\varepsilon^\star\rvert$	α	Time	Data
1.00	39	1.000	0.1310	31	2^{230}	2^{13}
0.90	97	0.655	0.0860	88	2^{174}	2^{15}
0.80	103	0.482	0.0634	93	2^{169}	2^{16}
0.70	113	0.202	0.0265	101	2^{162}	2^{19}
0.60	124	0.049	0.0064	108	2^{155}	2^{23}
0.50	131	0.017	0.0022	112	2^{151}	2^{26}

Attack on 256-bit Salsa20/8. We use again the differential $([\Delta_1^4]_{14} \mid [\Delta_7^0]_{31})$ with $|\varepsilon_d^\star| = 0.131$. The $\mathcal{OD}$ is observed after working four rounds backward from an 8-round keystream block. For the threshold $\gamma = 0.12$ we find $n = 36$, $|\varepsilon_a^\star| = 0.0011$, and $|\varepsilon^\star| = 0.00015$. For $\alpha = 8$, this results in time 2^{251} and data 2^{31}. The list of PNB's is {26, 27, 28, 29, 30, 31, 71, 72, 120, 121, 122, 148, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 210, 211, 212, 224, 225, 242, 243, 244, 245, 246, 247}. Note that our attack reaches the same success probability and supports an identical degree of parallelism as brute force. The previous attack in [16] claims 2^{255} trials with data 2^{10} for success probability 44%, but exhaustive search succeeds with probability 50% within the same number of trials, with much less data and no additional computations. Therefore their attack does not constitute a break of Salsa20/8.

Attack on 128-bit Salsa20/7. Our attack can be adapted to the 128-bit version of Salsa20/7. With the differential $([\Delta_1^4]_{14} \mid [\Delta_7^0]_{31})$ and $\gamma = 0.4$, we find $n = 38$, $|\varepsilon_a^\star| = 0.045$, and $|\varepsilon^\star| = 0.0059$. For $\alpha = 21$, this breaks Salsa20/7 within 2^{111} time and 2^{21} data. Our attack fails to break 128-bit Salsa20/8 because of the insufficient number of PNB's.

Attack on 256-bit ChaCha6. We use the differential $([\Delta_{11}^3]_0 \mid [\Delta_{13}^0]_{13})$ with $|\varepsilon_d^\star| = 0.026$. The $\mathcal{OD}$ is observed after working three rounds backward from an 6-round keystream block. For the threshold $\gamma = 0.6$ we find $n = 147$, $|\varepsilon_a^\star| = 0.018$, and $|\varepsilon^\star| = 0.00048$. For $\alpha = 123$, this results in time 2^{139} and data 2^{30}.

Attack on 256-bit ChaCha7. We use again the differential $([\Delta_{11}^3]_0 \mid [\Delta_{13}^0]_{13})$ with $|\varepsilon_d^\star| = 0.026$. The $\mathcal{OD}$ is observed after working four rounds backward from an 7-round keystream block. For the threshold $\gamma = 0.5$ we find $n = 35$, $|\varepsilon_a^\star| = 0.023$, and $|\varepsilon^\star| = 0.00059$. For $\alpha = 11$, this results in time 2^{248} and data 2^{27}. The list of PNB's is $\{3, 6, 15, 16, 31, 35, 67, 68, 71, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 103, 104, 127, 136, 191, 223, 224, 225, 248, 249, 250, 251, 252, 253, 254, 255\}$.

Attack on 128-bit ChaCha6. Our attack can be adapted to the 128-bit version of ChaCha6. With the differential $([\Delta_{11}^3]_0 \mid [\Delta_{13}^0]_{13})$ and $\gamma = 0.5$, we find $n = 51$, $|\varepsilon_a^\star| = 0.013$, and $|\varepsilon^\star| = 0.00036$. For $\alpha = 26$, this breaks ChaCha6 within 2^{107} time and 2^{30} data. Our attack fails to break 128-bit ChaCha7.

3.6 Discussion

Our attack on reduced-round 256-bit Salsa20 exploits a 4-round differential, to break the 8-round cipher by working four rounds backward. For ChaCha, we use a 3-round differential to break 7 rounds. We made intensive experiments for observing a bias after going five rounds backwards from the guess of a subkey, in order to attack Salsa20/9 or ChaCha8, but without success. Four seems to be the highest number of rounds one can invert from a partial key guess, while still observing a non-negligible bias after inversion, and such that the overall cost improves from exhaustive key search. Can one hope to break further rounds by statistical cryptanalysis? We believe that it would require novel techniques and ideas, rather than the relatively simple XOR difference of 1-bit input and 1-bit output. For example, one might combine several biased $\mathcal{OD}$'s to reduce the data complexity, but this requires almost equal subsets of guessed bits; according to our experiments, this seems difficult to achieve. We also found some highly biased multibit differentials such as $([\Delta_1^4]_0 \oplus [\Delta_2^4]_9 \mid [\Delta_7^0]_{26})$ with bias $\varepsilon_d = -0.60$ for four rounds of Salsa20. However, exploiting multibit differentials, does not improve efficiency either. Note that an alternative approach to attack Salsa20/7 is to consider a 3-round biased differential, and observe it after going four rounds backward. This is however much more expensive than exploiting directly 4-round differentials. Unlike Salsa20, our exhaustive search showed no bias in 4-round ChaCha, be it with one, two, or three target output bits. This argues in favor of the faster diffusion of ChaCha. But surprisingly, when comparing the attacks on Salsa20/8 and ChaCha7, results suggest that after four rounds backward, key bits are more correlated with the target difference in ChaCha than in Salsa20. Nevertheless, ChaCha looks more trustful on the overall, since we could break up to seven ChaCha rounds against eight for Salsa20. For the variants with a 128-bit key, we can break up to seven Salsa20 rounds, and up to six ChaCha rounds.

4 Analysis of Rumba

This section describes our results for the compression function Rumba. Our goal is to efficiently find colliding pairs for R-round Rumba, i.e. input pairs (M, M') such that $\text{RumbaR}(M) \oplus \text{RumbaR}(M') = 0$. Note that, compared to our attacks on Salsa20 (where a single biased bit could be exploited in an attack), a collision attack targets all 512 bits (or a large subset of them for near-collisions).

4.1 Collisions and Preimages in Simplified Versions

We show here the weakness of two simplified versions of Rumba, respectively an iterated version with 2048-bit-input compression function, and the compression function without the final feedforward.

On the Role of Diagonal Constants. Rumba20 is fed with 1536 bits, copied in a 2048-bit state, whose remaining 512 bits are the diagonal constants. It is tempting to see these values as the IV of a derived iterated hash function, and use diagonal values as chaining variables. However, Bernstein implicitly warned against such a construction, when claiming that *"Rumba20 will take about twice as many cycles per eliminated byte as Salsa20 takes per encrypted byte"* [7]; indeed, the 1536-bit input should contain both the 512-bit chaining value and the 1024-bit message, and thus for a 1024-bit input the Salsa20 function is called four times (256 bits processed per call), whereas in Salsa20 it is called once for a 512-bit input. We confirm here that diagonal values should *not* be replaced by the chaining variables, by presenting a method for finding collisions within about $2^{128}/6$ trials, against 2^{256} with a birthday attack: Consider the following algorithm: pick an arbitrary 1536-bit message block M^0, then compute $\text{Rumba}(M^0) = H_0 \| H_1 \| H_2 \| H_3$, and repeat this until two distinct 128-bit chunks H_i and H_j are equal—say H_0 and H_1, corresponding to the diagonal constants of F_0 and F_1 in the next round; hence, these functions will be identical in the next round. A collision can then be obtained by choosing two distinct message blocks $M^1 = M^1_0 \| M^1_1 \| M^1_2 \| M^1_3$ and $(M')^1 = M^1_1 \| M^1_0 \| M^1_2 \| M^1_3$, or $M^1 = M^1_0 \| M^1_0 \| M^1_2 \| M^1_3$ and $(M')^1 = (M'_0)^1 \| (M'_0)^1 \| M^1_2 \| M^1_3$. How fast is this method? By the birthday paradox, the amount of trials for finding a suitable M^0 is about $2^{128}/6$ (here 6 is the number of distinct sets $\{i,j\} \subset \{0,\ldots,3\}$), while the construction of M^1 and $(M')^1$ is straightforward. Regarding the price-performance ratio, we do not have to store or sort a table, so the price is $2^{128}/6$—and this, for any potential filter function—while performance is much larger than one, because there are many collisions (one can choose 3 messages and 1 difference of 348 bits arbitrarily). This contrasts with the cost of 2^{256} for a serial attack on a 512-bit digest hash function.

On the Importance of Feedforward. In Davies-Meyer-based hash functions like MD5 or SHA-1, the final feedforward is an obvious requirement for one-wayness. In Rumba the feedforward is applied in each F_i, before an XOR of the four branches, and omitting this operation does not trivially lead to an inversion of the function, because of the incremental construction. However, as we will demonstrate, preimage resistance is not guaranteed with this setting. Let $F_i(M_i) = X_i^{20}$, $i = 0,\ldots,3$ and assume that we are given a 512-bit value H, and our goal is to find $M = (M_0, M_1, M_2, M_3)$ such that Rumba$(M) = H$. This can be achieved by choosing *random* blocks M_0, M_1, M_2, and set

$$Y = F_0(M_0) \oplus F_1(M_1) \oplus F_2(M_2) \oplus H \ . \tag{10}$$

We can find then the 512-bit state X_3^0 such that $Y = X_3^{20}$. If X_3^0 has the correct diagonal values (the 128-bit constant of F_3), we can extract M_3 from X_0^3 with respect to Rumba's definition. This randomized algorithm succeeds with probability 2^{-128}, since there are 128 constant bits in an initial state. Therefore, a preimage of an arbitrary digest can be found within about 2^{128} trials, against $2^{512/3}$ $(= 2^{512/(1+\log_2 4)})$ with the generalized birthday method.

4.2 Differential Attack

To obtain a collision for RumbaR, it is sufficient to find two messages M and M' such that

$$F_0(M_0) \oplus F_0(M_0') = F_2(M_2) \oplus F_2(M_2') \ , \tag{11}$$

with $M_0 \oplus M_0' = M_2 \oplus M_2'$, $M_1 = M_1'$ and $M_3 = M_3'$. The freedom in choosing M_1 and M_3 trivially allows to derive many other collisions (*multicollision*). We use the following notations for differentials: Let the initial states X_i and X_i' have the $\mathcal{ID}$ $\Delta_i^0 = X_i \oplus X_i'$ for $i = 0,\ldots,3$. After r rounds, the *observed* difference is denoted $\Delta_i^r = X_i^r \oplus (X_i')^r$, and the $\mathcal{OD}$ (without feedforward) becomes $\Delta_i^R = X_i^R \oplus (X_i')^R$. If feedforward is included in the $\mathcal{OD}$, we use the notation $\nabla_i^R = (X_i + X_i^R) \oplus (X_i' + (X_i')^R)$. With this notation, Eq. 11 becomes $\nabla_0^R = \nabla_2^R$, and if the feedforward operation is ignored in the F_i's, then Eq. 11 simplifies to $\Delta_0^R = \Delta_2^R$. To find messages satisfying Eq. 11, we use an R-round differential path of high-probability, with intermediate *target* difference δ^r after r rounds, $0 \le r \le R$. Note that the differential is applicable for both F_0 and F_2, thus we do not have to subscript the target difference. The probability that a random message pair with $\mathcal{ID}$ δ^0 conforms to δ^r is denoted p_r. To satisfy the equation $\Delta_0^R = \Delta_2^R$, it suffices to find message pairs such that the observed differentials equal the target one, that is, $\Delta_0^R = \delta^R$ and $\Delta_2^R = \delta^R$. The naive approach is to try about $1/p_r$ random messages each. This complexity can however be lowered down by:

- Finding constraints on the message pair so that it conforms to the difference δ^1 after one round with certainty (this will be achieved by *linearization*).
- Deriving message pairs conforming to δ^r from a single conforming pair (the message-modification technique used will be *neutral bits*).

Finally, to have $\nabla_0^R = \nabla_2^R$, we need to find message pairs such that $\nabla_0^R = \delta^R \oplus \delta^0$ and $\nabla_2^R = \delta^R \oplus \delta^0$ (i.e. the additions are not producing carry bits). Given a random message pair that conforms to δ^R, this holds with probability about 2^{-v-w} where v and w are the respective weights of the $\mathcal{ID}$ δ^0 and of the target $\mathcal{OD}$ δ^R (excluding the linear MSB's). The three next paragraphs are respectively dedicated to finding an optimal differential, describing the linearization procedure, and describing the neutral bits technique.

Remark 3. One can observe that the constants of F_0 and F_2 are almost similar, as well as the constants of F_1 and F_3 (cf. Appendix A). To improve the generalized birthday attack suggested in [7], a strategy is to find a pair (M_0, M_2) such that $F_0(M_0) \oplus F_2(M_2)$ is biased in any c bits after R rounds (where $c \approx 114$, see [7]), along with a second pair (M_1, M_3) with $F_1(M_1) \oplus F_3(M_3)$ biased in the same c bits. The sum $F_0(M_0) \oplus F_2(M_2)$ can be seen as the feedforward $\mathcal{OD}$ of two states having an $\mathcal{ID}$ which is nonzero in some diagonal words. However, differences in the diagonal words result in a large diffusion, and this approach seems to be much less efficient than differential attacks for only one function F_i.

Finding a High-Probability Differential. We search for a *linear* differential over several rounds of Rumba, i.e. a differential holding with certainty when additions are replaced by XOR's, see [13]. The differential is independent of the diagonal constants, and it is expected to have high probability for genuine Rumba if the linear differential has low weight. An exhaustive search for suitable $\mathcal{ID}$'s is not traceable, so we choose another method: We focus on a *single column* in X_i, and consider the weight of the input (starting with the diagonal element, which must be zero). With a fixed relative position of the non-zero bits in this input, one can obtain an output of low weight after the first linear round of Rumba (i.e. using the linearized Eq. 3). Here is a list of the mappings (showing the weight only) which have at most weight 2 in each word of the input and output:

$$
\begin{array}{ll}
g_1 : (0,0,0,0) \to (0,0,0,0) & g_8 : \ (0,1,2,0) \to (1,1,1,0) \\
g_2 : (0,0,1,0) \to (2,0,1,1) & g_9 : \ (0,1,2,2) \to (1,1,1,2) \\
g_3 : (0,0,1,1) \to (2,1,0,2) & g_{10} : (0,2,1,1) \to (0,1,0,0) \\
g_4 : (0,1,0,1) \to (1,0,0,1) & g_{11} : (0,2,1,2) \to (0,0,1,1) \\
g_5 : (0,1,1,0) \to (1,1,0,1) & g_{12} : (0,2,2,1) \to (0,1,1,1) \\
g_6 : (0,1,1,1) \to (1,0,1,0) & g_{13} : (0,2,2,1) \to (2,1,1,1) \\
g_7 : (0,0,2,1) \to (2,1,1,1) & g_{14} : (0,2,2,2) \to (2,0,2,0)
\end{array}
$$

The relations above can be used to construct algorithmically a suitable $\mathcal{ID}$ with all 4 columns. Consider the following example, where the state after the first round is again a combination of useful rows: $(g_1, g_{10}, g_1, g_{11}) \to (g_1, g_2, g_4, g_1)$. After 2 rounds, the difference has weight 6 (with weight 3 in the diagonal words). There is a class of $\mathcal{ID}$'s with the same structure: $(g_1, g_{10}, g_1, g_{11})$, $(g_1, g_{11}, g_1, g_{10})$, $(g_{10}, g_1, g_{11}, g_1)$, $(g_{11}, g_1, g_{10}, g_1)$. The degree of freedom is large enough to construct these 2-round linear differentials: the positions of the nonzero bits in a single mapping g_i are symmetric with respect to rotation of words (and the required g_i have an additional degree of freedom). Any other linear differential constructed with g_i has larger weight after 2 rounds. Let $\Delta_{i,j}$ denote the difference of word $j = 0, \ldots, 15$ in state $i = 0, \ldots, 3$. For our attacks on Rumba, we will consider the following input difference (with optimal rotation, such that many MSB's are involved):

$$
\begin{array}{llll}
\Delta^0_{i,2} & = \mathsf{00000002} & \Delta^0_{i,8} & = \mathsf{80000000} \\
\Delta^0_{i,4} & = \mathsf{00080040} & \Delta^0_{i,12} & = \mathsf{80001000} \\
\Delta^0_{i,6} & = \mathsf{00000020} & \Delta^0_{i,14} & = \mathsf{01001000}
\end{array}
$$

and $\Delta^0_{i,j} = 0$ for all other words j. The weight of differences for the first four linearized rounds is as follows (the subscript of the arrows denotes the approximate probability p_r that a random message pair conforms to this differential for a randomly chosen value for diagonal constants):

$$
\begin{pmatrix} 0\,0\,1\,0 \\ 2\,0\,1\,0 \\ 1\,0\,0\,0 \\ 2\,0\,2\,0 \end{pmatrix}
\xrightarrow[2^{-4}]{\text{Round}}
\begin{pmatrix} 0\,0\,0\,0 \\ 0\,0\,0\,0 \\ 1\,0\,0\,0 \\ 1\,0\,1\,0 \end{pmatrix}
\xrightarrow[2^{-7}]{\text{Round}}
\begin{pmatrix} 0\,0\,0\,0 \\ 0\,0\,0\,0 \\ 1\,1\,2\,0 \\ 0\,0\,1\,1 \end{pmatrix}
\xrightarrow[2^{-41}]{\text{Round}}
\begin{pmatrix} 2\,2\,3\,1 \\ 0\,3\,4\,2 \\ 1\,1\,7\,3 \\ 1\,1\,1\,6 \end{pmatrix}
\xrightarrow[2^{-194}]{\text{Round}}
\begin{pmatrix} 8 & 3 & 2 & 4 \\ 5 & 10 & 3 & 4 \\ 9 & 11 & 13 & 7 \\ 6 & 9 & 10 & 9 \end{pmatrix}
$$

With this fixed $\mathcal{ID}$, we can determine the probability that the $\mathcal{OD}$ obtained by genuine Rumba corresponds to the $\mathcal{OD}$ of linear Rumba. Note that integer addition is the only nonlinear operation. Each nonzero bit in the $\mathcal{ID}$ of an integer addition behaves linearly (i.e. it does not create or annihilate a sequence of carry bits) with probability 1/2, while a difference in the MSB is always linear. In the first round, there are only four bits with associated probability 1/2, hence $p_1 = 2^{-4}$ (see also the subsection on linearization). The other cumulative probabilities are $p_2 = 2^{-7}$, $p_3 = 2^{-41}$, $p_4 = 2^{-194}$. For 3 rounds, we have weights $v = 7$ and $w = 37$, thus the overall complexity to find a collision after 3 rounds is about $2^{41+37+7} = 2^{85}$. For 4 rounds, $v = 7$ and $w = 112$, leading to a complexity 2^{313}. The probability that feedforward behaves linearly can be increased by choosing low-weight inputs.

Linearization. The first round of our differential has a theoretical probability of $p_1 = 2^{-4}$ for a random message. This is roughly confirmed by our experiments, where exact probabilities depend on the diagonal constants (for example, we experimentally observed $p_1 = 2^{-6.6}$ for F_0, and $p_1 = 2^{-6.3}$ for F_2, the other two probabilities are even closer to 2^{-4}). We show here how to set constraints on the message so that the first round differential holds with certainty, using methods similar to the ones in [13].

Let us begin with the first column of F_0, where $c_{0,0} = x_{0,0} = \texttt{73726966}$. In the first addition $x_{0,0} + x_{0,12}$, we have to address $\Delta^0_{0,12}$, which has a nonzero (and non-MSB) bit on position 12 (counting from 0). The bits of the constant are $[x_{0,0}]_{12-10} = (\texttt{010})_2$, hence the choice $[x_{0,12}]_{11,10} = (\texttt{00})_2$ is sufficient for linearization. This corresponds to $x_{0,12} \leftarrow x_{0,12} \wedge \texttt{FFFF3FFF}$. The subsequent 3 additions of the first column are always linear as only MSB's are involved. Then, we linearize the third column of F_0, where $c_{0,2} = x_{0,10} = \texttt{30326162}$. In the first addition $x_{0,10} + x_{0,6}$, we have to address $\Delta^0_{0,6}$, which has a nonzero bit on position 5. The relevant bits of the constant are $[x_{0,10}]_{5-1} = (\texttt{10001})_2$, hence the choice $[x_{0,6}]_{4-1} = (\texttt{1111})_2$ is sufficient for linearization. This corresponds to $x_{0,6} \leftarrow x_{0,6} \vee \texttt{0000001E}$. In the second addition $z_{0,14} + x_{0,10}$, the updated difference $\Delta^1_{0,14}$ has a single bit on position 24. The relevant bits of the constant are $[x_{0,10}]_{24,23} = (\texttt{00})_2$, hence the choice $[z_{0,14}]_{23} = (\texttt{0})_2$ is sufficient. Notice that conditions on the updated words must be transformed to the initial state words. As $z_{0,14} = x_{0,14} \oplus (x_{0,10} + x_{0,6}) \lll 8$, we find the condition $[x_{0,14}]_{23} = [x_{0,10} + x_{0,6}]_{16}$. If we let both sides be zero, we have $[x_{0,14}]_{23} = (\texttt{0})_2$ or $x_{0,14} \leftarrow x_{0,14} \wedge \texttt{FF7FFFFF}$, and $[x_{0,10} + x_{0,6}]_{16} = (\texttt{0})_2$. As $[x_{0,10}]_{16,15} = (\texttt{00})_2$, we can choose $[x_{0,6}]_{16,15} = (\texttt{00})_2$ or $x_{0,6} \leftarrow x_{0,6} \wedge \texttt{FFFE7FFF}$. Finally, the third addition $z_{0,2} + z_{0,14}$ must be linearized with respect to the single bit in $\Delta^1_{0,14}$ on position 24. A sufficient condition for linearization is $[z_{0,2}]_{24,23} = (\texttt{00})_2$ and $[z_{0,14}]_{23} = (\texttt{0})_2$. The second condition is already satisfied, so we can focus on the first condition. The update is defined by $z_{0,2} = x_{0,2} \oplus (z_{0,14} + x_{0,10}) \lll 9$, so we set $[x_{0,2}]_{24,23} = (\texttt{00})_2$ or $x_{0,2} \leftarrow x_{0,2} \wedge \texttt{FE7FFFFF}$, and require $[z_{0,14} + x_{0,10}]_{15,14} = (\texttt{00})_2$. As $[x_{0,10}]_{15-13} = (\texttt{011})_2$, we can set $[z_{0,14}]_{15-13} = (\texttt{101})_2$. This is satisfied by choosing $[x_{0,14}]_{15-13} = (\texttt{000})_2$ or $x_{0,14} \leftarrow x_{0,14} \wedge \texttt{FFFF1FFF}$, and by choosing $[x_{0,10} + x_{0,6}]_{8-6} = (\texttt{101})_2$. As $[x_{0,10}]_{8-5} = (\texttt{1011})_2$, we set $[x_{0,6}]_{8-5} = (\texttt{1111})_2$ or $x_{0,6} \leftarrow x_{0,6} \vee \texttt{000001E0}$. Altogether, we fixed 18 (distinct) bits of the input, other linearizations are possible.

The first round of F_2 can be linearized with exactly the same conditions. This way, we save an average factor of 2^4 (additive complexities are ignored). This linearization with sufficient conditions does not work well for more than one round because of an avalanche effect of fixed bits. We lose many degrees of freedom, and contradictions are likely to occur.

Neutral Bits. Thanks to linearization, we can find a message pair conforming to δ^2 within about $1/(2^{-7+4}) = 2^3$ trials. Our goal now is to efficiently derive from such a pair many other pairs that are conforming to δ^2, so that a search for three rounds can start after the second round, by using the notion of neutral bits again (cf. §3.3). Neutral bits can be identified easily for a fixed pair of messages, but if several neutral bits are complemented in parallel, then the resulting message pair may not conform anymore. A heuristic approach was introduced in [9], using a *maximal 2-neutral set*. A 2-neutral set of bits is a subset of neutral bits, such that the message pair obtained by complementing any two bits of the subset in parallel also conform to the differential. The size of this set is denoted n. In general, finding a 2-neutral set is an NP-complete problem—the problem is equivalent to the Maximum Clique Problem from graph theory, but good heuristic algorithms for dense graphs exist, see e.g. [10]. In the case of Rumba, we compute the value n for different message pairs that conform to δ^2 and choose the pair with maximum n. We observe that about 1/2 of the 2^n message pairs (derived by flipping some of the n bits of the 2-neutral set) conform to the differential[5]. This probability p is significantly increased, if we complement at most $\ell \ll n$ bits of the 2-neutral set, which results in a message space (not contradicting with the linearization) of size about $p \cdot \binom{n}{\ell}$. At this point, a full collision for 3 rounds has a reduced theoretical complexity of $2^{85-7}/p = 2^{78}/p$ (of course, p should not be smaller than 2^{-3}). Since we will have $p > \frac{1}{2}$ for a suitable choice of ℓ, the complexity gets reduced from 2^{85} to less than 2^{79}.

4.3 Experimental Results

We choose a random message of low weight, apply the linearization for the first round and repeat this about 2^3 times until the message pairs conforms to δ^2. We compute then the 2-neutral set of this message pair. This protocol is repeated a few times to identify a message pair with large 2-neutral set:

- For F_0, we find the pair of states (X_0, X_0') of low weight, with 251 neutral bits and a 2-neutral set of size 147. If we flip a random subset of the 2-neutral bits, then the resulting message pair conforms to δ^2 with probability $\Pr = 0.52$.

$$X_0 = \begin{pmatrix} \texttt{73726966} & \texttt{00000400} & \texttt{00000080} & \texttt{00200001} \\ \texttt{00002000} & \texttt{6d755274} & \texttt{000001fe} & \texttt{02000008} \\ \texttt{00000040} & \texttt{00000042} & \texttt{30326162} & \texttt{10002800} \\ \texttt{00000080} & \texttt{00000000} & \texttt{01200000} & \texttt{636f6c62} \end{pmatrix}$$

- For F_2, we find the pair of states (X_2, X_2') of low weight, with 252 neutral bits and a 2-neutral set of size 146. If we flip a random subset of the 2-neutral bits, then the resulting message pair conforms to δ^2 with probability $\Pr = 0.41$.

$$X_2 = \begin{pmatrix} \texttt{72696874} & \texttt{00000000} & \texttt{00040040} & \texttt{00000400} \\ \texttt{00008004} & \texttt{6d755264} & \texttt{000001fe} & \texttt{06021184} \\ \texttt{00000000} & \texttt{00800040} & \texttt{30326162} & \texttt{00000000} \\ \texttt{00000300} & \texttt{00000400} & \texttt{04000000} & \texttt{636f6c62} \end{pmatrix}$$

[5] In the case of SHA-0, about 1/8 of the 2^n message pairs (derived from the original message pair by complementing bits from the 2-neutral set) conform to the differential for the next round.

Given these pairs for 2 rounds, we perform a search in the 2-neutral set by flipping at most 10 bits (that gives a message space of about 2^{50}), to find pairs that conform to δ^3. This step has a theoretical complexity of about 2^{34} for each pair (which was verified in practice). For example, in (X_0, X_0') we can flip the bits {59, 141, 150, 154, 269, 280, 294 ,425} in order to get a pair of states $(\bar{X}_0, \bar{X}_0')$ that conforms to δ^3.In the case of (X_2, X_2'), we can flip the bits {58, 63, 141, 271, 304, 317, 435, 417, 458, 460} in order to get a pair of states $(\bar{X}_2, \bar{X}_2')$ that conforms to δ^3.

$$\bar{X}_0 = \begin{pmatrix} \texttt{73726966} & \texttt{08000400} & \texttt{00000080} & \texttt{00200001} \\ \texttt{04400000} & \texttt{6d755274} & \texttt{000001fe} & \texttt{02000008} \\ \texttt{01002040} & \texttt{00000002} & \texttt{30326162} & \texttt{10002800} \\ \texttt{00000080} & \texttt{00000200} & \texttt{01200000} & \texttt{636f6c62} \end{pmatrix}$$

$$\bar{X}_2 = \begin{pmatrix} \texttt{72696874} & \texttt{84000000} & \texttt{00040040} & \texttt{00000400} \\ \texttt{0000a004} & \texttt{6d755264} & \texttt{000001fe} & \texttt{06021184} \\ \texttt{00008000} & \texttt{20810040} & \texttt{30326162} & \texttt{00000000} \\ \texttt{00000300} & \texttt{00080402} & \texttt{04001400} & \texttt{636f6c62} \end{pmatrix}$$

At this point, we have collisions for 3-round Rumba without feedforward, hence $\Delta_0^3 \oplus \Delta_2^3 = 0$. If we include feedforward for the above pairs of states, then $\nabla_0^3 \oplus \nabla_2^3$ has weight 16, which corresponds to a near-collision. Note that a near-collision indicates non-randomness of the reduced-round compression function (we assume a Gaussian distribution centered at 256). This near-collision of low weight was found by using a birthday-based method: we produce a list of pairs for F_0 that conform to δ^3 (using neutral bits as above), together with the corresponding value of ∇_0^3. The same is done for F_2. If each list has size N, then we can produce N^2 pairs of $\nabla_0^3 \oplus \nabla_2^3$ in order to identify near-collisions of low weight.

However, there are no neutral bits for the pairs $(\bar{X}_0, \bar{X}_0')$ and $(\bar{X}_2, \bar{X}_2')$ with respect to δ^3. This means that we cannot completely separate the task of finding full collisions with feedforward, from finding collisions without feedforward (and we can not use neutral bits to iteratively find pairs that conform to δ^4). To find a full collision after three rounds, we could perform a search in the 2-neutral set of (X_0, X_0') and (X_2, X_2') by flipping at most 20 bits. In this case, the resulting pairs conform to δ^2 with probability at least $\Pr = 0.68$, and the message space has a size of about 2^{80}. The overall complexity becomes $2^{78}/0.68 \approx 2^{79}$ (compared to 2^{85} without linearization and neutral bits). Then, we try to find near-collisions of low weight for 4 rounds, using the birthday method described above. Within less than one minute of computation, we found the pairs $(\bar{\bar{X}}_0, \bar{\bar{X}}_0')$ and $(\bar{\bar{X}}_2, \bar{\bar{X}}_2')$ such that $\nabla_0^4 \oplus \nabla_2^4$ has weight 129. Consequently, the non-randomness of the differential is propagating up to 4 rounds.

$$\bar{\bar{X}}_0 = \begin{pmatrix} \texttt{73726966} & \texttt{00020400} & \texttt{00000080} & \texttt{00200001} \\ \texttt{00002400} & \texttt{6d755274} & \texttt{000001fe} & \texttt{02000008} \\ \texttt{00000040} & \texttt{00220042} & \texttt{30326162} & \texttt{10002800} \\ \texttt{00000080} & \texttt{00001004} & \texttt{01200000} & \texttt{636f6c62} \end{pmatrix}$$

$$\bar{\bar{X}}_2 = \begin{pmatrix} \texttt{72696874} & \texttt{00001000} & \texttt{80040040} & \texttt{00000400} \\ \texttt{00008804} & \texttt{6d755264} & \texttt{000001fe} & \texttt{06021184} \\ \texttt{00000000} & \texttt{80800040} & \texttt{30326162} & \texttt{00000000} \\ \texttt{00000300} & \texttt{00000450} & \texttt{04000000} & \texttt{636f6c62} \end{pmatrix}$$

5 Conclusions

We presented a novel method for attacking reduced-round Salsa20 and ChaCha, inspired by correlation attacks and by the notion of neutral bits. This allows to give the first attack faster than exhaustive search on the stream cipher Salsa20/8 with a 256-bit key. For the compression function Rumba the methods of linearization and neutral bits are applied to a high probability differential to find collisions on 3-round Rumba within 2^{79} trials, and to efficiently find low weight near collisions on 3-round and 4-round Rumba.

Acknowledgments

The authors would like to thank Dan Bernstein for insightful comments on a preliminary draft, the reviewers of FSE 2008 who helped us to improve the clarity of the paper, and Florian Mendel for his proofreading. J.-Ph. Aumasson is supported by the Swiss National Science Foundation (SNF) under project number 113329. S. Fischer is supported by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center of the SNF under grant number 5005-67322. W. Meier is supported by Hasler Foundation (see `http://www.haslerfoundation.ch`) under project number 2005. C. Rechberger is supported by the Austrian Science Fund (FWF), project P19863, and by the European Commission through the IST Programme under Contract IST-2002-507932 ECRYPT.

References

1. Thomas Baignères, Pascal Junod, and Serge Vaudenay. How far can we go beyond linear cryptanalysis? In Pil Joong Lee, editor, *ASIACRYPT*, volume 3329 of *LNCS*, pages 432–450. Springer, 2004.
2. Mihir Bellare and Daniele Micciancio. A new paradigm for collision-free hashing: Incrementality at reduced cost. In Walter Fumy, editor, *EUROCRYPT*, volume 1233 of *LNCS*, pages 163–192. Springer, 1997.
3. Daniel J. Bernstein. ChaCha, a variant of Salsa20. See `http://cr.yp.to/chacha.html`. See also [8].
4. Daniel J. Bernstein. Salsa20 and ChaCha. eSTREAM discussion forum, May 11, 2007.
5. Daniel J. Bernstein. Salsa20. Technical Report 2005/025, eSTREAM, ECRYPT Stream Cipher Project, 2005. See also `http://cr.yp.to/snuffle.html`.
6. Daniel J. Bernstein. Salsa20/8 and Salsa20/12. Technical Report 2006/007, eSTREAM, ECRYPT Stream Cipher Project, 2005.

7. Daniel J. Bernstein. What output size resists collisions in a XOR of independent expansions? ECRYPT Workshop on Hash Functions, 2007. See also `http://cr.yp.to/rumba20.html`.
8. Daniel J. Bernstein. ChaCha, a variant of Salsa20. In *SASC 2008 – The State of the Art of Stream Ciphers*. ECRYPT, 2008. See also `http://www.ecrypt.eu.org/stvl/sasc2008`.
9. Eli Biham and Rafi Chen. Near-collisions of SHA-0. In Matthew K. Franklin, editor, *CRYPTO*, volume 3152 of *LNCS*, pages 290–305. Springer, 2004.
10. Samuel Burer, Renato D.C. Monteiro, and Yin Zhang. Maximum stable set formulations and heuristics based on continuous optimization. *Mathematical Programming*, 64:137–166, 2002.
11. Paul Crowley. Truncated differential cryptanalysis of five rounds of Salsa20. In *SASC 2006 – Stream Ciphers Revisited*, 2006.
12. ECRYPT. eSTREAM, the ECRYPT Stream Cipher Project. See `http://www.ecrypt.eu.org/stream`.
13. Simon Fischer, Willi Meier, Côme Berbain, Jean-François Biasse, and Matthew J. B. Robshaw. Non-randomness in eSTREAM candidates Salsa20 and TSC-4. In Rana Barua and Tanja Lange, editors, *INDOCRYPT*, volume 4329 of *LNCS*, pages 2–16. Springer, 2006.
14. Pascal Junod and Serge Vaudenay. Optimal key ranking procedures in a statistical cryptanalysis. In Thomas Johansson, editor, *FSE*, volume 2887 of *LNCS*, pages 235–246. Springer, 2003.
15. Thomas Siegenthaler. Decrypting a class of stream ciphers using ciphertext only. *IEEE Transactions on Computers*, 34(1):81–85, 1985.
16. Yukiyasu Tsunoo, Teruo Saito, Hiroyasu Kubo, Tomoyasu Suzaki, and Hiroki Nakashima. Differential cryptanalysis of Salsa20/8. In *SASC 2007 – The State of the Art of Stream Ciphers*, 2007.
17. David Wagner. A generalized birthday problem. In Moti Yung, editor, *CRYPTO*, volume 2442 of *LNCS*, pages 288–303. Springer, 2002.

A Constants

Here are the diagonal constants for Salsa20 and ChaCha (function Round) and for Rumba (functions F_0 to F_3).

	Round	F_0	F_1	F_2	F_3
c_0	61707865	73726966	6f636573	72696874	72756f66
c_1	3320646E	6d755274	7552646e	6d755264	75526874
c_2	79622D32	30326162	3261626d	30326162	3261626d
c_3	6B206574	636f6c62	6f6c6230	636f6c62	6f6c6230