Noname manuscript No.
(will be inserted by the editor)

Improved Key Recovery Attacks on Reduced-Round Salsa20

Sabyasachi Dey *, Gregor Leander **, Nitin Kumar Sharma ***

the date of receipt and acceptance should be inserted later

Abstract In this paper, we present an improved attack on the stream cipher Salsa20. Our improvements are based on two technical contributions. First, we make use of a distribution of a linear combination of several random variables that are derived from different differentials and explain how to exploit this in order to improve the attack complexity. Secondly, we study and exploit how to choose the actual value for so-called probabilistic neutral bits optimally. Because of the limited influence of these key bits on the computation, in the usual attack approach, these are fixed to a constant value, often zero for simplicity. As we will show, despite the fact that their influence is limited, the constant can be chosen in significantly better ways, and intriguingly, zero is the worst choice. Using this, we propose the first-ever attack on 7.5-round of 128-bit key version of Salsa20. Also, we provide improvements in the attack against the 8-round of 256-bit key version of Salsa20 and the 7-round of 128-bit key version of Salsa20.

Keywords Salsa20, Differential-Linear Cryptanalysis, Probabilistic Neutral Bits, Key recovery.

1 Introduction

Salsa20 is, as part of the eSTREAM portfolio for software, the base design for the cipher ChaCha used in TLS and one of the most important and analyzed stream ciphers today. It was designed by Daniel J. Bernstein in 2005 and was submitted to the ECRYPT Stream Cipher Project (eSTREAM) [3]. The cipher was a part of the final eSTREAM portfolio in the software profile. The original design has 20 rounds. To improve the encryption time, the designer tweaked the submissions into two round-reduced variants, Salsa20/8 and Salsa20/12 [2]. The cipher uses a function, referred to as Salsa-core, that takes a 64-byte (512-bit) input and produces a 64-byte (512-bit) output. The input consists of a nonce, a counter, some constants, and the key. The key used in the encryption is either 256-bit length or 128-bit length. The Salsa-core is an ARX design, i.e., it is based on addition, fixed-distance rotation, and XOR of 32-bit words. We recall the details of the design in Section 2.

* Department of Mathematics, Birla Institute of Technology and Science Pilani, Hyderabad, Jawahar Nagar, Hyderabad 500078, India.

** Ruhr University Bochum, Bochum, Germany.

*** Department of Mathematics, Birla Institute of Technology and Science Pilani, Hyderabad, Jawahar Nagar, Hyderabad 500078, India.

E-mail: *sabya.ndp@gmail.com, **gregor.leander@rub.de, ***sharmanitinkumar685@gmail.com.

Known Cryptanalysis

The cipher has been cryptanalyzed ever since it was made public. Most of the works focused on the 256-bit key version. Below, we summarize the main cryptanalytic results on the 256 and 128-bit key versions of Salsa20 separately.

Attacks on 256*-bit key version of Salsa20*

The first cryptanalysis on 256-bit key version of Salsa20 was given by P. Crowley in 2005 [7]. The attack was on the 5-round variant of the cipher using a meet-in-the-middle approach technique with 3 rounds in the forward direction and 2 rounds in the backward direction after guessing 160 key bits. The time complexity of the attack was 2^{165}. In 2006, a significantly faster attack on the 6-round Salsa20 cipher was presented at Indocrypt by S. Fischer, W. Meier, C. Berbain, J. F. Biasse, and M. J. B. Robshaw [10]. This attack runs 4 rounds forward and 2 rounds backward after guessing 160 bits of the key. Its time complexity is 2^{177}. In the next year, Tsunoo et al. proposed an attack on 7-round Salsa20 by guessing 171 key bits [17]. In 2007, Aumasson et al. proposed an attack on the 8-round variant of the Salsa20 cipher by guessing 228 key bits [1]. Technically, the main contribution of this paper was the introduction of the idea of non-significant key bits, namely Probabilistic Neutral Bits (PNBs). The attack complexity achieved by this technique was 2^{251}. In 2012, this attack was improved slightly by Shi et al. [16] to 2^{250}. After that, Maitra, G. Paul, and W. Meier [13] provided an attack complexity of $2^{247.2}$ by improving the idea of finding non-significant key bits. Later in [14], Maitra improved the attack complexity further and reduced it to $2^{245.5}$. In 2016, A. R. Choudhuri and S. Maitra [6] provided a new approach to getting a multi-bit output instead of a single bit and hence improved the attack complexity to $2^{244.9}$. Finally in 2017, Dey et al. provided an improved analysis of finding non-significant key bits and reduced the complexity to $2^{243.67}$ [9].

Attacks on 128*-bit key version of Salsa20*

Compared to the main works on Salsa20 with a 256-bit key, there are relatively fewer attacks on the 128-bit key version of Salsa20. The work of Fischer et al. [10] mentioned that we can recover the 5 rounds of 128-bit key version of Salsa20 with around 2^{81} operations. Aumasson et al.'s work [1] also proposed an attack on the 7-round of 128-bit key version of Salsa20 with a complexity of 2^{111}. In 2012, Shi et al. [16] improved this attack to 2^{109}. Much later, in 2018, Deepthi et al. [8] provided an attack with time complexity 2^{107}. Till date, there has been no attack on the 128-bit key version of Salsa20 beyond 7 rounds.

Our Contribution

In this work, we significantly improve the known attacks on Salsa20. In particular, we provide the first-ever attack on the 7.5-round of the 128-bit key version of Salsa20 in Subsection 5.1. For the Salsa20 version with 256 key bits, we produce an improved attack with complexity $2^{240.23}$. For comparison with the previous attacks, those complexities are given in Table 1 as well.

Technically, our improvements are based on two orthogonal ideas that we combine for our new attacks. After recalling the details of the structure of the cipher and the usual differential-linear attack approaches on the cipher in Section 2, we discuss our findings in Section 3 and Section 4.

As our first improvement in Section 3, we propose a technique based on the linear combination of variables derived from several differentials together to construct the differential attack.

Our second improvement stems from a better understanding of the PNBs that we present in Section 4. We provide a mathematical study on how to optimally assign the values of the PNBs so that we can

produce a good backward bias. While the value of the PNBs should be expected to be irrelevant, as this is the basic idea, it turns out that this is not the case. Indeed, there is the optimal choice, and, most intriguingly, the most natural choice of fixing the non-guessed key bits to zero is the worst possible choice. We start by providing a simplified example in order to explain the effect more clearly. We then give a theoretical analysis of choosing values at the PNB positions.

Finally, we apply those ideas on 128 and a 256-bit key version of Salsa20 in Section 5. For the 128-bit key version, we produce an attack on the 7-round, which is 18 times faster than the previously existing attack [8] and present the first attack on 7.5-round [Subsection 5.1]. For the 256-bit key version, our attack on 8-round improves the best-known attack [9] by a factor of roughly 8 both in time and data complexity [Subsection 5.2].

Cipher	Key-size	Rounds	Time	Data	Ref.
Salsa20	128	7	2^{128}	0	Brute-force attack
			2^{111}	2^{21}	[1]
			2^{109}	2^{19}	[16]
			2^{107}	2^{24}	[8]
			$2^{102.82}$	$2^{28.77}$	[Our work]
		7.5	2^{128}	0	Brute-force attack
			$2^{124.22}$	$2^{23.06}$	[Our work]
	256	8	2^{256}	0	Brute-force attack
			2^{251}	2^{31}	[1]
			2^{250}	2^{27}	[16]
			$2^{247.2}$	2^{30}	[13]
			$2^{245.5}$	2^{96}	[14]
			$2^{244.9}$	2^{96}	[6]
			$2^{243.7}$	$2^{30.86}$	[9]
			$2^{240.62}$	$2^{27.56}$	[Our work]

Table 1: Known Full Key Recovery Attacks.

2 Details of Salsa20

Salsa20 is an ARX cipher. The detailed structure and the use of various tools in its design have been discussed in [4]. Below, we discuss the structure of the 256-bit and 128-bit key versions of the Salsa20 cipher.

2.1 Design

The cipher is represented in a 4×4 matrix form consisting of 16 words, where each word is 32 bits. The 256-bit key version of cipher takes 8 key words $(k_0, k_1, \ldots, k_7)$, 4 constants words (c_0, c_1, c_2, c_3), 2

$\mathcal{IV}$ words (t_0, t_1) and 2 counter words (v_0, v_1) as input and generates a 512-bit output. The constant words (c_0, c_1, c_2, c_3) for 256-bit key version have fixed value as: $c_0 =$ `0x61707865`, $c_1 =$ `0x3320646e`, $c_2 =$ `0x79622d32`, $c_3 =$ `0x6b206574`. In case of 128-bit key version the 8 key words $(k_0, k_1, \ldots, k_7)$ are taken as $k_{i+4} = k_i,\ \forall\ 0 \leq i \leq 3$. The constant words (c_0, c_1, c_2, c_3) for 128-bit key version are slightly different from 256-bit key version and are given as: $c_0 =$ `0x61707865`, $c_1 =$ `0x3120646e`, $c_2 =$ `0x79622d36`, $c_3 =$ `0x6b206574`. The state matrix form of the Salsa20 is given by:

$$X = \begin{pmatrix} X_0 & X_1 & X_2 & X_3 \\ X_4 & X_5 & X_6 & X_7 \\ X_8 & X_9 & X_{10} & X_{11} \\ X_{12} & X_{13} & X_{14} & X_{15} \end{pmatrix} = \begin{pmatrix} c_0 & k_0 & k_1 & k_2 \\ k_3 & c_1 & v_0 & v_1 \\ t_0 & t_1 & c_2 & k_4 \\ k_5 & k_6 & k_7 & c_3 \end{pmatrix}.$$

In the Salsa20 cipher algorithm, the function operated in each round is a nonlinear operation which transforms a vector (a, b, c, d) into (a', b', c', d') by performing the $\oplus, \boxplus$ and $\lll$ operation for each round. Here, $\oplus$ denotes XOR operation between the bits, $\boxplus$ is the addition modulo 2^{32}, $\lll$ is left cyclic rotation operation.

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

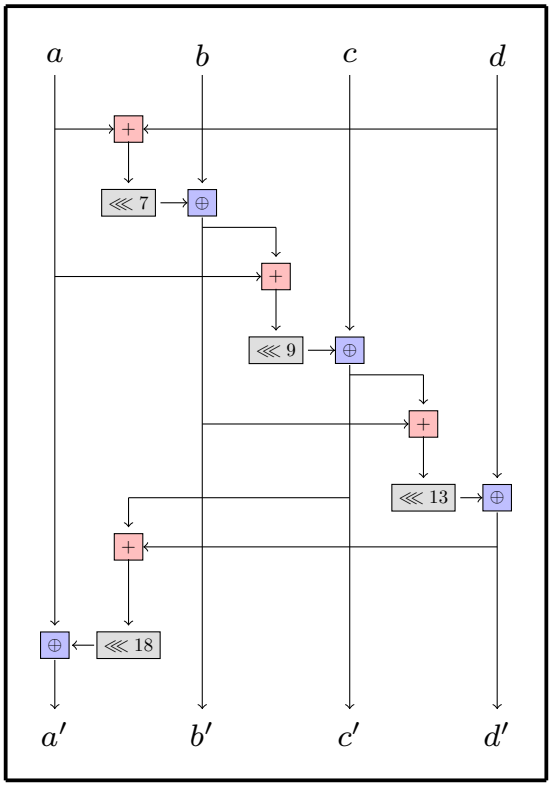

Fig. 1: Quaterround Function in Salsa20.

This function described in Equation 1 is known as the quarterround function. This function is applied to each column and row of the matrix X. The application of this function along columns is known as column-round. The ordering of (a, b, c, d) along the columns is $(X_0, X_4, X_8, X_{12}), (X_5, X_9, X_{13}, X_1)$, $(X_{10}, X_{14}, X_2, X_6)$ and $(X_{15}, X_3, X_7, X_{11})$. After columns the quarterround function is operated along the rows called row-round. The ordering along the rows is $(X_0, X_1, X_2, X_3), (X_5, X_6, X_7, X_4), (X_{10}, X_{11}, X_8, X_9)$ and $(X_{15}, X_{12}, X_{13}, X_{14})$. In the Salsa20 cipher algorithm, if the quarterround function is performed on X matrix up to n rounds, then we denote the obtained matrix by $X^{(n)}$. The final output matrix is denoted by Z and achieved by the addition of input matrix X and iterated matrix $X^{(n)}$, i.e., $Z = X + X^{(n)}$. The quarterround function is reversible and known as the reverse quarterround function. This function is used to obtain the relation between the intermediate state and ciphertext by operating in the reverse direction. The detailed design of the Salsa20 cipher is described in [3] and [4].

Notation	Meaning
X	The state matrix of the cipher consisting of 16 words
$X^{(0)}$	Initial state matrix
$\Delta X_i^{(r)}[j]$	XOR difference after r-th round of the j-th bit of the i-th word of two states X and X'
$\mathcal{ID}$ Column	Column which includes the input difference
$\epsilon_{d'}$	Denotes the bias obtained after r_1 rounds in forward direction
ϵ_l	Denotes the bias obtained on linear approximation in forward direction
γ_l	Denotes the bias of the event $\Delta\tilde{M}_p[q] = \Delta X_p^{(r)}[q]$
$\tilde{X}$	State obtained by assigning values to the PNBs in the initial state matrix X
Z	Key stream block obtained by xoring X and X^n
α	Parameter for finding significance level in Neyman-Pearson lemma

Table 2: List of Notations.

2.2 General Attack Procedure

Differential and linear cryptanalysis techniques are the two major attack techniques applied to symmetric ciphers. The differential cryptanalysis was introduced by Biham and Shamir in 1990 [5] for cryptanalysis of DES-like cryptosystems. Linear cryptanalysis was first applied to FEAL-cipher by Matsui in 1992 [15]. Later, Martin Hellman and Susan K. Langford gave a combination of both attacks as Differential-Linear cryptanalysis [12] in 1994. This attack technique is widely used for cryptanalysis of symmetric ciphers.

At first, we explain the differential and linear attacks. A differential attack is a chosen plaintext attack. In this attack procedure, we introduce a change in the input of the cipher and observe the corresponding change in the output. For example, if we consider a plaintext P_t, we construct P_t' by introducing a difference in it. Now, we observe the output of the cipher for both these inputs. Let us assume the outputs are C_t and C_t', respectively. We observe the correlation between the two ciphertexts C_t and C_t', which in turn is used to find out the key. In a linear attack, a linear relation is constructed between plaintext, ciphertext, and key with high bias. The attacker then uses these relations together with known plaintext-ciphertext pairs to find the key.

These two attack procedures together form the differential-linear attack. In Differential-Linear cryptanalysis, the cipher E is divided into subciphers E_1 and E_2. The input differential exists in the first subcipher E_1 and linear approximation in the second subcipher E_2. The combination of these two attacks on $E_2 \circ E_1$ is the differential-linear analysis.

Let us take an initial state matrix X and give an input difference at the j-th bit of the i-th word to get another state matrix X'. The input difference matrix for the two initial matrices X and X' is defined as $\Delta X = X \oplus X'$. To introduce a difference at a single-bit position in the matrix, let us define a matrix $D_{\mathcal{ID}}$ as shown below:

$$D_{\mathcal{ID}} = \begin{cases} 1 & j\text{-th bit of the } i\text{-th word} \\ 0 & \text{otherwise.} \end{cases}$$

For example, if the difference is introduced at the 0-th bit of the 7-th word, then the matrix $D_{\mathcal{ID}}$ is written as:

$$D_{\mathcal{ID}} = \begin{pmatrix} 0\ 0\ 0 & 0 \\ 0\ 0\ 0 & \texttt{0x00000001} \\ 0\ 0\ 0 & 0 \\ 0\ 0\ 0 & 0 \end{pmatrix}.$$

The output differential between two states is observed after r_1-rounds. We choose a particular bit of both the matrices as the position of the output differential. Let the bit be the q-th bit of the p-th word. We denote it by $X_p^{(r_1)}[q]$. Let $\Delta X_p^{(r_1)}[q] = X_p^{(r_1)}[q] \oplus X_p'^{(r_1)}[q]$ denotes the difference observed between the two states. If a good bias is observed for this after r_1-rounds, we will use these state matrices to attack the ciphers. The bias is denoted by ϵ_d' and is defined as:

$$\Pr\left[\Delta X_p^{(r_1)}[q] = 0 | \Delta X = D_{\mathcal{ID}}\right] = \frac{1}{2}(1 + \epsilon_d'), \tag{2}$$

After the differential part we extend this differential by r_2-rounds with the help of a linear combination of bits $\left(\bigoplus_i X_{p_i}^{(r)}[q_i]\right)$ and observed the output at $r = (r_1 + r_2)$-th round. ϵ_l denotes the bias for linear approximation. The linear approximation works in a similar manner for both X and X' states. The differential-linear bias after $r = (r_1 + r_2)$-rounds is given as $\epsilon_d = \epsilon_d' \epsilon_l^2$. The bias is also known as forward bias, and the probability is given as:

$$\Pr\left[\left(\bigoplus_i \Delta X_{p_i}^{(r)}[q_i]\right) = 0 | \Delta X = D_{\mathcal{ID}}\right] = \frac{1}{2}(1 + \epsilon_d). \tag{3}$$

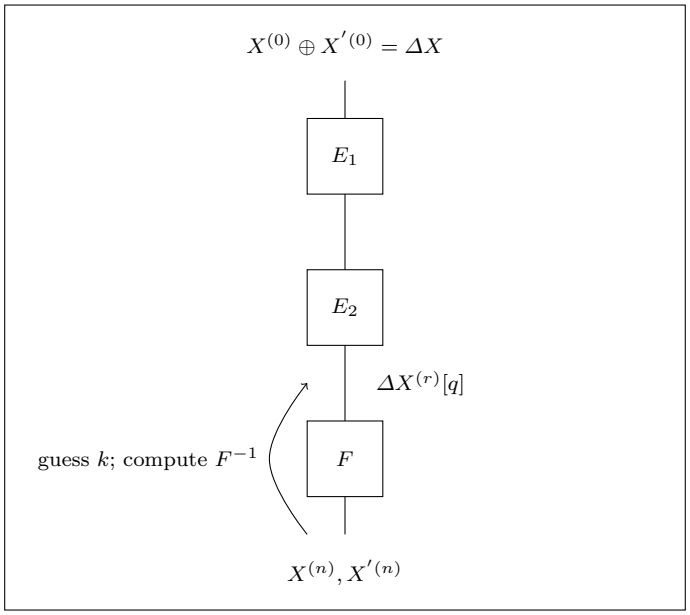

Fig. 2: General Overview of the Attack.

Finding Probabilistic Neutral Bits

Next, we explain the probabilistic neutral bits (also called non-significant key bits) and the procedure of finding those bits. Using this procedure, we partition the key bits into significant key bits and non-significant key bits. Non-significant key bits are those which influence the output difference bit with low probability. The aim of partitioning the key bits is that, instead of searching over all 2^{256} feasible possibilities of the key bits (for 256 bit key), if, for example, m bits are significant and the remaining $(256 - m)$ bits are non-significant. At first, we aim to search the m significant bits only. As a result, the maximum number of guesses is reduced to 2^m. Once we achieve these bits, we can find the remaining bits by exhaustive search.

Let us define the non-significant bits or PNBs formally. For an initial state matrix X, after introducing a suitable non-zero input difference ($\mathcal{ID}$), we get another state X'. Running X, X' for r-rounds $(1 \leqslant r < n)$ we observe the output difference ($\mathcal{OD}$) at position (p, q), i.e., $\Delta X_p^{(r)}[q]$. The bias is given by ϵ_d. After completing the n-rounds, we obtain the final state $X^{(n)}$ and $X'^{(n)}$, which are operated with the respective initial states X and X' to obtain keystream blocks Z and Z'. Considering Figure 2, we get $X^{(r)} = F^{-1}(Z - X)$ and $X'^{(r)} = F^{-1}(Z' - X')$.

In the procedure of finding the PNB, we will alter one key bit, say l, among the total key bits (128 or 256) in the initial states X and X'. $\bar{X}$ and $\bar{X}'$ are the new altered states. We apply the reverse round function on $Z - \bar{X}$ and $Z' - \bar{X}'$ by $(n - r)$-rounds and obtain the state matrices $\bar{M}$ and $\bar{M}'$ i.e., $\bar{M} = F^{-1}(Z - \bar{X})$ and $\bar{M}' = F^{-1}(Z' - \bar{X}')$. For a non-significant bit, the probability of the event $\Delta\bar{M}_p[q] = \Delta X_p^{(r)}[q]$ is expected to be high. We denote γ_l to be the bias of this event, i.e.,

$$\Pr_{v,t}\left[\Delta X_p^{(r)}[q] \oplus \Delta\bar{M}_p[q] = 0 | \Delta X = D_{\mathcal{ID}}\right] = \frac{1}{2}(1 + \gamma_l). \tag{4}$$

To construct the set of PNBs, in the work of [1], at first a threshold γ is chosen. Each of the key bits for which $\gamma_i \geq \gamma$ are considered to be probabilistically neutral.

Attack Idea and Complexity

In the attack procedure, for finding the complete key, we will assign arbitrary values to these PNBs and fix guessed values to the significant-key bits in the initial states X and X'. The newly obtained states are $\tilde{X}$ and $\tilde{X}'$. Now we apply reverse round function on $Z - \tilde{X}$ and $Z' - \tilde{X}'$ and obtain state matrices $\tilde{M}$ and $\tilde{M}'$, i.e., $\tilde{M} = F^{-1}(Z - \tilde{X})$ and $\tilde{M}' = F^{-1}(Z' - \tilde{X}')$. $\Pr_{v,t}\left[\Delta\tilde{M}_p[q] = 0 | \Delta X = D_{\mathcal{ID}}\right] = \frac{1}{2}(1 + \tilde{\epsilon})$. If $\tilde{\epsilon}$ has a higher value, then the guessed values of the significant-key bits are correct. A pictorial overview is given in Figure 2. To perform the attack, the following steps are done:

1. N pairs of keystream blocks are collected for each guessed key.
2. For every significant key bit, we compute the bias of output differential using the N pairs. Then, we compute an exhaustive search among the remaining keys to detect the non-significant keys.

If $(\Delta\tilde{M}_p[q] = \Delta X_p^{(r)}[q])$ with high probability we obtain a good bias. The bias observed is called backward bias and is denoted by ϵ_a.

For 2^m possible guesses of the significant subkey, only one guess is correct. Let us consider the null hypothesis and alternative hypothesis as follows:
H_0 : the selected guess is not correct.
H_1 : the selected variable is correct.
So $2^m - 1$ guesses satisfy H_0, and only one guess satisfies H_1. The two possible errors are as follows:

1. Error of Non-detection: In this selected variable is correct, but it is not detected. The probability of this event is $\Pr_{e_{nd}}$.
2. Error of False Alarm: In this, an incorrect variable is chosen because it gives significant bias. The probability of the event is $\Pr_{e_{fa}}$.

Using the Neyman-Pearson lemma, for $\Pr_{e_{fa}} = 2^{-\alpha}$ and $\Pr_{e_{nd}} = 1.3 \times 10^{-3}$, required number of samples N to achieve a bound on these probabilities is

$$N \approx \left(\frac{\sqrt{\alpha \log 4} + 3\sqrt{1 - \epsilon_a^2 \epsilon_d{}^2}}{\epsilon_a \epsilon_d} \right)^2 .$$

The complexity of the attack is given by the equation

$$2^m \left(N + 2^{(256-m)} Pr_{e_{fa}} \right) = 2^m N + 2^{(256-\alpha)}. \tag{5}$$

3 Exploiting Linear Combination of Random Variables Generated from Distinguishers to Reduce the Attack Complexity

In our idea, the primary requirement is that there must be multiple $\mathcal{ID}$ positions that produce a good bias after the r-th round at the same output difference position. Suppose there are k such $\mathcal{ID}$ positions, and we denote them by $\mathcal{ID}_1, \mathcal{ID}_2, \cdots \mathcal{ID}_i, \cdots \mathcal{ID}_k$. By $\mathcal{OD}$, we denote the bit/combination of bits where we observe the output difference. The corresponding biases for $\mathcal{ID}_1, \mathcal{ID}_2, \cdots \mathcal{ID}_i, \cdots \mathcal{ID}_k$ are $\epsilon_1, \epsilon_2, \cdots \epsilon_i, \cdots \epsilon_k$ respectively. This means,

$$\Pr(\Delta X_{\mathcal{OD}} = 0 | \Delta X_{\mathcal{ID}_i} = D_{\mathcal{ID}}) = \frac{1}{2}(1 + \epsilon_i).$$

In the usual attack, we have N pairs of initial states X, X', from which we generate N pairs of output keystream Z, Z'. In our new approach, since we exploit k Input Difference positions separately, and for each of them, we need N pairs of initial states X, X'.

Now, for each $(\mathcal{ID}_i)$, we have a sample of N state pairs (X, X') with the input difference at $\mathcal{ID}_i$. The set of (X, X') pairs corresponding to $\mathcal{ID}_i$ is denoted by I_i. Let us define k variables $Y_1, Y_2, \cdots Y_k$ as follows

$$Y_i = \{(X, X') \in I_i : \Delta X_{\mathcal{OD}} = 0\}.$$

Each Y_i denotes the number of pairs in I_i for which $\Delta X_{\mathcal{OD}} = 0$ out of the N pairs with input difference at $\mathcal{ID}_i$. Clearly, each of the Y_i's follows a binomial distribution with N trials with $\Pr(success) = \frac{1}{2}(1 + \epsilon_{d_i})$. To exploit all the k distinguishers, we define a new variable $Y_{cipher} = a_1Y_1 + a_2Y_2 + \cdots + a_kY_k$, where a_n's are some constants. We assign the values of these constants later.

Attack Procedure

Here, we use the bias of $Y_{cipher} = a_1Y_1 + a_2Y_2 + \cdots + a_kY_k$ as our distinguisher. Now, to produce an attack from this, we use the usual procedure where for each X and X', we assign some constant values to the PNBs and aim to guess the significant bits correctly. Please note that since the output difference position is the same for each $\mathcal{ID}_i$, the set of PNBs are also the same. From the output keystream Z, Z', we compute $Z - \tilde{X}$ and $Z' - \tilde{X}'$, run the reverse algorithm and observe the difference at the desired output difference position $(\mathcal{OD})$. Suppose $\tilde{Y}_i$ $(i \in \{1, 2, \ldots k\})$ be the variables corresponding to the $\mathcal{ID}_i$'s which count the number of pairs for which the difference is 0. Now, we consider the variable $\tilde{Y}_{cipher} = a_1\tilde{Y}_1 + a_2\tilde{Y}_2 + \cdots + a_k\tilde{Y}_k$. If this value is higher than some predetermined threshold T, we conclude that the guess of the significant bits is correct.

3.1 **Complexity Computation**

We have to distinguish the Normal distribution $\tilde{Y}$ from the distribution generated from random output. Each $\tilde{Y}_i$ follows a binomial distribution. Suppose the probability is $\frac{1}{2}(1 + \epsilon_i)$. Then, since the backward bias is ϵ_a, we express ϵ_i as $\epsilon_{d_i}.\epsilon_a$. We approximate the distribution of $\tilde{Y}_i$ by the Normal Distribution with parameters

$$E(\tilde{Y}_i) = \frac{N}{2}(1 + \epsilon_i) \quad \text{and} \quad \sigma_{\tilde{Y}_i} = \sqrt{Var(\tilde{Y}_i)} = \sqrt{\frac{N}{4}(1 + \epsilon_i)(1 - \epsilon_i)}.$$

From the distribution of a linear combination of some independent normal random variables, we know that $\tilde{Y}_{cipher}$ is a Normal distribution with mean

$$\mu_{cipher} = \frac{N}{2}\left(\sum_{i=1}^{k} a_i(1 + \epsilon_i)\right) \quad \text{and} \quad \sigma_{cipher} = \sqrt{\sum_{i=1}^{k} a_i^2\sigma_{\tilde{Y}_i}^2}.$$

Suppose we denote the distribution generated from random output by Y_0, with

$$\mu_0 = \frac{N}{2}(a_1 + a_2 \cdots + a_k) \quad \text{and} \quad \sigma_0^2 = \frac{N}{4}(a_1^2 + a_2^2 \cdots + a_k^2).$$

$$\mu_0 = \frac{N}{2}\left(\sum_{i=1}^{k} a_i\right) \quad \text{and} \quad \sigma_0 = \frac{\sqrt{N}}{2}\left(\sqrt{\sum_{i=1}^{k} a_i^2}\right).$$

Let Y be the same linear combination with coefficients $a_1, a_2, \cdots a_k$ of the outputs coming from a generator. If the generator is the cipher, $Y = \tilde{Y}_{cipher}$ and if it is random then $Y = Y_0$.

Let $H_0 : Y = \tilde{Y}_{cipher}$ and $H_1 : Y = Y_0$. Let T be the threshold. We want our error probabilities to be bounded as follows: $\Pr(Y \leq T|H_1) \leq 1.3 \times 10^{-3}$ and $\Pr(Y \geq T|H_0) \leq 2^{-\alpha}$ for some suitable α.

False Alarm Error

Therefore, $\Pr(Y_0 \geq T)$ should be upper bounded by $2^{-\alpha}$. Converting to standard normal, we have:

$$Z = \frac{Y_0 - \mu_0}{\sigma_0} = \frac{Y_0 - \frac{N}{2}\left(\sum_{i=1}^{k} a_i\right)}{\frac{\sqrt{N}}{2}\left(\sqrt{\sum_{i=1}^{k} a_i^2}\right)}. \tag{6}$$

So, $\Pr\left(\frac{Y_0-\mu_0}{\sigma_0} \geq \frac{T-\mu_0}{\sigma_0}\right) \leq 2^{-\alpha}$, where $\Pr\left(\frac{Y_0-\mu_0}{\sigma_0}\right)$ is standard normal. Since, $\Pr(Z \geq x) \leq e^{-x^2/2}$ for any x, therefore we choose $e^{-\left(\frac{T-\mu_0}{\sigma_0}\right)^2/2}$ to be $2^{-\alpha}$. This implies

$$\begin{aligned} & \left(\frac{T-\mu_0}{\sigma_0}\right)^2 = \alpha \log 4 \\ \implies \quad & \frac{T - \frac{N}{2}\left(\sum_{i=1}^{k} a_i\right)}{\frac{\sqrt{N}}{2}\left(\sqrt{\sum_{i=1}^{k} a_i^2}\right)} = \sqrt{\alpha \log 4}. \end{aligned} \tag{7}$$

Non-Detection Error

So, after converting Y_{cipher} to standard normal, we have:

$$Z = \frac{Y_{cipher} - \mu_{cipher}}{\sigma_{cipher}} = \frac{Y_{cipher} - \frac{N}{2}\left(\sum_{i=1}^{k} a_i(1+\epsilon_i)\right)}{\frac{\sqrt{N}}{2}\left(\sqrt{\sum_{i=1}^{k} a_i^2(1-\epsilon_i^2)}\right)}.$$

Now, the non-detection error probability is restricted to 1.3×10^{-3}. So, $\Pr\left(Y_{cipher} \leq T\right) \leq 1.3 \times 10^{-3}$. Since $\Pr(Z \leq -3) = 1.3 \times 10^{-3}$, therefore

$$\left[\frac{T - \frac{N}{2}\left(\sum_{i=1}^{k} a_i(1+\epsilon_i)\right)}{\frac{\sqrt{N}}{2}\left(\sqrt{\sum_{i=1}^{k} a_i^2(1-\epsilon_i^2)}\right)}\right] = -3 \tag{8}$$

From this, we want to find the value of the L.H.S of Equation 7. Then, equating the two R.H.S will give us an expression for N. So, from Equation 8, we get

$$T - \frac{N}{2}\left(\sum_{i=1}^{k} a_i\right) = -3 \times \frac{\sqrt{N}}{2}\left(\sqrt{\sum_{i=1}^{k} a_i^2(1-\epsilon_i^2)}\right) + \frac{N}{2}\left(\sum_{i=1}^{k} a_i\epsilon_i\right)$$

$$\implies \quad \frac{T - \frac{N}{2}\left(\sum_{i=1}^{k} a_i\right)}{\frac{\sqrt{N}}{2}\left(\sqrt{\sum_{i=1}^{k} a_i^2}\right)} = \sqrt{N}\left(\frac{\sum_{i=1}^{k} a_i\epsilon_i}{\sqrt{\sum_{i=1}^{k} a_i^2}}\right) - 3 \times \left(\sqrt{\frac{\sum_{i=1}^{k} a_i^2(1-\epsilon_i^2)}{\sum_{i=1}^{k} a_i^2}}\right)$$
$$= \sqrt{N} \cdot A - 3B. \tag{9}$$

For simplification, we use the notations $A = \left(\frac{\sum_{i=1}^{k} a_i\epsilon_i}{\sqrt{\sum_{i=1}^{k} a_i^2}}\right)$ and $B = \left(\sqrt{\frac{\sum_{i=1}^{k} a_i^2(1-\epsilon_i^2)}{\sum_{i=1}^{k} a_i^2}}\right)$. This gives a relation between T and N.

Formulation of N

We equate the R.H.S. of Equation 7 with Equation 9, and get the following:

$$\begin{aligned} & \sqrt{\alpha \log 4} = \sqrt{N} \cdot A - 3B \\ \implies \quad & \sqrt{N} \cdot A = \sqrt{\alpha \log 4} + 3B \\ \implies \quad & N = \left(\frac{\sqrt{\alpha \log 4} + 3B}{A}\right)^2. \end{aligned} \tag{10}$$

Since ϵ_i's are biases, those are fractions between 0 and 1. In order to reduce the value of N, we have to choose a_i's in such a way that the denominator A increases. So, we aim to maximize the function.

$$A(a_1, a_2, \cdots, a_k) = \left(\frac{\sum_{i=1}^{k} a_i\epsilon_i}{\sqrt{\sum_{i=1}^{k} a_i^2}}\right). \tag{11}$$

Cauchy-Schwarz Inequality states that for all real numbers u_i and v_i we have

$$\begin{aligned} & \left(\sum_{i=1}^{k} u_i v_i\right) \leq \left(\sqrt{\sum_{i=1}^{k} u_i^2}\right)\left(\sqrt{\sum_{i=1}^{k} v_i^2}\right). \\ & \frac{\left(\sum_{i=1}^{k} u_i v_i\right)}{\left(\sqrt{\sum_{i=1}^{k} u_i^2}\right)} \leq \left(\sqrt{\sum_{i=1}^{k} v_i^2}\right). \end{aligned} \tag{12}$$

We know equality holds if and only if $\frac{u_i}{v_i} = h, \quad \forall \ 1 \leq i \leq k$, for some constant $h \in \mathbb{R}^+$.

If we replace the values of u_i and v_i in the Equation 12 with a_i and ϵ_i, respectively, we find that $\left(\frac{\sum_{i=1}^{k} a_i\epsilon_i}{\sqrt{\sum_{i=1}^{k} a_i^2}}\right) \leq \left(\sqrt{\sum_{i=1}^{k} \epsilon_i^2}\right)$ and equality holds if and only if $\frac{a_i}{\epsilon_i} = h, \quad \forall \ 1 \leq i \leq k$, for some constant $h \in \mathbb{R}^+$.

Hence the maximum value of $A = \left(\frac{\sum_{i=1}^{k} a_i \epsilon_i}{\sqrt{\sum_{i=1}^{k} a_i^2}} \right)$ is $\left(\sqrt{\sum_{i=1}^{k} \epsilon_i^2} \right)$, where all ϵ_i's are constant. This maximum value of A is attained if and only if $\frac{a_i}{\epsilon_i} = h, \quad \forall \ \ 1 \leq i \leq k$, for some constant $h \in \mathbb{R}^+$.

Therefore, value of N will be minimum if $\frac{a_i}{\epsilon_i} = h, \ \ \forall \ \ 1 \leq i \leq k$, for some constant $h \in \mathbb{R}^+$. In particular, if we take $h = 1$ i.e., $a_i = \epsilon_i \ \ \forall \ \ 1 \leq i \leq k$, the value of N is given by

$$N = \left\lceil \frac{\sqrt{\alpha \log 4} + 3 \times \left(\sqrt{1 - \frac{\sum_{i=1}^{k} \epsilon_i^4}{\sum_{i=1}^{k} \epsilon_i^2}} \right)}{\left(\sqrt{\sum_{i=1}^{k} \epsilon_i^2} \right)} \right\rceil^2 . \tag{13}$$

Using this value of N, we can reduce the value of complexity given in Equation 5.

3.2 Generation of Samples for Each $\mathcal{ID}$

In our new approach, since we exploit k Input Difference positions separately, and for each of them, we need N pairs of initial states X and X', apparently, the data complexity should be kN. But, in our technique, we will use the same initial states in multiple $\mathcal{ID}$s. Let X be a random state. Suppose, for each of $i = 1, 2, \cdots k$, $e_i \in \{0, 1\}$ and $X_{(e_1, e_2, \cdots e_k)}$ denotes the state where e_i is the value at the position $\mathcal{ID}_i$, and the remaining bits are same as X. For example, $X_{(0,0,\cdots 0)}$ is the matrix where the values at all the $\mathcal{ID}$ positions are 0, and the remaining are the same as X. So, $X_{(e_1, e_2, \cdots e_k)}$ can be 2^k different possible states, out of which X itself is one. Let S be the set of all such $X_{(e_1, e_2, \cdots e_k)}$. Let us assume that we have the output keystreams corresponding to each of these $X_{(e_1, e_2, \cdots e_k)}$, and are denoted by $Z_{(e_1, e_2, \cdots e_k)}$.

Now, let us consider the input difference at $\mathcal{ID}_1$. For all possible values of $e_2, e_3 \cdots e_k$, $X = X_{(0, e_2, \cdots, e_k)}$ and $X' = X_{(1, e_2, \cdots, e_k)}$ forms a pair with the difference at $\mathcal{ID}_1$. Therefore, we can form 2^{k-1} such pairs. For any other input difference, say $\mathcal{ID}_i$, we can similarly form 2^{k-1} such X, X' pairs from the same set S. Therefore, if we only have the encryption corresponding to the 2^k possible $X_{(e_1, e_2, \cdots e_k)}$ matrices, for each of the $\mathcal{ID}$ positions, we can actually achieve 2^{k-1} pairs of X, X' and their corresponding (Z, Z'). Even for input difference at two positions simultaneously, we can achieve 2^{k-1} such pairs. So, to achieve N pairs of matrices for each $\mathcal{ID}$, we follow the following procedure: We choose $\frac{N}{2^{k-1}}$ randomly chosen initial states. For each of them, we form the set S_X of 2^k possible matrices $X_{(e_1, e_2, \cdots e_k)}$. We find the encryption of all these $\frac{N}{2^{k-1}} \times 2^k = 2N$ matrices. Since each S_X produces 2^{k-1} pairs for each $\mathcal{ID}$, we will achieve N pairs in total for each of the $\mathcal{ID}$s. Since we have to encrypt $2N$ matrices in total, the data complexity is N.

4 Analysing the PNBs and Choice of Assigned Vectors

For a general discussion, let us denote the function computing from the ciphertext backward by f. This function takes as input a ciphertext (or a pair of ciphertexts) and a key guess to compute the internal state that serves as a distinguisher.

$$f : \mathbb{F}_2^n \times \mathbb{F}_2^\kappa \to \mathbb{F}_2$$

$$(x, k) \mapsto f(x, k)$$

Now, the hope is that one can compute this function f without actually having knowledge of all key bits. The easiest case is if the function f is independent of some bits of k. This can be captured by the notion of linear structures. A Boolean function g has a linear structure β for a non-zero vector $\beta \in F_2^n$ if

$$g(x) + g(x + \beta) = 0$$

for all x. In the case where β is a vector in the canonical basis, this corresponds to being independent of a bit. The set of all linear-structures

$$L(g) = \{\beta \in F_2^n \mid \beta \text{ is a linear structure}\}$$

actually forms a vector space. In the case of f as above, if f is independent of the first bit of k and the second bit of k, it is also independent of both bits.

We now split the key into two parts: the (probabilistic) independent bits or vector space and a complement for it. That is, now f takes three inputs: the ciphertext, the significant part of the key k_s, and the neutral part of the key k_n.

$$\begin{aligned} f : \mathbb{F}_2^n \times \mathbb{F}_2^{\kappa_1} \times \mathbb{F}_2^{\kappa_2} &\to \mathbb{F}_2 \\ (x, k_s, k_n) &\mapsto f(x, k). \end{aligned}$$

In the case of linear-structures, it holds that

$$f(x, k_s, k_n) = f(x, k_s, 0)$$

for all ciphertexts x and keys k_s. That translates to the obvious strategy to set the insignificant, neutral bits to zero and only guess the significant part. Setting to any other value than zero clearly does not make any difference, simply due to the definition of a linear structure.

However, the situation changes if the bits are not neutral but only probabilistic neural bits.

4.1 **Detecting Probabilistic Neutral Subspaces**

In order to detect a subspace of the key space that does not have to be guessed, one usually relies on detecting probabilistic neutral bits in the first step. Here, one computes

$$\Delta(\beta_i) = |\{x \mid f(x, k) + f(x, k + \beta_i) = 0\}|$$

for all the standard unit vectors $\beta_i \in \mathbb{F}_2^\kappa$. All the ones that are above a certain threshold are collected into the set of probabilistic neutral bits, and the key space is decomposed into this space (dimension κ_2) and a space of the significant bits (dimensions κ_1).

Now, for a given key (k_s, k_n) and a fixed vector $\beta \in \mathbb{F}_2^{\kappa_2}$ we hope that

$$f(x, k_s, \beta) = f(x, k_s, k_n)$$

with good probability. That is

$$T(k_s, k_n, \beta) = |\{x \in \mathbb{F}_2^n \mid f(x, k_s, \beta) = f(x, k_s, k_n)\}|$$

should be close to 2^n.

What is (usually experimentally) computed is the average of those counters, averaged over all keys and all fixed constants. This corresponds to the backward bias that is then taken into account when computing the complexity of the entire attack. Note that there is no general way to deduce the value of the backward bias given the thresholds $\Delta(\beta_i)$ as those thresholds simply do not provide the full picture of what happens if more than a single bit is flipped.

Value of Assigned Vector (β) Matters

As we do not know and have no control over k_s and k_n, it makes sense to consider the average over all keys, i.e., consider

$$C(\beta) = \sum_{k_s, k_n} T(k_s, k_n, \beta).$$

However, even so, the constants β should not have a strong influence (as otherwise, they are not probabilistic neutral); it is not clear that

$$C(\beta) = C(0)$$

that is, it might well be that fixing to other constants than zero for computing backward might be beneficial for the attacker.

Example: Consider the following example, where x, k are 8-bit numbers, the addition operation is modulo 2^8, and we are interested in computing the most significant bit of y.

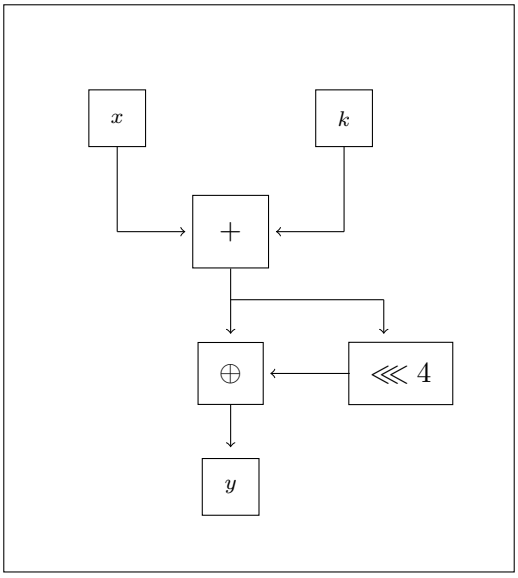

Fig. 3: Function $f(x, k)$.

Here f corresponds to the function

$$f(x, k) = ((x + k) \oplus ((x + k) \lll 4)) \, [7] \tag{14}$$

Bit no. (k_i)	0	1	2	3	4	5	6	7
Bias	0.765	0.531	0.062	-0.875	0.750	0.500	0.000	-1.000

Table 3: Bias Observed for the Function $f(x, k)$ for Each Bit k_i.

In the above experiment, we choose three bits $\{0, 1, 4\}$ as PNBs. For these three bits, we express $\beta = (k_0, k_1, k_4) \in \mathbb{F}_2^3$ in decimal form and their corresponding backward bias in Table 4.

β	0	1	2	3	4	5	6	7
Backward Bias	0.57	0.57	0.66	0.66	0.66	0.66	0.57	0.57

Table 4: Backward Bias Observed for Different Assigned Vectors β for the PNBs k_0, k_1, k_4.

4.2 Improving the Backward Bias by Proper Choice of Assigned Vector

We observe that the PNBs are located in the form of blocks, i.e., collections of consecutive bits. For example, the PNB set for the 8-round of 256-bit key version of Salsa20 has many PNB set blocks, which we observe in the list given in subsection 5.2. As examples, $\{25, 26, 27, 28, 29, 30, 31\}$ and $\{164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176\}$ are two blocks. The first block is of size 7 at X_0 from $X_0[26]$ to $X_0[31]$. Similarly, the second block is of size 13 at X_6 from $X_6[4]$ to $X_6[16]$. We decompose the subspace k_s in the form of subspaces k_{s_i}, where each k_{s_i} corresponds to a block. Now, consider one such block of PNBs of size b at X_i from $X_i[j]$ to $X_i[j + b - 1]$.

In our attack, after assigning the vector β at the PNBs, we compute $Z - \tilde{X}$ and $Z' - \tilde{X}'$. From Subsection 2.2 we know that $\tilde{M} = F^{-1}(Z - \tilde{X})$ and $X^{(r)} = F^{-1}(Z - X)$. Therefore, in order to get a good backward bias, we aim to find out how closely $Z - \tilde{X}$ replicates $Z - X$. In simple words, the more the number of bits of $Z - X$ matches with $Z - \tilde{X}$, the more is the backward bias. During the subtraction $Z - \tilde{X}$, the difference between X and $\tilde{X}$ can also propagate to the bit at the position $j + b$ and onwards of $Z - \tilde{X}$ due to carry-propagation. Now, we observe that this probability of propagation varies based on the assigned values at the PNBs. So, if the assigned values at $\tilde{X}[j] \cdots \tilde{X}[j + b - 1]$ can be chosen in such a way that the probability of this propagation can be reduced, we achieve a higher backward bias. We next analyze how the values can be chosen so that this probability of propagation is minimal.

Which Values of β Minimizes the Probability of Propagation

For each of Z_i, X_i and $\tilde{X}_i$, if we consider the PNB block from j to $j + b - 1$ bit, each of them is a number between 0 to $2^b - 1$. Let us denote them z, k_n and β respectively. From attackers perspective, z and k_n are unknown, and β is decided by him/her. We aim to find which value of β would be most suitable. The difference between $Z - X$ and $Z - \tilde{X}$ would propagate to $j + b$-th bit if either $k_n > z \geq \beta$ or $\beta > z \geq k_n$.

Theorem 1 *Let for some $\beta \in \{0, 1, 2, \cdots 2^b - 1\}$, S_β=$\{$(x, z): either $x > z \geq \beta$ or $\beta > z \geq x\}$. Then $|S_\beta|$ is minimum if $\beta = 2^{b-1} - 1$ or $\beta = 2^{b-1}$, and maximum if $\beta = 0$ or $2^b - 1$.*

Proof. Possible values of (x, z) such that $x > z \geq \beta$ is ${}^{2^b-\beta}C_2$, since x, z can be any integer in the range $[\beta, 2^b - 1]$. Similarly, possible values of (x, z) such that $\beta > z > x$ is ${}^{\beta}C_2$, and possible values such that $\beta > z = x$ is β. Therefore,

$$|S_\beta| = {}^{2^b-\beta}C_2 + {}^{\beta}C_2 + \beta = \frac{(2^b - \beta)!}{(2!)(2^b - \beta - 2)!} + \frac{\beta!}{(2!)(\beta - 2)!} + \beta$$
$$= \frac{(2^b - \beta)(2^b - \beta - 1)}{2!} + \frac{\beta(\beta - 1)}{2!} + \beta$$

$$
\begin{aligned}
&= \frac{(2^b-\beta)(2^b-\beta-1)+\beta(\beta-1)+2\beta}{2} \\
&= \frac{2^{2b}-\beta\times 2^b-2^b(\beta+1)+\beta(\beta+1)+\beta(\beta-1)+2\beta}{2} \\
&= \frac{2^{2b}-2\beta\times 2^b-2^b+2\beta^2+2\beta}{2} \\
&= 2^{2b-1}-2^b\times\beta-2^{b-1}+\beta^2+\beta.
\end{aligned}
$$

We aim to find the value of β for which $|S_\beta|$ is minimum. For this, we can write it as:

$$
\begin{aligned}
|S_\beta| &= 2^{2b-1}-2^{b-1}+\beta^2-(2^b-1)\times\beta \\
&= 2^{2b-1}-2^{b-1}+\beta^2-2\beta\times\left(\frac{2^b-1}{2}\right)+\left(\frac{2^b-1}{2}\right)^2-\left(\frac{2^b-1}{2}\right)^2 \\
&= 2^{2b-1}-2^{b-1}-\left(\frac{2^b-1}{2}\right)^2+\left(\beta-2^{b-1}+\frac{1}{2}\right)^2.
\end{aligned}
$$

The term $(\beta-2^{b-1}+\frac{1}{2})^2$ is non-negative. Since β is an integer, $(\beta-2^{b-1}+\frac{1}{2})^2$ gives minimum value either at $\beta=2^{(b-1)}-1$ or at $\beta=2^{b-1}$, and at both of them $(\beta-2^{b-1}+\frac{1}{2})^2=\frac{1}{4}$. So, in both these cases, we get $\mathrm{g}(2^{(b-1)}-1)=2^{2(b-1)}$ and $\mathrm{g}(2^{b-1})=2^{2(b-1)}$. Similarly, to find the β for which $|S_\beta|$ is maximum, we focus on the term $\left(\beta-2^{b-1}+\frac{1}{2}\right)^2$. It gives the maximum value when $\beta=0$ or 2^b-1. ■

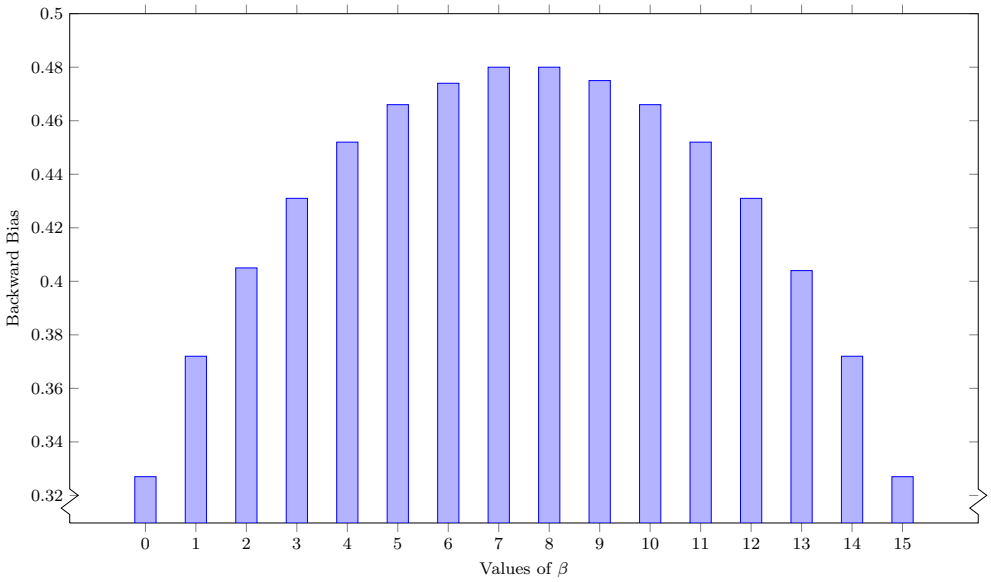

Fig. 4: Graphical Representation of the Backward Biases for Different Values of β for the PNBs $\{13, 14, 15, 16\}$.

Observation for 7-round 128-bit key version of Salsa20

We experiment on a set of four consecutive PNBs $\{13, 14, 15, 16\}$ for 7 round 128-bit key version of Salsa20 from the PNB set given in Section 5.1. We know that there can be 2^4 possible values of this PNB set, i.e., $\{0, 1, 2, \cdots 15\}$. For each of these values, we will find the value of ϵ_a (backward bias). We observe that we got a maximum backward bias for the values $7, 8$ and a minimum backward bias at $0, 15$. In this case b=4, hence from above theorem $|S_\beta|$ is minimum at $\beta = (2^{b-1} - 1) = (2^3 - 1) = 7$ or $\beta = 2^{b-1} = 2^3 = 8$.

5 Attacks on Salsa20

We exploit 6 single-bit differentials and 7 two-bit differentials in our attack, which are given in Table 5. We use the linear combination of the output differences at the positions $X_9^{(5)}[0] \oplus X_1^{(5)}[13] \oplus X_{13}^{(5)}[0]$ at the 5th round. For the 128-bit key version, we assign the $\mathcal{IV}$s randomly since there are a significant number of key bits in the input difference column.

i	$\mathcal{ID}$	Bias (ϵ_{d_i})	
		Chosen $\mathcal{IV}$	Random $\mathcal{IV}$
1	$(7, 0)$	-0.2334	-0.1147
2	$(7, 1)$	0.0876	0.0334
3	$(7, 7)$	-0.0970	-0.0459
4	$(7, 18)$	0.0914	0.0449
5	$(7, 20)$	-0.0882	- 0.0442
6	$(7, 30)$	0.2194	0.0542
7	$(7, 0), (7, 1)$	-0.103	-0.0568
8	$(7, 0), (7, 18)$	-0.062	-0.0171
9	$(7, 0), (7, 20)$	0.095	0.0293
10	$(7, 0), (7, 30)$	-0.131	-0.0178
11	$(7, 1), (7, 20)$	-0.133	-0.0244
12	$(7, 1), (7, 30)$	0.072	0.0077
13	$(7, 7), (7, 30)$	-0.060	-0.0080

Table 5: Biases Observed at the $\mathcal{OD}$ $X_9^{(5)}[0] \oplus X_1^{(5)}[13] \oplus X_{13}^{(5)}[0]$ for Different $\mathcal{ID}$ Positions.

5.1 128-bit key version of Salsa20

Attack on 7-round

Using the PNB procedure after assigning threshold $\gamma = 0.17$ we achieve 53 PNBs, if we assign $\gamma = 0.14$ we get 2 more PNBs $\{40, 58\}$. We get the best result if we add only $\{40\}$ as a PNB along with the first 53 PNBs. The list is given below:

{0, 1, 13, 14, 15, 16, 19, 20, 27, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 44, 57, 62, 63, 64, 76, 77, 78, 84, 85, 86, 87, 88, 89, 90, 91, 96, 97, 101, 107, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127}.

Complexity: For these 54 PNBs we have the backward bias $\epsilon_a = 0.0026$. For calculating the value of N we use Equation 13. In these equations ϵ_i is the product of ϵ_a and ϵ_{d_i}, i.e., $\epsilon_i = \epsilon_{d_i} * \epsilon_a \ \ \forall\ 0 \leq i \leq 13$. All the values of ϵ_{d_i} are mentioned in Table 5. For $\alpha = 30$, Equation 13 gives $N = 2^{28.77}$ and from Equation 5 we have time complexity as $2^{102.82}$.

Attack on 7.5-round

For finding the first-ever set of PNBs for 7.5 rounds of Salsa20, we use the PNB procedure. We assign the bias limit to be $\gamma = 0.1$ to get the 27 PNBs mentioned below. We are providing a GitHub link [11] of a C program ('PNB Set 7.5.c') for finding the PNB set.

{27, 35, 36, 37, 38, 39, 44, 76, 77, 84, 85, 86, 87, 88, 89, 90, 96, 101, 119, 120, 121, 122, 123, 124, 125, 126, 127}.

Complexity: We have the backward bias $\epsilon_a = 0.122$ for these 27 PNBs. Using Equation 13, we have $N = 2^{23.06}$ for $\alpha = 7$, and time complexity is $2^{124.22}$.

5.2 8-round of 256-bit key version of Salsa20

For the 256-bit key version of Salsa20, we use the chosen $\mathcal{IV}$ approach since we have only 2 key bits in the Input Difference column. As shown in [9], we will have $\mathcal{IV}$s available, which will give the minimum difference after the first round for all possible values assigned at the PNBs. For the chosen $\mathcal{IV}$ approach, the biases corresponding to each Input Difference are given in Table 5. The PNB set we use is the set used in [9], which is given as follows:

{25, 26, 27, 28, 29, 30, 31, 39, 70, 71, 72, 107, 119, 120, 121, 122, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 209, 210, 211, 212, 213, 224, 225, 241, 242, 243, 244, 245, 246, 255}.

Complexity: Assigning the values in each block of PNBs in the form 2^{b-1} or $2^{(b-1)} - 1$, (b is the size of the block), we achieve backward bias $\epsilon_a = 0.0013$. This value of ϵ_a is multiplied by each ϵ_{d_i} to get the value of ϵ_i. Therefore for $\alpha = 20$, we achieve $N = 2^{27.56}$ and the final complexity is $2^{240.62}$.

6 Conclusion

After the attack of [1], all the attacks on Salsa20 afterward provided some minor improvements in the complexity only. None of the attack was able to find a partial or full round extension to the next round. We propose a new direction in the approach of differential-linear attacks on Salsa20 and produce a new attack on this cipher after 5 years. Importantly, we provided a half-round improvement in the 128-bit key version of the cipher. We hope that this work will regain the flow of cryptanalysis of this cipher, which is important considering the influence of the design principle of Salsa20 on future ciphers. Also, this approach has the potential to find more applications in other ARX-based designs.

Acknowledgement

The authors acknowledge the financial support provided by the Department of Science and Technology through a Start-Up Research Grant (SRG/2020/000523).

References

1. Aumasson, J., Fischer, S., Khazaei, S., Meier, W., Rechberger, C.: New features of latin dances: Analysis of Salsa20, ChaCha and Rumba. In: FSE 2008, Revised Selected Papers. LNCS, vol. 5086, pp. 470–488. Springer (2008). https://doi.org/10.1007/978-3-540-71039-4_30.
2. Bernstein, D. J. : Salsa20/8 and Salsa20/12. Technical Report 2005/025, eSTREAM, ECRYPT Stream Cipher Project (2005). https://cr.yp.to/snuffle/812.pdf.
3. Bernstein, D. J. : Salsa20. Technical Report 2005/025, eSTREAM, ECRYPT Stream Cipher Project (2005). https://www.ecrypt.eu.org/stream/papers.html.
4. Bernstein, D.J.: The Salsa20 family of stream ciphers. In: Robshaw, M.J.B., Billet, O. (eds.). New Stream Cipher Designs - The eSTREAM Finalists, LNCS, vol. 4986, pp. 84–97. Springer (2008). https://doi.org/10.1007/978-3-540-68351-3_8.
5. Biham, E., Shamir, A.: Differential cryptanalysis of DES-like cryptosystems. In:Menezes, A. , Vanstone, S. A. (eds.) CRYPTO. Lecture Notes in Computer Science, vol. 537, pp. 2–21. Springer (1990). https://doi.org/10.1007/3-540-38424-3_1.
6. Choudhuri, A. R., Maitra, S.: Significantly improved multi-bit differentials for reduced round Salsa20 and ChaCha. IACR Trans. Symmetric Cryptol, 261–287 (2016). https://doi.org/10.13154/tosc.v2016.i2.261-287.
7. Crowley, P.: Truncated differential cryptanalysis of five rounds of Salsa20. In: SASC 2006 – Stream Ciphers Revisited (2006). http://eprint.iacr.org/2005/375.
8. Deepthi K., Singh K.: Cryptanalysis of Salsa20 and ChaCha: revisited. In: International Conference on Mobile Networks and Management (2018). https://doi.org/10.1007/978-3-319-90775-8_26.
9. Dey, S., Sarkar, S.: Improved analysis for reduced round Salsa20 and ChaCha. Discrete Applied Mathematics 227, 58–69 (2017). https://doi.org/10.1016/j.dam.2017.04.034.
10. Fischer, S., Meier, W., Berbain, C., Biasse, J.-F., Robshaw, MJB: Non randomness in eSTREAM candidates Salsa20 and TSC-4. In: Barua, R., Lange, T. (eds.) INDOCRYPT 2006. LNCS, vol. 4329, pp. 2–16. Springer, Heidelberg (2006). https://doi.org/10.1007/11941378_2.
11. https://github.com/SharmaNitinKumar/Salsa.git
12. Langford, S. K., Hellman, M. E.: Differential-linear cryptanalysis. Annual Int. Cryptology Conf., California, USA, 1994, pp. 17–25. https://doi.org/10.1007/3-540-48658-5_3.
13. Maitra, S., Paul, G., Meier, W.: Salsa20 cryptanalysis: new moves and revisiting old styles. In: 9th International Workshop on Coding and Cryptography, WCC 2015 (2015). https://eprint.iacr.org/2015/217.
14. Maitra, S.: Chosen IV cryptanalysis on reduced round ChaCha and Salsa20. Discrete Applied Mathematics 208, 88–97 (2016). https://doi.org/10.1016/j.dam.2016.02.020.
15. Matsui, M., Yamagishi, A.: A new method for known plaintext attack of FEAL cipher. EUROCRYPT. Lecture Notes in Computer Science,vol. 658, pp. 81–91. Springer (1992). https://doi.org/10.1007/3-540-47555-9_7.
16. Shi, Z., Zhang, B., Feng, D., Wu, W.: Improved key recovery attacks on reduced round Salsa20 and ChaCha. ICISC 2012. LNCS, vol. 7839, pp. 337–351. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-37682-5_24.
17. Tsunoo, Y., Saito, T., Kubo, H., Suzaki, T., Nakashima, H.: Differential Cryptanalysis of Salsa20/8. eSTREAM, ECRYPT Stream Cipher Project (2007). http://www.ecrypt.eu.org/stream/papersdir/2007/010.pdf.