Advances in Mathematics of Communications
Volume 13, No. 4, 2019, 689–704
doi:10.3934/amc.2019041

# REVISITING DESIGN PRINCIPLES OF SALSA AND CHACHA

Sabyasachi Dey, Tapabrata Roy and Santanu Sarkar

Department of Mathematics
Indian Institute of Technology Madras
Sardar Patel Road, Chennai 600036, India

Abstract. Salsa and ChaCha are well known names in the family of stream ciphers. In this paper, we first revisit the existing attacks on these ciphers. We first perform an accurate computation of the attack complexities of the existing technique instead of the estimation used in previous works. This improves the complexity by some margin. The differential attacks using probabilistic neutral bits against ChaCha and Salsa involve two probability biases: forward probability bias ($\epsilon_d$) and backward probability bias ($\epsilon_a$). In the second part of the paper, we suggest a method to increase the backward probability bias, which helps reduce the attack complexity. Finally, we focus on the design principle of ChaCha. We suggest a slight modification in the design of this cipher as a countermeasure of the differential attacks against it. We show that the key recovery attacks proposed against ChaCha will not be effective on this modified version.

## 1. Introduction

Salsa20, a stream cipher submitted by D. J. Bernstein [2] to eSTREAM project in 2008, was selected as Phase 3 design for software, by receiving highest votes. Though original Salsa has 20 rounds, the submitted version in eSTREAM is of 12 rounds.

From the beginning, Salsa has been seriously analysed. A few differential attacks have been proposed against Salsa. The basic idea of differential attack is to put some input difference at the initial stage and obtain a bias in the output after a number of rounds. In 2005, Crowley [5] proposed the first differential attack breaking the 5 rounds Salsa with time complexity $2^{165}$. Then in Indocrypt 2006, 6 rounds version of Salsa was attacked by Fischer et al. [7] with time complexity $2^{177}$ which was further extended to 7 rounds by Tsnunoo et al. [12] with around $2^{190}$ trials. Next in FSE 2008, an improvement in the backward inversion to 4 rounds was suggested by Aumasson et al. [1]. This led to an attack on 8 rounds Salsa with $2^{251}$ time complexity using the concept of probabilistic neutral key bits (PNB). Shi et al. [11] improved this attack in ICISC 2012 reducing the complexity to $2^{250}$. Later, Maitra et al. [9] provided give some new ideas in this attack and reduced the complexity to $2^{247.2}$ for 8-round Salsa. Next, the complexity was further improved upto $2^{245.5}$ by Maitra [8]. Again, Choudhuri et al. [4] improved it up to $2^{244.9}$ using multibit approach. Recently, the complexity is improved upto $2^{243.7}$ by Dey et al. [6].

ChaCha [3], a variant of Salsa, was published by Bernstein in 2008, to achieve better performance. Aumasson et al. [1] attacked the 256 bit version of ChaCha upto 7th round. Later, this complexity was further improved to $2^{238.9}$ by Maitra [8]. Next, instead of single bit output, Choudhuri et al. [4] suggested to use multiple

2010 *Mathematics Subject Classification:* Primary: 11Y05; Secondary: 94A60.
*Key words and phrases:* Stream cipher, ChaCha, cryptanalysis, new design.

bit output and improved the complexity to $2^{237.6}$. Recently, this complexity was further improved upto $2^{235.2}$ by Dey et al. [6].

The design of Salsa family uses ARX operation (addition-rotation-Xor) in its update function. The combination of these three operations brings about huge diffusion in the entries very fast, which is the backbone of the security of these ciphers. Due to this non-linearity, this design pattern possesses strong security against linear approximation attacks, cube attacks, algebraic attacks etc. To the best of our knowledge, diffential attack is the only attack which has been successfully applied in key recovery attack against these two ciphers upto few rounds. In recent times, ChaCha has been selected as an encryption algorithm by Google, which has made these ciphers one of the major areas of research. Since this family of ciphers has great contribution in the market, both the detailed security analysis of this design principle and the possible improvement of security further are very important issues. The differential attacks against Salsa and ChaCha are applicable upto 8th and 7th round respectively. These attacks are mainly based on an idea of partitioning the keybits into significant key bits (non-PNB) and insignificant key bits (PNB) and searching them separately in two different steps.

In this paper, we provide a detailed study of the differential attack using probabilistically neutral bits and provide an accurate estimation of the attack complexity of the attacks available so far. Also, we provide an improvement in the attack method by finding out a set of values that can be asigned to the PNB's while searching the non-PNB's. Finally, we suggest a small tweak in the design principle of ChaCha and show that this tweak can defend all the key recovery attacks based on the idea of PNB's without harming the security of the cipher against any other kind of attacks.

**Notations:** We follow the following notations throughout the paper.

- $X_i$ will denote the word in $i$-th cell of the matrix $X$.
- $X_{i,j}$ will denote the $j$-th bit of $X_i$, starting from right ( $X_{i,0}$ is the least significant bit of $X_i$.). So $X_{i,j}$ denotes $j$-th bit of $i$-th cell of $X$. We represent it also by 'position $(i, j)$'.
- $X'$ will denote the matrix obtained by inputting a difference at an intended position of $X$.
- $X^r$ will denote the matrix obtained after $r$-th round of $X$.
- $X_i^r$ will denote the $i$-th word of $X^r$.
- $X_{i,j}^r$ will denote the $j$-th bit of $X_i^r$.
- $\Delta_{i,j}^r$ will represent $X_{i,j}^r \oplus X_{i,j}'^r$. In particular for $r = 0$, we use $\Delta_{i,j}^0 = \Delta_{i,j}$.
- $|X|$ will denote the number of elements in the set $X$.

**Organisation of the paper:**

- In Section 2.1 and Section 2.2 we have explained the structure of Salsa and ChaCha in short.
- In Section 3, we revisit the differential attack idea against this family of ciphers. We go through the error estimation in detail and improve the complexity by more accurate estimation. We revisit an attack idea called chaining distinguisher and find some weakness of this strategy.
- In Section 4, we discuss a procedure to find the values that can be assigned to the PNB's instead of some random values during searching the non-PNB's. We provide the full list of values in all PNB's and discuss the improvement in attack by experiments in 4.

- In Section 5, we suggest a small modification in the design of ChaCha. This change can defend all the differential attacks based on probabilistic neutral bits which have been proposed so far against this family of ciphers. We also provide a detailed discussion on the security of this tweak against other attacks.

## 2. Structure of the ciphers

2.1. **Structure of Salsa:** Salsa involves a $4 \times 4$ matrix, whose each cell is of 32 bits. The 16 cells of the matrix include 4 constant cells, 8 key cells, 2 IV cells and 2 counter cells. The 256 bit Salsa20 divides the 256 bit input key into 8 parts, each containing 32 bits and assigned to 8 cells of the matrix. 128 bit Salsa20 replicates the key to another copy of 128 bit and makes it 256 bits, and then use the same opearions as Salsa256.

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

In the above matrix, $c_0 = 0x61707865, c_1 = 0x3320646e, c_2 = 0x79622d32, c_3 = 0x6b206574$ are the constant cells, $k_i$ are key cells, $v_i$ are IV cells and $t_i$ are counter cells.

**Quarterround Function**: This is a set of nonlinear functions operating on a 4-tuple $(a, b, c, d)$ to give an output of 4-tuple $(a, b, c, d)$, where each of $a$, $b$, $c$ and $d$ is a 32 bit word. The function is defined as:

$$b = b \oplus ((a + d) \lll 7)$$

$$c = c \oplus ((b + a) \lll 9)$$

$$d = d \oplus ((c + b) \lll 13)$$

$$a = a \oplus ((d + c) \lll 18).$$

Note that here $'+'$, $\oplus$ and $\lll$ signs denote respectively the addition modulo $2^{32}$, the usual XOR operation and the left cyclic rotation respectively.

At first, we apply quarterround function to each column (from 1st to 4th) of the matrix. This is called a columnround. Each columnround is followed by a rowround, where this function is applied to the respective rows. Here in columnround, the order of the cells taken is respectively $(X_0, X_4, X_8, X_{12}), (X_5, X_9, X_{13}, X_1), (X_{10}, X_{14}, X_2, X_6)$ and $(X_{15}, X_3, X_7, X_{11})$. Each set of columnround and rowround is together called a doubleround.

In Salsa, 20 rounds are performed. $R$ denotes the total number of rounds. So, $X^0$ and $X^R$ are respectively the initial and the final matrices. Clearly, for Salsa20, $R = 20$. We finally get an output keystream of 512 bits as $Z = X + X^R$.

Since the quarterround function is reversible, from any round of Salsa we can get back the matrix of the previous round. We call this reverse algorithm ReverseSalsa and each round of the reverse algorithm reverseround so that application of reverseround on $X^{r+1}$ gives $X^r$. Thus, using ReverseSalsa algorithm, we can get back the initial matrix $X^0$ from the final matrix $X^R$.

2.2. STRUCTURE OF CHACHA: As ChaCha is a variant of Salsa, it has a structure almost similar to that of Salsa. Here, in the initial matrix, the positions of the cells are as follows

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

Here $c_0 = 0x61707865, c_1 = 0x3320646e, c_2 = 0x79622d32$ and $c_3 = 0x6b206574$. Also $k_i, v_i$ and $t_i$'s denote the key cells, IV cells and counter cells respectively.

**Round Function:** In ChaCha, the nonlinear round functions are different and more complicated than that of Salsa. Here, the quarterround function is given by:

$$a = a + b,\ d = ((d \oplus a) \lll 16),$$
$$c = c + d,\ b = ((b \oplus c) \lll 12),$$
$$a = a + b,\ d = ((d \oplus a) \lll 8),$$
$$c = c + d,\ b = ((b \oplus c) \lll 7).$$

The way of application of the nonlinear function is also different here. Unlike columnround and rowround in Salsa, ChaCha applies the function along column and diagonals. In case of columns the order is $(X_0, X_4, X_8, X_{12}), (X_1, X_5, X_9, X_{13})$, $(X_2, X_6, X_{10}, X_{14})$ and $(X_3, X_7, X_{11}, X_{15})$. In case of diagonals the order is $(X_0, X_5, X_{10}, X_{15})$, $(X_1, X_6, X_{11}, X_{12})$, $(X_2, X_7, X_8, X_{13})$ and $(X_3, X_4, X_9, X_{14})$. Similar to Salsa, each round in ChaCha is also reversible.

## 3. IDEA OF ATTACK ON SALSA AND CHACHA

So far various differential attacks have been proposed against Salsa and ChaCha. The basic idea of differential attacks is to input a difference at any intended bit $X_{i,j}$ of the initial matrix $X$ and achieve a new matrix $X'$. Then we run the algorithm on both the matrices by few rounds and find some correlation between $X^r$ and $X'^r$. This correlation is called a distinguisher. Similarly, from the final state, we can come backward by ReverseSalsa. In this attack method, Aumasson et al. [1] introduced a new idea named Probabilistic Neutral Bits (PNB's). We first explain idea of Probabilistic Neutral Bits or PNB as given in [1, 9]. The aim of this idea is to reduce the complexity of searching 256 bits of the unknown key by partitioning the set of keybits into two parts: Significant Keybits and insignificant Keybits.

**Detailed explanation:** Applying an input difference at an IV position $(i, j)$ ($j$-th bit of $i$-th word ), two matrices $X$ and $X'$ are achieved. Now the algorithm is performed on both $X$ and $X'$ by some $r$ rounds and a correlation between $X^r$ and $X'^r$ is found at some position $(p, q)$. Suppose $X^r_{p,q}$ and $X'^r_{p,q}$ be the entries at position $(p, q)$. Now, $\Delta^r_{p,q} = X^r_{p,q} \oplus X'^r_{p,q}$. This $\Delta^r_{p,q}$ shows a high bias $\epsilon_d$ i.e., $\Pr(\Delta^r_{p,q} = 1 \mid \Delta_{i,j} = 1) = \frac{1}{2}(1 + \epsilon_d)$ ; $\epsilon_d$ being the measure of the bias of the output difference. This bias works as the distinguisher of the cipher.

Now, using this distinguisher, a key recovery attack can be formed using the idea of probabilistically neutral bits. Let us give a brief idea of PNB's. Suppose, $R$ is the total number of rounds, and $Z = X + X^R$ and $Z' = X' + X'^R$. In $X$ and $X'$ , a particular key bit position $k$ is complemented to yield the states $\overline{X}$ and $\overline{X'}$. Next, one can reverse the states $Z - \overline{X}$ and $Z' - \overline{X'}$ by $R - r$ rounds to yield the states $Y$ and $Y'$ respectively. Let $\Gamma_{p,q} = Y_{p,q} \oplus Y'_{p,q}$. If the bias in the event $(\Gamma_{p,q} = 0)$ is high, then $k$ is considered to be a probabilistic neutral bit (PNB). A

predetermined threshold probability bias $\gamma$ is already considered to identify PNB's. Using this idea, the keybits are partitioned into two sets: Probabilistic neutral bits (PNB) and non-Probabilistically neutral bits (non-PNB). After this, in the main attack, we aim to find the values of the non-PNB's first. For this purpose, some random values are assigned to the PNB's. The attacker tries to guess the values of the non-PNB's. Suppose, the matrix obtained by guessing the non-PNB's and assigning random values to PNB's are $\tilde{X}$ and $\tilde{X}'$. Reverse algorithm is applied on $Z-\tilde{X}$ and $Z'-\tilde{X}'$ by $R-r$ rounds to obtain $\tilde{Y}$ and $\tilde{Y}'$. If the guess of non-PNB's is correct, $\tilde{\Gamma}_{p,q} = \tilde{Y}_{p,q} \oplus \tilde{Y}'_{p,q}$ gives a high bias $\epsilon$. The bias of the event $\tilde{\Gamma}_{p,q} = \Delta_{p,q}$ is called backward bias and denoted by $\epsilon_a$. The bias $\epsilon$ can be approximated by $\epsilon_d \cdot \epsilon_a$. Thus, we now achieve the  values non-PNB set. Suppose the size of non-PNB set is $n$. Achieving the values of the non-PNBs, we fix those values try to guess the values for PNBs. Instead of an exhaustive search over all possible $2^{256}$ values for the keybits, this idea helps to reduce the search time complexity.

**Improving the Complexity and the Error Estimation:** Now we revisit the estimation provided in [1] and perform an accurate computation of the complexity, which improves the attack by some margin. We have $2^n$ possible sequences of random values for the $n$ non-PNB's among which only 1 sequence is correct and remaining $2^n - 1$ sequences are incorrect. Here, we consider the null hypothesis and alternative hypothesis respectively as:

1. $H_0$ : chosen sequence is incorrect.
2. $H_1$: chosen sequence is the correct one.

So, $2^n - 1$ sequences satisfy the null hypothesis and only 1 sequence satisfies the alternative hypothesis.

Suppose we make our decision based on $N$ keystream bits $Z_1, Z_2, \ldots, Z_N$. We focus on the probability of the event $\tilde{\Gamma}_{p,q} = 0$. If the guess of $n$ non-PNB's are correct, then this probability is $p_1 = \frac{1}{2}(1+\epsilon)$. On the other side, if the guess is wrong, the probability is $p_0 = \frac{1}{2}$. So, this experiment on each $Z_i$ is basically a Bernoulli trial where getting $\tilde{\Gamma}_{p,q} = 0$ can be considered to be 'success' and $\tilde{\Gamma}_{p,q} = 1$ to be 'failure'. Repeating this experiment on $N$ keystream bits constructs a binomial distribution with the parameters $p_0/p_1$ (based on the correctness of the guess of non-PNB's) and $N$. So, if the guess of $n$ non-PNB's is incorrect, the mean of the binomial distribution is $\mu_0 = Np_0 = \frac{N}{2}$ and variance is $\sigma_0 = Np_0(1-p_0) = \frac{N}{4}$. On the other side, for correct guess, probability of success $p_1 = \frac{1}{2}(1+\epsilon)$. So, mean is $\mu_1 = \frac{N}{2}(1+\epsilon)$ and variance is $\sigma_1 = \frac{N}{4}(1+\epsilon)(1-\epsilon)$. Now, from central limit theorem we know that for large $N$, both these binomial distributions can be approximated by normal distributions with same means and variances. Therefore, the corresponding normal distribution functions can be given by $f(x) = \frac{1}{\sqrt{2\pi}\sigma_i} e^{-\frac{(x-\mu_i)^2}{2\sigma_i^2}}$, where $i = 1$ for correct guess.

Let us denote the distribution associated to the correct guess as distribution 1 and the one associated with the wrong guess as distribution 2. In the hypothesis testing, at first, a threshold $T$ is decided. Among the $N$ Bernoulli trials, if the number of successes is more than $T$, we choose the alternative hypothesis $H_1$ to be true. In other words, in our $N$ trials, say $t$ be the number of times the event $(\tilde{\Gamma}_{p,q} = 0)$ has occurred, which we declare as 'success'. Then, if $t \geq T$, then we declare the alternate hypothesis to be true, i.e., the guess of non-PNB's is correct. Otherwise, if $t < T$, the null hypothesis $H_0$ is declared to be true.

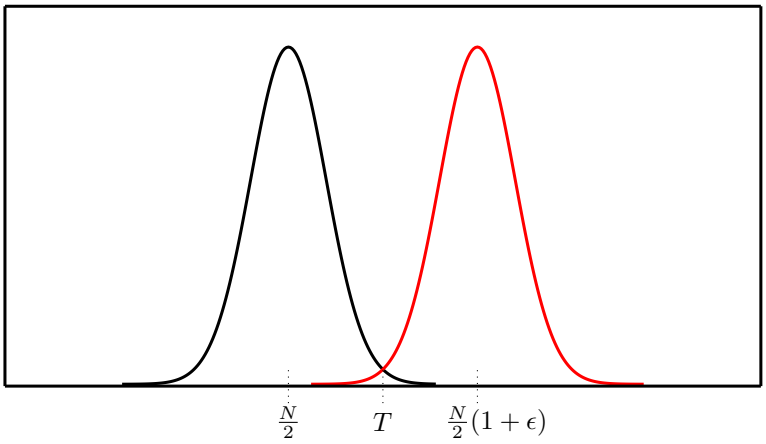


FIGURE 1. Two Normal distribution curves

Now, in the figure 1, we can see that some portion of the two graphs overlap with each other. In fact, for any point on the $x$-axis, their is some positive pdf value for each of the two distributions. This means, for any $t$, we can't say for sure that one of the hypothesis is wrong and the other is right. So, whatever be the value of the threshold $T$, there is a possibility of error in decision making. Two possible errors can occur:

1. **False Alarm Error:** This error occurs when the chosen sequence $A$ is incorrect, i.e., $A \in H_0$, but in the experiment we achieve $t \geq T$. As a result we accept a wrong sequence to be the correct one. The probability of this event is denoted by $P_{fa}$. Since the chosen sequence is actually incorrect, the probability distribution associated with this experiment is the one with mean $\mu_0$ and variance $\sigma_0$. So, to find the probability of this kind of error, we have to find the probability that $t \geq T$ in the normal distribution 1. This can be given by $P_{fa} = \frac{1}{\sqrt{2\pi}\sigma_0} \int_T^\infty e^{-\frac{(x-\mu_0)^2}{2\sigma_0^2}} dx$.
2. **Error of Non-Detection:** This error occurs when the chosen sequence $A$ is the correct one, i.e., $A \in H_1$, but it can't be detected because in the experiment, $t < T$. Denote the probability of this error by $P_{nd}$. This is basically the probability of $t < T$ in distribution 2, which can be given by $P_{nd} = \frac{1}{\sqrt{2\pi}\sigma_1} \int_{-\infty}^T e^{-\frac{(x-\mu_1)^2}{2\sigma_1^2}} dx$.

So, our primary aim is to minimize these two errors. The probability of these two errors depend on how we choose our threshold $T$. It is very clear from the expressions of $P_{fa}$ and $P_{nd}$ that if we want to reduce one of them by shifting the value of $T$, the other one will obviously increase. But, this change is not same. Also, the value of $N$ plays a vital role here. Let us focus on the bound of these two errors. Authors in [1] restricted the false alarm error to be less than some $2^{-\alpha}$ and the Non-Detection error to be bounded by $1.3 \times 10^{-3}$. So, we have the two equations:

1. $\frac{1}{\sqrt{2\pi}\sigma_0} \int_T^\infty e^{-\frac{(x-\mu_0)^2}{2\sigma_0^2}} dx \leq 2^{-\alpha}$

TABLE 1. New attack complexities for 8 rounds Salsa and 7 rounds ChaCha

| Cipher | $n$ | $\lvert\epsilon\rvert$ | $\alpha$ | $T$ | $N$ | Existing complexity | New complexity |
|---|---|---|---|---|---|---|---|
| Salsa | 43 | 0.000176 | 16.43 | $2^{29.66}$ | $2^{30.66}$ | $2^{243.93}$ | $2^{243.74}$ |
| ChaCha | 53 | 0.000100 | 24.83 | $2^{31.72}$ | $2^{32.72}$ | $2^{235.93}$ | $2^{235.78}$ |

TABLE 2. Attack complexities using chaining distinguisher

| Cipher | Existing complexity | New complexity |
|---|---|---|
| Salsa | $2^{243.67}$ | $2^{243.23}$ |
| ChaCha | $2^{235.22}$ | $2^{234.78}$ |

2. $\frac{1}{\sqrt{2\pi}\sigma_1}\int_{-\infty}^{T} e^{-\frac{(x-\mu_1)^2}{2\sigma_1^2}}dx \leq 1.3\times 10^{-3}$.

From these two equations we can find the value of $N$ and $T$ which gives the best possible result. In [1], instead of finding the actual integration value, the authors used the fact that for any $x$, $\frac{1}{\sqrt{2\pi}}\int_x^\infty e^{-\frac{y^2}{2}}dy < e^{-\frac{x^2}{2}}$. So, $e^{-\frac{x^2}{2}}$ can be used as an upper bound of $\frac{1}{\sqrt{2\pi}}\int_x^\infty e^{-\frac{y^2}{2}}dy$, and this expression was used instead of the actual integral to find $N$ and $T$. According to this, the total number of samples is $N \approx \left(\frac{\sqrt{\alpha\log 4}+3\sqrt{1-\epsilon^2}}{\epsilon}\right)^2$. These many samples are used to achieve the bounds of $P_{nd} = 1.3\times 10^{-3}$ and $P_{fa}$ by $2^{-\alpha}$. The final complexity can be given by $2^n\big(N + 2^m P_{fa}\big) = 2^n \cdot N + 2^{256-\alpha}$. In our work, we perform the actual integration using Sage [10]. In Table 1, we present new attack complexities for 8 rounds Salsa and 7 rounds ChaCha.

**Chaining Distinguisher approach and our observation:** In [11], Shi et al. provided a new technique to recover the key of Salsa and ChaCha with better complexity. Instead of finding the non-PNB keybits by exhaustive search, they proposed to do it step by step. Suppose the set of non-PNB's is $K$. We find $r$ subsets of $K$, namely $K_1, K_2, \ldots, K_r$ such that for all $i = 1, 2, \ldots, (r-1)$, $K_i \subset K_{i+1}$ and $K_r = K$. Now, while finding the non-PNB's, first we aim to find the values of keybits of $K_1$ only. This takes an exhaustive search over only $2^{|K_1|}$ options. After finding these values correctly, we go for the keybits of $K_2 \setminus K_1$. Putting the correct values in $K_1$, we try all possible combinations for the keybits of $K_2 \setminus K_1$. When we find the correct one, as a whole we achieve the values of all keybits of $K_2$. Like this, at $i$-th iteration, we perform the exhaustive search over all combinations of $K_i \setminus K_{i-1}$, find the correct one, and thus we achieve $K_i$. After $r$ such iterations, we can achieve $K_r = K$. This technique is very helpful because searching the non-PNB's step by step reduces the search by huge margin, which at the end reduces the final complexity. Using chaining approach, Dey et al. [6] reported attack complexities for 8 rounds Salsa and 7 rounds ChaCha as $2^{243.67}$ and $2^{235.22}$ respectively. Using our actual integration, the attack complexity for Salsa reduces to

$$2^{256-43}\times 2^{29.96} + 2^{256-39-3.76}\times 2^{27.34} + 2^{256-3.76-15.62} = 2^{243.23}.$$

Similarly the complexity for ChaCha becomes

$$2^{256-53}\times 2^{31.52} + 2^{256-52-4.16}\times 2^{32.27} + 2^{256-4.16-24.25} = 2^{234.78}.$$

**Remark 1.** An important fact in this approach is the non-detection error probability. Since in this technique the PNB set is found in $r$ steps, in each of them there is chance of non-detection error. So, if the probability of non-detection error is $P_{nd}$, this means, correct guess can be detected with probability $(1 - P_{nd})$. Now, in each of the $r$ iterations, the correct guess can be detected with probability $(1 - P_{nd})$. Assuming these events independent, the probability that in all the iterations guesses are correct is detected is $(1 - P_{nd})^r \approx (1 - rP_{nd})$. This means, in the whole procedure, the probability that the non-detection error occurs is $1 - (1 - rP_{nd}) = rP_{nd}$, which is $r$ times the original non-detection probability $P_{nd}$.

## 4. How to assign values in PNB's

In the process of searching the non-PNB's, the idea was to assign random values to PNB's. Here, instead of assigning any random values, we try to focus on the values which will improve the whole attack complexity. For each PNB positions we find out some fixed values, which we always assign at those positions during finding the non-PNB's. Suppose, there are $m$ probabilistic neutral bits. So, as a tuple it has $2^m$ possible values, among which only one is correct. We observe that there are few set of values, which give a better bias of the backward probability on average, even if the values are not correct. Suppose $X$ and $\overline{X}$ are respectively the initial matrix and the matrix obtained by putting some arbitrary values at the PNB's. Now, we compute both $Z - X$ and $Z - \overline{X}$. If the differences between $Z - X$ and $Z - \overline{X}$ are at less number of positions, we observe that after applying $R - r$ rounds of reverse algorithm of ChaCha on $Z - \overline{X}$, the backward probability $\epsilon_a$ becomes high. Due to this high bias, from Neyman-Pearson formula, we can achieve a lower value of $N$, which will help to reduce the complexity. On the other hand, if $Z - X$ and $Z - \overline{X}$ have differences at many positions, the bias $\epsilon_a$ becomes low, increasing the value of $N$.

So, if the values of the PNB's can be chosen in such a way that the differences between $Z - X$ and $Z - \overline{X}$ can be minimized, we can achieve a high $\epsilon_a$. Of course, this difference depends on the actual values of the PNB's. If some guessed values of PNB's give very low difference between $Z - X$ and $Z - \overline{X}$ for some key $K_1$, the same guessed value may give large difference for some other key $K_2$. But, considering all possible keys, there are some values for PNB's which give low difference on average. If those values are assigned to the PNB's instead of assigning arbitrary values, we get advantage in our attack in average case.

**Procedure:** In ChaCha and Salsa, we have 8 cells which contains keybits. We work on a single key cell at a time. For convenience, let us assume that we work on key cell $k$. To find the values of the PNB's located at $k$, we consider all possible values for those bits. Suppose, $k$ contains $m$ PNB's and we denote them as $p_1, p_2, \ldots, p_m$. So, the block $p_1 p_2, \ldots, p_m$ has $2^m$ possible values. Let the values be $v_0, v_1, \ldots, v_{2^m-1}$. When we compute $Z - X$, by $W$ we denote the 32 bit block of $Z$ from which $k$ is subtracted. Now, we choose random values for $W$ and $k$. For each $j$ from 0 to $2^m - 1$, we construct 32 bit block $k_j$ by replacing the original value of PNB block $p_1 p_2, \ldots, p_m$, by value $v_j$. Next we compute $W - k$ and $W - k_j$ for all $j$. Then, for each $j$, we count the number of differences between $W - k_j$ and $W - k$, i.e., we count the number of 1's appearing in $(W - k) \oplus (W - k_j)$. Let this value be $c_{j_1}$. After this, we again choose random values for $W$ and $k$ and repeat the above operations. Thus, we repeat the same procedure and count the number of differences between $W - k$ and $W - k_j$. Let it be $c_{j_2}$. We repeat this for large number of arbitrary

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 3 | 6 | 7 | 15 | 16 | 17 | 18 | 31 | 35 | 38 | 67 | 68 |
| 1 | 1 | 0 | 1 | 1 | 1 | 0 | x | 1 | 0 | 0 | 1 |
| 71 | 72 | 73 | 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 |
| 1 | 1 | 0 | x | x | x | x | x | 1 | 1 | 1 | 1 |
| 100 | 103 | 104 | 105 | 106 | 107 | 127 | 136 | 137 | 138 | 139 | 156 |
| 0 | 1 | 1 | 1 | 1 | 0 | x | 0 | 0 | 0 | 1 | x |
| 159 | 191 | 223 | 224 | 225 | 226 | 227 | 228 | 248 | 249 | 250 | 251 |
| x | x | x | 1 | 1 | 1 | 1 | 0 | x | x | x | x |
| 252 | 253 | 254 | 255 | | | | | | | | |
| x | x | x | x | | | | | | | | |

Table 3. Values for Probabilistic Neutral Bits of ChaCha

values of $W$ and $k$. Say this value is $\ell$. We add all $c_{j_1}, c_{j_2}, c_{j_3}, \ldots, c_{j_\ell}$'s to get the total number of differences, say $c_j$. Thus, for all $v_j$ we have a corresponding $c_j$. Now suppose , $c_{j_0} = \min_j\{c_j\}$ Then we assign $v_{j_0}$ for the PNB block $p_1p_2, \ldots, p_m$. We repeat the same operation for each keycell and obtain a value for the PNB block of that cell.

However, if there is a PNB block consisting of consecutive bits ending at the MSB of any cell, i.e, of the form $p_{31}p_{30}p_{29}, \ldots, p_{32-i}$, then we observe that for that block any arbitrary value can be assigned to the PNB's. This means, all $2^i$ possible values for $p_{31}p_{30}p_{29}, \ldots, p_{32-i}$ give same bias on average. Suppose $k = k_{31}k_{30}, \ldots, k_0$ is a keycell, of which the first $i$ most significant keybits, i.e, $k_{31}, k_{30}, k_{29}, \ldots, k_{32-i}$ are PNB's. Now, suppose $z_{31}z_{30}, \ldots, z_0$ be the corresponding $Z$. Suppose $k'_{31}k'_{30}k'_{29}, \ldots, k'_{32-i}$ is any arbitrary value that we assign to the PNB's. We call this new 32-bit value $k'$. Now, we compare the differences between $Z - k$ and $Z - k'$.

By $Z_1, k_1, k'_1$ we denote the most significant $i$ bits of $Z, k, k'$ respectively. Since the last $32-i$ bits are not PNB's, they are same for $k, k'$. As a result, the last $32-i$ bits of $Z-k, Z-k'$ are same. So the number of positions where $Z-k \bmod 2^{32}$ and $Z - k' \bmod 2^{32}$ differ is same as the number of positions where $(Z_1 - k_1) \bmod 2^i$ and $(Z_1 - k'_1) \bmod 2^i$ differ. Now, we consider all possible values for $k_1$. So, for all possible values of $k_1$, $Z_1 - k_1$ gives all possible $i$-bit values that can be generated by 0 and 1. Let us call them $k_1^1, k_1^2, \ldots, k_1^{2^i}$. For all $k_j$, we count the difference between $Z_1 - {k_1}^j$ and $Z_1 - {k'_1}^j$ and find their sum. Now, the number of differences between $Z_1 - k_1^j$ and $Z_1 - {k'_1}^j$ is the number of 1's appearing in $(Z_1 - k_1^j \oplus Z_1 - {k'_1}^j)$. Now, the set $\{Z - k_1 \oplus Z - k'_1 \mid k_1 \text{ is a i bit number}\}$ is basically the set of all possible i bit numbers. So, the sum is $\sum_{j=0}^{2^i} j\binom{i}{j}$, because there are $\binom{i}{j}$ $i$-bit numbers which contains exactly $j$ 1's. Now, this value is same for any value of $k'_1$, i.e., $\sum_{j=0}^{2^i} j\binom{i}{j}$ does not depend on the value of $k_1$. So, the total number of difference is same for any value of the block $p_{31}p_{30}, \ldots, p_{32-i}$.

**Experimental Results**

**ChaCha:** We run our experiment over ChaCha. We use the idea of Maitra [8] to minimize the number of differences after first round by choosing proper IV. To find the best value for the PNB block of each keycell, we experimented on $10^7$ keys. We provide the values for 52 PNB's in Table 3. The PNB blocks where any arbitrary values give same bias, is denoted by x. Like [4], we put the input difference at position $(13, 13)$, i.e., at 13th bit of 13th word. The output difference is observed

| Percentage of keys | bias (existing) | bias (our) | existing complexity | our complexity |
|---|---|---|---|---|
| 10 | 0.000200 | 0.000648 | $2^{234.97}$ | $2^{231.56}$ |
| 20 | 0.000178 | 0.000433 | $2^{235.28}$ | $2^{232.68}$ |
| 30 | 0.000165 | 0.000314 | $2^{235.50}$ | $2^{233.57}$ |
| 40 | 0.000152 | 0.000231 | $2^{235.73}$ | $2^{234.43}$ |
| 50 | 0.000139 | 0.000182 | $2^{235.96}$ | $2^{235.09}$ |

TABLE 4. Comparison of bias $\epsilon$ and complexities between existing and our method for ChaCha when $n = 52$.

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 25 | 26 | 27 | 28 | 29 | 30 | 31 | 39 | 70 | 71 | 72 | 107 |
| x | x | x | x | x | x | x | x | 1 | 1 | 0 | 1 |
| 119 | 120 | 121 | 122 | 164 | 165 | 166 | 167 | 168 | 169 | 170 | 171 |
| 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| 172 | 173 | 174 | 175 | 176 | 209 | 210 | 211 | 212 | 213 | 224 | 225 |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| 241 | 242 | 243 | 244 | 245 | 246 | 255 | | | | | |
| 1 | 1 | 1 | 1 | 1 | 0 | x | | | | | |

TABLE 5. Values for the probabilistic neutral bits of Salsa

after 4.5 rounds at $\Delta^{4.5}_{0,0} \oplus \Delta^{4.5}_{0,8} \oplus \Delta^{4.5}_{1,0} \oplus \Delta^{4.5}_{5,12} \oplus \Delta^{4.5}_{11,0} \oplus \Delta^{4.5}_{9,0} \oplus \Delta^{4.5}_{15,0} \oplus \Delta^{4.5}_{12,16} \oplus \Delta^{4.5}_{12,24}$. The average bias $\bar{\epsilon}$ observed for random assignment of values for PNB's is 0.000144, whereas our selected values for PNB's give bias 0.000318. We observe that for 67% keys, special PNB gives higher $\epsilon$ than random PNB. For 10% of the keys, our complexity is around 10 times faster than the existing complexity. The comparison between our complexity and existing complexity is provided in Table 4. In the figure 2, we represent this comparison graphically. The x-axis presents the percentile of keys and the y-axis presents the probabilities.

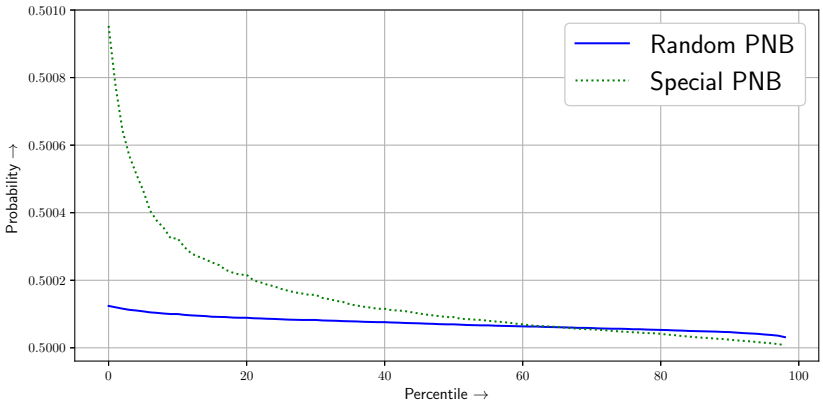


FIGURE 2. Comparison between the probability achieved by random values and our values for ChaCha

**Salsa:** In Table 5, we provide the values of PNB blocks which gives the best bias for Salsa. The PNB blocks where any arbitrary values give same bias, is denoted by x.

| Percentage of keys | bias (existing) | bias (our) | existing complexity | our complexity |
|---|---|---|---|---|
| 10 | -0.000232 | -0.000667 | $2^{243.18}$ | $2^{240.11}$ |
| 20 | -0.000207 | -0.000397 | $2^{243.48}$ | $2^{241.53}$ |
| 30 | -0.000192 | -0.000305 | $2^{243.69}$ | $2^{242.24}$ |
| 40 | -0.000181 | -0.000226 | $2^{243.86}$ | $2^{243.06}$ |
| 50 | -0.000176 | -0.000192 | $2^{243.93}$ | $2^{243.51}$ |

TABLE 6. Comparison of bias and complexities between existing and our method for Salsa when $n = 43$.

The input difference is put at position $(7, 0)$ and output difference is observed at the XOR of $(9, 0), (13, 0), (1, 13)$. The average bias $\bar{\epsilon}$ observed for random assignment of values for PNB's is -0.000170, whereas our selected values for PNB's give bias -0.000308. We observe that for 57% keys, special PNB gives higher $\epsilon$ than random PNB. In Table 4 we provide the comparison between our result and existing result upto 50% of keys. Kindly note that from the Table 4, it is clear that, for around 10% of the keys, the complexity is 8.40 times faster than existing result. However, as the percentage of keys increases, our result gets closer to existing best result, but still it is much better. In the figure 3, we represent this comparison graphically. The x-axis presents the percentile of keys and the y-axis presents the probabilities.

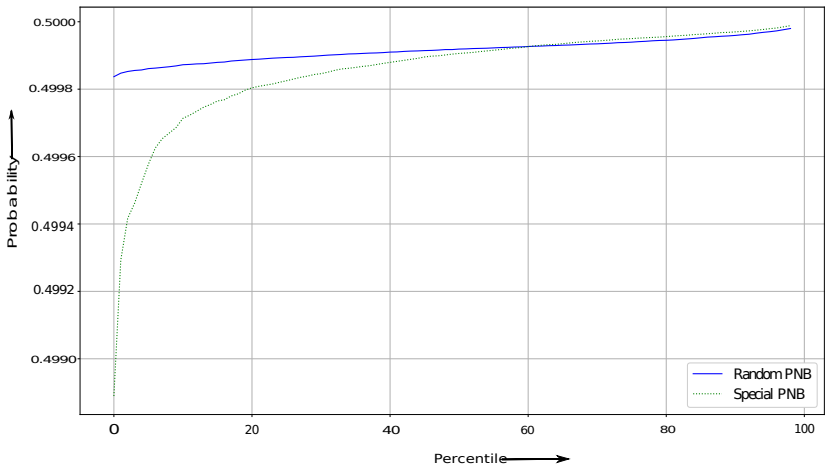


FIGURE 3. Comparison between the probability achieved by random values and our values for Salsa

## 5. MODIFICATION OF DESIGN AS A COUNTERMEASURE OF DIFFERENTIAL ATTACK

In this section, we analyse the structure of ChaCha and aim to find countermeasure to defend the proposed differential attacks without hampering the basic design pattern of the cipher. We propose a small modification of the output keystream function. Since the design of ChaCha is an excellent one, which has been proven by its security and use in last few years, we do not intend to bring about any major change in its design. We keep the basic quarterround function same as the actual design. However, we make a small tweak in the output function of the cipher. But

interestingly we can see that this change can defend all the key recovery attacks based on probabilistic neutral bits against ChaCha.

**Output keystream:** In the original version of ChaCha, the output keystream $Z$ is given by the sum of initial and final matrix: $Z = X + X^R$. In our modification, we change it and define the output function as: $Z = X^1 + X^R$. This small change does not affect the speed or memory of the cipher. In fact, $R = 7$ is secure enough to defend the differential attacks. Therefore, actually the tweak can help to reduce the number of rounds and make the cipher faster. In the next subsection we discuss the advantage of this tweak against differential attacks.

5.1. ANALYSIS AND ADVANTAGE IN MODIFIED VERSION: Here we discuss the advantage of the tweak in the design of the cipher. Our primary focus is to resist the differential attack based on Probabilistic neutral bits. Later we also show that the tweak is not going to hamper any other security, i.e., it does not make the cipher vulnerable against any other type of attack.

**Differential distinguisher:** In the modified version, no changes has been made in the basic quarterround function. Therefore the usual differential distinguishers of ChaCha will also work for the modified version. ChaCha can be distinguished upto 4th round using single bit distinguisher and 5th round using triple bits. Same differential is applicable for modified version as well. So, we do not resist the distinguishing attack using this modification. But, in the modified version, this distinguisher will not lead into any key recovery attack. The tweak will resist the generation of probabilistically neutral bits. As a result, the meet in the middle approach of attack will not be helpful against the cipher. A detailed idea of Probabilistic Neutral Bits has been explained in in 3. Using this idea, original ChaCha can be attacked upto 7 rounds. Here, we explain why this tweaked version of ChaCha is much secure than ChaCha against all these attacks.

**Unavailability of Probabilistically neutral bits:** Suppose the size of the PNB set is $m$, and non-PNB set is $256 - m$. Then, in the original version of ChaCha, if the guess of non-PNB's is correct, then between $X$ and $\tilde{X}$, 128 constant bits, 128 counter and nonce bits, $256 - m$ non-PNB's are exactly same. So, in total, $512 - m$ bits are exactly same between $X$ and $\tilde{X}$. Same is true between $X'$ and $\tilde{X}'$ also. So, in $Z - \tilde{X}$, a large number of bits are same as $X^R$ (since $X^R = Z - X$). Due to this large number of matches, $Z - \tilde{X}$ behaves almost same as $X^R$. Exactly the same is true between $X'^R$ and $Z' - \tilde{X}'$. As a result, $\tilde{Y}$ and $\tilde{Y}'$, which are obtained by applying reverse algorithm on $Z - \tilde{X}$ and $Z' - \tilde{X}'$, behaves similar to $Y$ and $Y'$ respectively. This is why $\tilde{\Gamma}_{p,q} = \tilde{Y}_{p,q} \oplus \tilde{Y}'_{p,q}$ gives a high bias for the correct guess.

Now, in our design, $Z$ is obtained by $Z = X^1 + X^R$. Unlike $X$, not a single bit of $X^1$ is known to the attacker. Even if the guess for non-PNB's is correct, that does not help to construct a matrix which has so many matches with $X^1$. As a result, even the correct guess of non-PNB's won't be able to produce a significant bias of $\tilde{\Gamma}_{p,q}$. However, one natural idea that may occur is: if the guess of non-PNB's is correct, then we can apply one round of ChaCha on $\tilde{X}$ and $\tilde{X}'$ and achieve some $\tilde{X}^1$ and $\tilde{X}'^1$, which can be considered as an approximation of actual $X^1$ and $X'^1$. Then, by applying reverse algorithm on $Z - \tilde{X}^1$ and $Z - \tilde{X}'^1$ we can proceed similarly as original ChaCha. But, experimentally we observe that this procedure is not helpful to produce an attack. We have run experiment on five million samples and observed that if a single keybit of $X$ is assigned an incorrect value, then applying one round of the algorithm, on average we have 12 bits of the matrix $\tilde{X}^1$ which are independent

TABLE 7. Number of *Random Value Positions* for some size of PNB

| Percentage of PNB's | Number of *Random Value Positions* |
|---|---|
| 5% | 70 |
| 10% | 127 |
| 15% | 180 |
| 20% | 219 |

from the respective bits of $X^1$. The minimum number of differences between $\tilde{X}^1$ and $X^1$ is 4. So, this means, if a single incorrect bit can bring so many differences after one round, then assigning arbitrary values to $m$ PNB's will bring about huge difference between $X^1$ and $\tilde{X}^1$, and $\tilde{X}^1$ will not work as an approximation of $X^1$ at all.

In our experiment we choose some keybit positions randomly, assume these to be our PNB set and assign arbitrary values to them. After that, applying one columnround on the matrix we check the number of difference between the original $X^1$ and this new $\tilde{X}^1$. For one such choice of keybit positions, we run experiment for 5 million different keys and finally find the positions of $\tilde{X}^1$ which behaves randomly with respect to $X^1$, i.e., the values at those positions have no positive or negative bias towards the values of the respective positions of $X^1$. Based on the size of the chosen PNB set, in Table 7, we provide the number of positions that behave randomly with respect to the original matrix. For each size given in the table, 100 different random sets of keybits are chosen. For each of these sets, we run one columnround on $X$ and $\tilde{X}$, achieve $X^1$ and $\tilde{X}^1$ and compare the value at each position of $X^1$ to the respective position of $\tilde{X}^1$. Repeating this for 5 million random keys, we find the probability $\Pr(X^1_{i,j} = \tilde{X^1_{i,j}})$. We call a position $(i, j)$ a *Random value position* if $\Pr(X^1_{i,j} = \tilde{X^1_{i,j}}) \approx \frac{1}{2}$. In this context, in our experiment we assign a threshold of range .0001 to declare a keybit position to be random. Now, we count the number of *Random value position*s and give it on the table. If the number of *Random value position*s is high, we can say that $X^1$ cannot be approximated by $\tilde{X}^1$. From the Table 7 it is clear that the number of *Random value positions* is very high, even if the number of PNB's are 5%. With the increment of the PNB set size, it increases by huge margin.

Therefore, even if a differential distinguisher can be found in this design, the distinguisher can't be used to achieve a key recovery attack, since no probabilistic neutral bit can be generated. Any key recovery attack of ChaCha which are based on the idea of PNB, is not effective in this modified design.

**Detailed Analysis of the Output Keystream:** Now, the question is, instead of computing the output function as $Z = X^1 + X^R$, is there any other $i < R$ such that defining the output function as $Z = X^i + X^R$ will provide even better security than $X^1 + X^R$. We do not claim that $X^1 + X^R$ is the best possible way we can define the output function. One thing is very clear. The advantage the attacker has in the usual output keystream of ChaCha ($Z = X + X^8$) is that half of $X$ is already known to him. This advantage he does not have if $X^1$ is instead of $X$. But, the knowledge of $X$ may help to predict some keybits of $X^1$. So, if we use some other $i > 1$, it is more difficult for the attacker to predict the keybits of $X^i$ because of the diffusion. So, this observation apparently may lead to the conclusion that increasing the value of $i$ will improve the security.

But, there is another factor which comes into play as we increase $i$. As $i$ gets closer to $R$, the correlation between $X^i$ and $X^R$ becomes more prominent. This correlation may be reflected in the computation of output $X^i + X^R$. As a result, strong correlation between $X^i$ and $X^R$ may weaken the design. For example, if $i = R - 1$, i.e., $Z$ is computed as $Z = X^{R-1} + X^R$, the cipher becomes very much weaker. Below we show that how easily the cipher can be attacked if $i = R - 1$.

**Case when $Z = X^R + X^{R-1}$:** Without loss of generality, let us assume that the $R$-th round is a columnround. So, $X^R$ is computed by applying columnround on the four columns of $X^{R-1}$. Now, let's assume the four column vectors of $X^{R-1}$ to be $C_1, C_2, C_3$ and $C_4$. Each of their size is 128. After applying columnround, suppose the achieved columns of $X^R$ are $C'_1, C'_2, C'_3$ and $C'_4$. So, each $C'_i$ is achieved by applying columnround on the respective $C_i$, and does not depend on the other columns of $X^{R-1}$. Now, in $Z = X^R + X^{R-1}$, each $C_i$ is added with $C'_i$. So, for each of $2^{128}$ possible values of $C_i$, we compute $C'_i$ and then compute $C_i + C'_i$. For the correct guess of $C_i$, we can achieve 128 bits of $Z$. So this complexity is $2^{128}$ only. Repeating the same procedure for all four columns, we can achieve all four $C_i$, with $4 \times 2^{128}$ attempts.

So, $i = R - 1$ can not ne used for output generation. Now, which value of $i$ can provide the best security of the cipher is clearly a topic of further analysis and study. But certainly, our design, where $i = 1$, is a very secure choice to defend all the differential attacks available against Salsa and ChaCha.

5.2. RESISTANCE TO OTHER COMMON ATTACKS: Though our tweak in the design of ChaCha is a minor one, it is very much effective in defending the differential key recovery attacks based on Probabilistically Neutral bits. However, we also have to make sure that this change of design doesnot weakens the cipher against other kind of attacks. Since the basic design principle is kept same as the original ChaCha, it is obvious that it will provide same security as ChaCha against all other attacks. Still we go through an analysis of this modified design against common attacks.

**Time Memory Data Tradeoff Attack:** This is a well known attack against stream ciphers where the attacker uses memory to some information before actual attack reduce the time complexity of the attack. However, this kind of attacks have been performed so far mostly against the stream ciphers with Grain-like structure. The basic design principle of Salsa or ChaCha is completely different from that. To the best of our knowledge, there is no TMDTO attack known against Salsa, ChaCha or any stream cipher with similar structure. So, the usual attack strategy of TMDTO attacks against Grain-like ciphers is not applicable against the design pattern of Salsa family ciphers. However, a conventional rule is followed to design Grain-like ciphers to defend TMDTO attack, which says that the state size should be at least twice the key size. That rule is already followed in SalSa, ChaCha and in our modified design, because the state size (512) here is twice the key size (256). Therefore, our design is without any doubt secure against the common TMDTO attack.

**Linear approximation attack:** This is another interesting attack which has been effective against some stream ciphers in recent times. However, in all the attacks performed by this method, a non-linear polynomial function is approximated by a linear function. After that, state is found by solving the linear equations. In binary polynomial functions, the product terms have a high bias towards zero. For example, a product term of $n$ variables $x_1 x_2 \dots x_n$ gives zero with probability $(1 - 1/2^n)$. This

is why, unless constructed very carefully, polynomial functions can be approximated probabilistically by linear functions. But in the quarterround function of ChaCha, four additions modulo $2^{32}$ are involved. This addition is a very strong non-linear function, where expressing each of the bits in polynomial form involves all the bits on the right side of it (less significant bits). So, approximating the addition of 32 bit numbers is very difficult to be approximated by a linear function throughout an attack. Suppose $a$ and $b$ are added to get $S$. Let us denote by $X[i]$ we denote the $i$-th entry of the number $X$, Then the $i$-th entry of $S$, which we denote by $S[i]$, can be given by $S[i] = a[i] \oplus b[i] \oplus C[i]$, where $C[i]$ is the carry received from the previous bits. This carry is the reason of the non-linearity of the function, and it depend on all the bits of $a$ and $b$ which are less significant than $i$-th (i.e., $a[0], a[1], \ldots, a[i-1]$ and $b[0], b[1], \ldots, b[i-1]$). The term $C[i]$ can be represented as a polynomial non-linear function, but it is a very complicated one. Therefore, a good linear approximation is not possible for the addition.

**Cube Attack:** Cube attack has been very useful against ciphers like Grain 128. The reason behind the success of the this attack the low degree of the function. However, as already mentioned, the addition function of the quarterround function is a complicated one. Also, quarterround function involves a rotation, which complicates the function further. Therefore, the polynomial representation of a quarterround function is of a very high degree. And as the number of rounds increases, the degree increases very fast. Therefore, cube attack is not applicable against this kind of cipher.

**Algebraic Attack:** There has not been any algebraic attack either against ciphers of this kind, the reason being the quarterround function. The nonilinearity and high degree of this function restricts any algebraic attack against the cipher. So, in our modification, since we do not make any in the quarterround function, there is no chance of algebraic attack.

## 6. Conclusion

In this paper, we first perform more accurate complexity computation and provide better result. Then, we aim at increasing the backward probability bias of differential attack against reduced round Salsa and ChaCha. Instead of assigning random values for probabilistic neutral bits, we found some fixed values for the PNB blocks of the keycells. These values give minimum difference between $Z - X$ and $Z - X'$ in average case. As a result, the backward probability bias increases significantly. This helps to reduce the complexity of the attack. Finally, we propose a slight change in the design of the original ChaCha. We can defend the differential attack ideas that have been proposed against ChaCha. Also, instead of running the algorithm by at least 8 rounds, we run our algorithm by 7 rounds only. Due to our modification, the usual differential attacks don't work even in 7 rounds. As a result, in one side we are making the keystream production faster. On the other side, this design is providing better security.

## References

[1] J. P. Aumasson, S. Fischer, S. Khazaei, W. Meier and C. Rechberger, New features of latin dances: Analysis of salsa, chacha, and rumba, *FSE 2008, LNCS,* **5086** (2008), 470–488.

[2] D. J. Bernstein, Salsa20 specification, eSTREAM Project algorithm description, http://www.ecrypt.eu.org/stream/salsa20pf.html, 2005.

[3] D. J. Bernstein, ChaCha, a variant of Salsa20, In Workshop Record of SASC, volume 8, 2008.

[4] A. R. Choudhuri and S. Maitra, *Significantly Improved Multi-bit Differentials for Reduced Round Salsa and ChaCha,* Accepted in FSE 2017, Available at http://eprint.iacr.org/2016/1034.
[5] P. Crowley, Truncated differential cryptanalysis of five rounds of Salsa20, IACR 2005. Available at http://eprint.iacr.org/2005/375.
[6] S. Dey and S. Sarkar, Improved analysis for reduced round Salsa and ChaCha, *Discrete Applied Mathematics*, **227** (2017), 58–69.
[7] S. Fischer, W. Meier, C. Berbain and J. F. Biasse, Non-randomness in eSTREAM Candidates Salsa20 and TSC-4, *Progress in Cryptology–INDOCRYPT 2006, LNCS*, **4329** (2006), 2–16.
[8] S. Maitra, Chosen IV cryptanalysis on reduced round chacha and salsa, *Discrete Applied Mathematics*, **208** (2016), 88–97.
[9] S. Maitra, G. Paul and W. Meier, *Salsa20 Cryptanalysis: New Moves and Revisiting Old Styles,* WCC 2015. Available at http://eprint.iacr.org/2015/217.
[10] Sage: Open Source Mathematics Software, http://www.sagemath.org.
[11] Z. Shi, B. Zhang, D. Feng and W. Wu, Improved key recovery attacks on reduced-round salsa20 and ChaCha, *ICISC 2012, LNCS*, **7839** (2012), 337–351.
[12] Y. Tsunoo, T. Saito, H. Kubo, T. Suzaki and H. Nakashima, *Differential Cryptanalysis of Salsa20/8,* 2007.

Received for publication October 2018.

*E-mail address:* sabya.ndp@gmail.com
*E-mail address:* tapabrata.roy.048@gmail.com
*E-mail address:* sarkar.santanu.bir@gmail.com