Latin Dances Reloaded: Improved Cryptanalysis against Salsa and ChaCha, and the proposal of Forró

Murilo Coutinho[1], Iago Passos[1], Juan C. Grados Vásquez[2], Fábio L. L. de Mendonça[1], Rafael T. de Sousa Jr.[1], and Fábio Borges[3]

[1] Electrical Engineering Department (ENE), Technology College, University of Brasília, Brasília, Brazil (e-mail: murilo.coutinho@redes.unb.br)
[2] Technology Innovation Institute, Abu Dhabi, UAE
[3] National Laboratory for Scientific Computing, Petrópolis, Brazil

Keywords: Differential-Linear Cryptanalysis · ARX · ChaCha · Salsa · Forró

Abstract. In this paper, we present 4 major contributions to ARX ciphers and in particular to the Salsa/ChaCha family of stream ciphers:

(a) We propose an improved differential-linear distinguisher against ChaCha. To do so, we propose a new way to approach the derivation of linear approximations by viewing the algorithm in terms of simpler subrounds. Using this idea we show that it is possible to derive almost all linear approximations from previous works from just 3 simple rules. Furthermore, we show that with one extra rule it is possible to improve the linear approximations proposed by Coutinho and Souza at Eurocrypt 2021 [11].

(b) We propose a technique called Bidirectional Linear Expansions (BLE) to improve attacks against Salsa. While previous works only considered linear expansions moving forward into the rounds, BLE explores the expansion of a single bit in both forward and backward directions. Applying BLE, we propose the first differential-linear distinguishers ranging 7 and 8 rounds of Salsa and we improve PNB key-recovery attacks against 8 rounds of Salsa.

(c) Using all the knowledge acquired studying the cryptanalysis of these ciphers, we propose some modifications in order to provide better diffusion per round and higher resistance to cryptanalysis, leading to a new stream cipher named Forró. We show that Forró has higher security margin, this allows us to reduce the total number of rounds while maintaining the security level, thus creating a faster cipher in many platforms, specially in constrained devices.

(d) Finally, we developed *CryptDances*, a new tool for the cryptanalysis of Salsa, ChaCha, and Forró designed to be used in high performance environments with several GPUs. With *CryptDances* it is possible to compute differential correlations, to derive new linear approximations for ChaCha automatically, to automate the computation of the complexity of PNB attacks, among other features. We make *CryptDances* available for the community at `https://github.com/MurCoutinho/cryptDances`.

1 Introduction

Cryptography is an indispensable tool used to protect information in computing systems. It is used to protect data at rest and data in motion by billions of people everyday. For example, cryptography is used in financial transactions, mobile messaging applications, blockchain technology, authentication systems, and many other systems and solutions. Among the most important cryptographic primitives, stream ciphers are symmetric algorithms used to encrypt large amounts of data with high performance both in software and in hardware.

In particular, ARX-based design is a major building block of modern ciphers due to its efficiency in software. ARX stands for addition, word-wise rotation and XOR. Indeed, ciphers following this framework are composed of those operations and avoid the computation of smaller S-boxes through look-up tables. ARX-based designs are not only efficient but also provide good security properties. The algebraic degree of ARX ciphers is generally high after only a very few rounds as the carry bit within one modular addition already reaches almost maximal degree. For differential and linear attacks, ARX-based designs show weaknesses for a small number of rounds. However, after some rounds the differential and linear probabilities decrease rapidly. Thus, the probabilities of differentials and the absolute correlations of linear approximations decrease very quickly as we increase the number of rounds.

Salsa [6] is an ARX-based stream cipher designed by Bernstein in 2005 as a candidate for the *eSTREAM* competition [27]. The original proposal was for 20 rounds. The 12-round variant of Salsa - Salsa20/12 - was accepted into the final *eSTREAM* software portfolio. Salsa is especially important and is used in practice in several applications, such as DNS implementations, in the Linux Kernel, Password managers (e.g., KeePassX and MacPass), messaging software (e.g., Viber and Discord), and many other (see [19] for a huge list of applications, protocols and libraries using Salsa).

Later, in 2008, Bernstein proposed some modifications to Salsa in order to provide better diffusion per round and higher resistance to cryptanalysis. These changes created a new stream cipher, a variant named ChaCha [5]. Although Salsa was one of the winners of the eSTREAM competition, ChaCha has received much more attention through the years. Nowadays, we see the usage of this cipher in several projects and applications.

ChaCha, along with Poly1305 [4], is one of the cipher suites of the new TLS 1.3 [21], which has been used by Google on both Chrome and Android. Not only has ChaCha been used in TLS but also in many other protocols such as SSH, Noise and S/MIME 4.0. In addition, the RFC 7634 proposes the use of ChaCha in IKE and IPsec. ChaCha has been used not only for encryption, but also as a pseudo-random number generator in any operating system running Linux kernel 4.8 or newer. Additionally, ChaCha has been used in several applications such as WireGuard (VPN) (see [18] for a huge list of applications, protocols and libraries using ChaCha).

Related Work. Due to the popularity of both Salsa and ChaCha, it is important to evaluate their security. Indeed, the cryptanalysis of Salsa is well

understood and several authors studied its security [17,8,25]. The cryptanalysis of Salsa was introduced by Crowley [13] in 2005. Later, Aumasson et al. at FSE 2008 [2] presented one of the most important works on the cryptanalysis of these ciphers with the introduction of the notion of Probabilistic Neutral Bits (PNBs), showing attacks against Salsa20/7, Salsa20/8, ChaCha20/6 and ChaCha20/7.

After that, several authors proposed small enhancements on the attack of Aumasson et al. For example, the work by Shi et al. [28] introduced the concept of Column Chaining Distinguisher (CCD) to achieve some incremental advancements over [2] Salsa and ChaCha. Maitra, Paul, and Meier [24] studied an interesting observation regarding round reversal of Salsa, but no significant cryptanalytic improvement could be obtained using this method. Maitra [23] used a technique of Chosen IVs to obtain certain improvements over existing results. Dey and Sarkar [15] showed how to choose values for the PNB to further improve the attack.

Then, in a paper presented at FSE 2017, Choudhuri and Maitra [9] significantly improved the attacks by considering the mathematical structure of Salsa and ChaCha to find differential characteristics with much higher correlations. Other types of attacks were also studied, such as, related-cipher attacks [16] and chosen-IV attacks [23].

Recently, several works presented improvements in attack against ChaCha. First, Coutinho and Souza [10] proposed new multi-bit differentials using the mathematical framework of Choudhuri and Maitra. In Crypto 2020, Beierle et al. [3] proposed improvements to the framework of differential-linear cryptanalysis against ARX-based designs and further improved the attacks against ChaCha. At Eurocrypt 2021, Coutinho and Souza [11] developed a new technique to expand linear trails improving the attack against ChaCha even further. However, these new techniques were not used against Salsa. At Eurocrypt 2022 Dey et al. [14] improved the analysis of the PNB construction and key recovery attacks against ChaCha. Finally, in Crypto 2022, rotational-cryptanalysis of ChaCha was improved [26].

Our Contribution. In this work, we present new attacks against ChaCha and Salsa. In the case of ChaCha, we propose a simpler way to derive linear approximations for the cipher. To do so, we view the algorithm in terms of subrounds. With this approach, we are able to derive the results from previous works from just 3 simple rules. As a reference, the methods of Coutinho and Souza [11] at Eurocrypt 2021 encompasses at least 18 different rules to derive linear approximations for ChaCha. Moreover, with our techniques we are able to improve the complexity of the best differential-linear distinguisher against ChaCha, reducing the complexity from 2^{224} to 2^{214}.

To attack Salsa, we introduce a novel technique called Bidirectional Linear Expansions (BLE). While previous works only considered linear expansions moving forward into the rounds, BLE explores the expansion of a single bit in both forward and backward directions. As we show, BLE is specially useful in situations that we do not have enough computational power to compute a differential correlation for the target single bit, but we can do so for each bit derived in

backward direction individually, and then combining them using the Piling-up Lemma. Using BLE we were able to improve attacks against Salsa. In particular, we improved key recovery attacks, significantly reducing the complexity from $2^{244.9}$ to 2^{218} for 8 rounds of Salsa. Also, we provide the first differential-linear distinguishers ranging 7 and 8 rounds of Salsa in the literature. Still using BLE, we were able to find several new differential for 3.5 rounds of ChaCha. Unfortunately, we were not able to improve key recovery attacks in this case.

Next, we propose a new modification of Salsa and ChaCha, the stream cipher Forró. We show that Forró has a higher security margin. For comparison, the best distinguishers against 5 rounds of Salsa, ChaCha, and Forró, have complexities of 2^8, 2^{16}, and 2^{130}, respectively. To achieve that we introduce a new design strategy, called Pollination, constructed to speed up confusion and diffusion. Then, we show that Forró can deliver the same security in less time in several platforms, specially in constrained devices. Finally, we present a new tool, called *CryptDances* (`https://github.com/MurCoutinho/cryptDances`) designed to allow researchers to explore the cryptanalysis of ChaCha, Salsa, and Forró in a high performance environment configured using MPI to distribute the work to several GPUs. We provide a summary of our cryptanalytic results in Table 1.

Organization of the paper. This paper is organized as follows: in Section 2, we review previous works and techniques. In Section 3, we propose a new approach to the derivation of linear approximations for ChaCha and present a new and improved differential-linear distinguisher. Then, in Section 4, we propose a new technique called Bidirectional Linear Expansions (BLE) and use it to improve attacks against Salsa. Next, in Section 5, we present the new stream cipher Forró and in Section 6 we give a brief description of the tool CryptDances. Finally, in Section 7 we present the conclusions and future works.

2 Specifications and Preliminaries

This section is divided in 5 parts as follows: first in Sections 2.1 and 2.2 we describe the algorithms Salsa and ChaCha, respectively. Then, in Section 2.3 we review the state-of-the-art differential-linear cryptanalysis, and in Section 2.4 we review the key recovery attacks using PNBs as used to attack Salsa and ChaCha. Finally, in Section 2.5 we review state-of-the-art techniques to create linear approximations for ARX ciphers and in particular to Salsa and ChaCha. To improve readability, we provide a summary of the main notation used throughout the paper in Table 2.

2.1 Salsa

Salsa operates on a state of 64 bytes, organized as a 4×4 matrix with 32-bit integers, initialized with a 256-bit key $k_0, k_1, ..., k_7$, a 64-bit nonce v_0, v_1 and a 64-bit counter t_0, t_1 (we may also refer to the nonce and counter words as IV words), and 4 constants c_0 = 0x61707865, c_1 = 0x3320646e, c_2 = 0x79622d32

Rounds	Algorithm	Type	Time	Data	Reference
3	Forró	Distinguisher	2^{19}	2^{19}	This work
4	ChaCha	Distinguisher	2^{6}	2^{6}	[9]
	Forró	Distinguisher	2^{37}	2^{37}	This work
5	Salsa	Distinguisher	2^{8}	2^{8}	[9]
	ChaCha	Distinguisher	2^{16}	2^{16}	[9]
	Forró	Key Recovery	2^{158}	2^{57}	This work
	Forró	Distinguisher	2^{130}	2^{130}	This work
6	Salsa	Distinguisher	2^{32}	2^{32}	[9]
	ChaCha	Key Recovery	2^{139}	2^{30}	[2]
	ChaCha	Key Recovery	$2^{127.5}$	$2^{37.5}$	[9]
	ChaCha	Key Recovery	$2^{77.4}$	2^{58}	[3]
	ChaCha	Distinguisher	2^{116}	2^{116}	[9]
	ChaCha	Distinguisher	2^{51}	2^{51}	[11]
7	Salsa	Key Recovery	2^{137}	2^{61}	[9]
	Salsa	Distinguisher	2^{109}	2^{109}	This work
	ChaCha	Key Recovery	2^{248}	2^{27}	[2]
	ChaCha	Key Recovery	$2^{237.7}$	2^{96}	[9]
	ChaCha	Key Recovery	$2^{230.86}$	$2^{48.8}$	[3]
	ChaCha	Key Recovery	$2^{221.95}$	$2^{48.83}$	[14]
	ChaCha	Distinguisher	2^{224}	2^{224}	[11]
	ChaCha	Distinguisher	2^{214}	2^{214}	This work
8	Salsa	Key Recovery	$2^{244.9}$	2^{96}	[9]
	Salsa	Key Recovery	2^{218}	2^{114}	This work
	Salsa	Distinguisher	2^{216}	2^{216}	This work

Table 1: Time and data complexity for the best attacks against ChaCha, Salsa, and Forró.

and $c_3 = 0\text{x}6b206574$. For Salsa, we have the following initial state matrix:

$$X^{(0)} = \begin{pmatrix} x_0^{(0)} & x_1^{(0)} & x_2^{(0)} & x_3^{(0)} \\ x_4^{(0)} & x_5^{(0)} & x_6^{(0)} & x_7^{(0)} \\ x_8^{(0)} & x_9^{(0)} & x_{10}^{(0)} & x_{11}^{(0)} \\ x_{12}^{(0)} & x_{13}^{(0)} & x_{14}^{(0)} & x_{15}^{(0)} \end{pmatrix} = \begin{pmatrix} c_0 & k_0 & k_1 & k_2 \\ k_3 & c_1 & v_0 & v_1 \\ t_0 & t_1 & c_2 & k_4 \\ k_5 & k_6 & k_7 & c_3 \end{pmatrix}. \tag{1}$$

The state matrix is modified in each round by a *Quarter Round Function* (QRF), named $QR_{Salsa}(a, b, c, d)$, which receives and updates 4 integers in the following way:

$$\begin{aligned} x_b^{(m)} &= x_b^{(m-1)} \oplus ((x_d^{(m-1)} + x_a^{(m-1)}) \lll 7) \\ x_c^{(m)} &= x_c^{(m-1)} \oplus ((x_a^{(m-1)} + x_b^{(m)}) \lll 9) \\ x_d^{(m)} &= x_d^{(m-1)} \oplus ((x_c^{(m)} + x_b^{(m)}) \lll 13) \\ x_a^{(m)} &= x_a^{(m-1)} \oplus ((x_d^{(m)} + x_c^{(m)}) \lll 18) \end{aligned} \tag{2}$$

One round of Salsa is defined as 4 applications of the QRF. There is a difference, however, between odd and even rounds. Thus, for odd rounds, when

Notation	Description
X	a 4×4 state matrix
$X^{(m)}$	state matrix after application of m rounds
$X^{[s]}$	state matrix after application of s subrounds
Z	output of Salsa, ChaCha or Forró, i.e., $Z = X + X^{(R)}$
$x_i^{(m)}$	i^{th} word of the state matrix $X^{(m)}$
$x_{i,j}^{(m)}$	j^{th} bit of i^{th} word of the state matrix $X^{(m)}$
$x_i^{(m)}[j_0, j_1, ..., j_t]$	the sum $x_{i,j_0}^{(m)} \oplus x_{i,j_1}^{(m)} \oplus \cdots \oplus x_{i,j_t}^{(m)}$
$x + y$	addition of x and y modulo 2^{32}
$\Theta(x, y)$	carry function of the sum $x + y$
$x \oplus y$	bitwise XOR of x and y
$x \lll n$	rotation of x by n bits to the left
Δx	XOR difference of x and x'. $\Delta x = x \oplus x'$
$\mathcal{ID}$	input difference
$\mathcal{OD}$	output difference

Table 2: Notation

$m \in \{1, 3, 5, 7, ...\}$, $X^{(m)}$ is defined from $X^{(m-1)}$, from $QR_{Salsa}(a, b, c, d)$ with $(a, b, c, d) = \{(0, 4, 8, 12), (5, 9, 13, 1), (10, 14, 2, 6), (15, 3, 7, 11)\}$, and for even rounds $m \in \{2, 4, 6, ...\}$ from $QR_{Salsa}(a, b, c, d)$ with $(a, b, c, d) = \{(0, 1, 2, 3), (5, 6, 7, 4), (10, 11, 8, 9), (15, 12, 13, 14)\}$.

The output of Salsa$20/R$ is then defined as the sum of the initial state with the state obtained after R rounds of operations $Z = X^{(0)} + X^{(R)}$. One should note that it is possible to parallelize each application of the QRF on each round and that each round is reversible, hence we can compute $X^{(m-1)}$ from $X^{(m)}$. For more information on Salsa, we refer to [6].

2.2 ChaCha

The stream cipher ChaCha was also proposed by Bernstein [5] as an improvement of Salsa. ChaCha consists of a series of ARX (addition, rotation, and XOR) operations on 32-bit words, being highly efficient in software and hardware. Each round of ChaCha has a total of 16 bitwise XOR, 16 addition modulo 2^{32} and 16 constant-distance rotations.

ChaCha operates on a state of 64 bytes, organized as a 4×4 matrix with 32-bit integers, initialized with a 256-bit key $k_0, k_1, ..., k_7$, a 64-bit nonce v_0, v_1 and a 64-bit counter t_0, t_1 (we may also refer to the nonce and counter words as IV words), and 4 constants $c_0 = 0x61707865$, $c_1 = 0x3320646e$, $c_2 = 0x79622d32$ and $c_3 = 0x6b206574$. For ChaCha, we have the following initial state matrix:

$$X^{(0)} = \begin{pmatrix} x_0^{(0)} & x_1^{(0)} & x_2^{(0)} & x_3^{(0)} \\ x_4^{(0)} & x_5^{(0)} & x_6^{(0)} & x_7^{(0)} \\ x_8^{(0)} & x_9^{(0)} & x_{10}^{(0)} & x_{11}^{(0)} \\ x_{12}^{(0)} & x_{13}^{(0)} & x_{14}^{(0)} & x_{15}^{(0)} \end{pmatrix} = \begin{pmatrix} c_0 & c_1 & c_2 & c_3 \\ k_0 & k_1 & k_2 & k_3 \\ k_4 & k_5 & k_6 & k_7 \\ t_0 & t_1 & v_0 & v_1 \end{pmatrix}. \quad (3)$$

The state matrix is modified in each round by a *Quarter Round Function* (QRF), denoted by $QR_{ChaCha}\left(x_a^{(r-1)}, x_b^{(r-1)}, x_c^{(r-1)}, x_d^{(r-1)}\right)$, which receives and updates 4 integers in the following way:

$$\begin{array}{ll} x_{a\prime}^{(r-1)} = x_a^{(r-1)} + x_b^{(r-1)}; & x_{d\prime}^{(r-1)} = (x_d^{(r-1)} \oplus x_{a\prime}^{(r-1)}) \lll 16; \\ x_{c\prime}^{(r-1)} = x_c^{(r-1)} + x_{d\prime}^{(r-1)}; & x_{b\prime}^{(r-1)} = (x_b^{(r-1)} \oplus x_{c\prime}^{(r-1)}) \lll 12; \\ x_a^{(r)} = x_{a\prime}^{(r-1)} + x_{b\prime}^{(r-1)}; & x_d^{(r)} = (x_{d\prime}^{(r-1)} \oplus x_a^{(r)}) \lll 8; \\ x_c^{(r)} = x_{c\prime}^{(r-1)} + x_d^{(r)}; & x_b^{(r)} = (x_{b\prime}^{(r-1)} \oplus x_c^{(r)}) \lll 7; \end{array} \tag{4}$$

One round of ChaCha is defined as 4 applications of the QRF. There is, however, a difference between odd and even rounds. For odd rounds, i.e. $r \in \{1, 3, 5, 7, ...\}$, $X^{(r)}$ is obtained from $X^{(r-1)}$ by applying $QR_{ChaCha}(a, b, c, d)$ with $(a, b, c, d) = \{(0, 4, 8, 12), (1, 5, 9, 13), (2, 6, 10, 14), (3, 7, 11, 15)\}$, and for even rounds $m \in \{2, 4, 6, ...\}$ from $QR_{ChaCha}(a, b, c, d)$ with $(a, b, c, d) = \{(0, 5, 10, 15), (1, 6, 11, 12), (2, 7, 8, 13), (3, 4, 9, 14)\}$.

The output of ChaCha20/R is then defined as the sum of the initial state with the state after R rounds $Z = X^{(0)} + X^{(R)}$. One should note that it is possible to parallelize each application of the QRF on each round and also that each round is reversible. Hence, we can compute $X^{(r-1)}$ from $X^{(r)}$.

Next, we introduce the concept of subrounds for ChaCha which will be very useful in the rest of this paper. First, we define the *Subround Function* (SRF), denoted by

$$(x_a^{[s]}, x_b^{[s]}, x_c^{[s]}, x_d^{[s]}) = SR_{ChaCha}\left(x_a^{[s-1]}, x_b^{[s-1]}, x_c^{[s-1]}, x_d^{[s-1]}, r_1, r_2\right),$$

which receives and updates 4 integers giving two rotation distances in the following way:

$$\begin{array}{ll} x_a^{[s]} = x_a^{[s-1]} + x_b^{[s-1]}; & x_d^{[s]} = (x_d^{[s-1]} \oplus x_a^{[s]}) \lll r_1; \\ x_c^{[s]} = x_c^{[s-1]} + x_d^{[s]}; & x_b^{[s]} = (x_b^{[s-1]} \oplus x_c^{[s]}) \lll r_2; \end{array} \tag{5}$$

Note that we can define the QRF in terms of the SRF. More precisely, we have that

$$QR_{ChaCha}(a, b, c, d) = SR_{ChaCha}(SR_{ChaCha}(a, b, c, d, 16, 12), 8, 7). \tag{6}$$

Therefore, it is easy to see that we can redefine ChaCha in terms of the SRF. Note that, giving our notation, for each round of ChaCha we have 2 subrounds being executed. In other words, if $X^{[2s]}$ denotes the state matrix after $2s$ subrounds, then we have that $X^{(s)} = X^{[2s]}$.

2.3 A review of Differential-Linear Cryptanalysis

In this section, we describe the technique of Differential-Linear cryptanalysis as used to attack ChaCha. Let E be a cipher and suppose we can write $E = E_2 \circ E_1$, where E_1 and E_2 are sub ciphers, covering m and l rounds of the main cipher,

respectively. We can apply an input difference $\mathcal{ID}$ $\Delta X^{(0)}$ in the sub cipher E_1 obtaining an output difference $\mathcal{OD}$ $\Delta X^{(m)}$ (see the left side of Fig. 1). The next step is to apply Linear Cryptanalysis to the second sub cipher E_2. Using masks Γ_m and Γ_{out}, we attempt to find good linear approximations covering the remaining l rounds of the cipher E. Applying this technique we can construct a differential-linear distinguisher covering all $m + l$ rounds of the cipher E. This is the main idea in Langford and Hellman's classical approach [20].

Alternatively the cipher E can be represented as the product of three ciphers, as follows: $E = E_3 \circ E_2 \circ E_1$. In this scenario, we can explore properties of the cipher in the first part E_1, and then apply a differential linear attack where we divide the differential part of the attack in two (see the right side of Fig. 1). Here, the $\mathcal{OD}$ from the sub cipher E_1 after r rounds, namely $\Delta X^{(r)}$, is the $\mathcal{ID}$ for the sub cipher E_2 which produces an output difference $\Delta X^{(m)}$. For more information in this regard, see [3].

It is important to understand how to compute the complexity of a differential-linear attack. We denote the differential of the state matrix as $\Delta X^{(r)} = X^{(r)} \oplus X'^{(r)}$ and the differential of individual words as $\Delta x_i^{(r)} = x_i^{(r)} \oplus x_i'^{(r)}$. Let $x_{i,j}^{(r)}$ denote the j-th bit of the i-th word of the state matrix after r rounds and let $\mathcal{J}$ be a set of bits. Also, let σ and σ' be linear combinations of bits in the set $\mathcal{J}$, i.e., $\sigma = \left(\bigoplus_{(i,j)\in\mathcal{J}} x_{i,j}^{(r)}\right), \sigma' = \left(\bigoplus_{(i,j)\in\mathcal{J}} x_{i,j}'^{(r)}\right)$. Then $\Delta\sigma = \left(\bigoplus_{(i,j)\in\mathcal{J}} \Delta x_{i,j}^{(r)}\right)$ is the linear combination of the differentials. We can write $\Pr\left[\Delta\sigma = 0 | \Delta X^{(0)}\right] = \frac{1}{2}(1 + \varepsilon_d)$, where ε_d is the differential correlation.

Using linear cryptanalysis, it is possible to go further and find new relations between the initial state and the state after $R > r$ rounds. To do so, let $\mathcal{L}$ denote another set of bits and define $\rho = \left(\bigoplus_{(i,j)\in\mathcal{L}} x_{i,j}^{(R)}\right), \rho' = \left(\bigoplus_{(i,j)\in\mathcal{L}} x_{i,j}'^{(R)}\right)$. Then, as before, $\Delta\rho = \left(\bigoplus_{(i,j)\in\mathcal{L}} \Delta x_{i,j}^{(R)}\right)$. We can define $\Pr[\sigma = \rho] = \frac{1}{2}(1 + \varepsilon_L)$, where ε_L is the linear correlation. We want to find γ such that $\Pr\left[\Delta\rho = 0 | \Delta X^{(0)}\right] = \frac{1}{2}(1 + \gamma)$. To compute γ, we write (to simplify the notation we make the conditional to $\Delta X^{(0)}$ implicit):

$$\Pr[\Delta\sigma = \Delta\rho] = \Pr[\sigma = \rho] \cdot \Pr[\sigma' = \rho'] + \Pr[\sigma = \bar{\rho}] \cdot \Pr\left[\sigma' = \overline{\rho'}\right] = \frac{1}{2}\left(1 + \varepsilon_L^2\right).$$

Then, $\Pr[\Delta\rho = 0] = \frac{1}{2}\left(1 + \varepsilon_d \cdot \varepsilon_L^2\right)$. Therefore, the differential-linear correlation is given by $\gamma = \varepsilon_d \cdot \varepsilon_L^2$, which defines a distinguisher with complexity $\mathcal{O}(\varepsilon_d^{-2}\varepsilon_L^{-4})$. For further information on differential-linear cryptanalysis we refer to [7].

2.4 Probabilistic Neutral Bits

This section reviews the attack of Aumasson et al. [2]. The attack first identifies good choices of truncated differentials, then it uses probabilistic backwards computation with the notion of Probabilistic Neutral Bits (PNB), and, finally, it estimates the complexity of the attack. In [2], the $\mathcal{ID}$ is defined for a single-bit

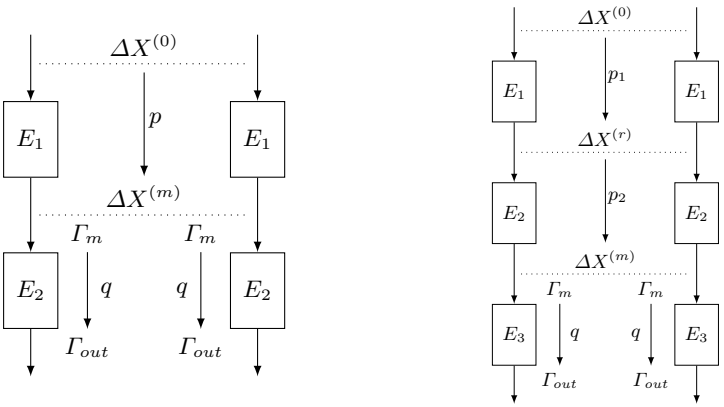


Fig. 1: A classical differential-linear distinguisher (on the left) and a differential-linear distinguisher with experimental evaluation of the correlation p_2 (on the right). E is divided into sub-ciphers $E = E_2 \circ E_1$, or $E = E_3 \circ E_2 \circ E_1$. In the differential part we may apply an $\mathcal{ID}$ $\Delta X^{(0)}$ in the sub cipher E_1 obtaining an $\mathcal{OD}$ $\Delta X^{(m)}$ after m rounds. The next step is to apply Linear Cryptanalysis using masks Γ_m and Γ_{out}. Applying this technique we can construct a differential-linear distinguisher of the cipher E. One way to improve attacks is to explore properties of the cipher in the first part E_1 (on the right), and then apply a differential linear attack where we divide the differential part of the attack in two.

difference $\Delta x_{i,j}^{(0)} = 1$ and a single-bit $\mathcal{OD}$ after r rounds $\Delta x_{p,q}^{(r)}$, such differential is denoted $(\Delta x_{p,q}^{(r)} | \Delta x_{i,j}^{(0)})$ and it has correlation ε_d.

Assume that the differential is fixed, and we observe outputs Z and Z' of $R = l + r$ rounds for nonce v, counter t and unknown key k. If we guess the key k we can invert l rounds of the algorithm to get $X^{(r)}$ and $X'^{(r)}$ and compute $\Delta x_{p,q}^{(r)}$. Then, let f be the function which executes this procedure, i.e., $f(k, v, t, Z, Z') = \Delta x_{p,q}^{(r)}$. Hence, we expect that $f(\hat{k}, v, t, Z, Z')$ has correlation ε_d only if $\hat{k} = k$. Then, if we have several pairs of Z and Z', it is possible to test our guesses for k. Thus, we can search only over a subkey of $m = 256 - n$ bits, provided we can find a function g that approximates f but only uses m key bits as input. Then, let $\bar{k}$ correspond to the subkey of m bits of key k and let f to be correlated to g with correlation ε_a, i.e., $\Pr(f(k, v, t, Z, Z') = g(\bar{k}, v, t, Z, Z')) = \frac{1}{2}(1 + \varepsilon_a)$.

If we denote the correlation of g by ε we can approximate ε by $\varepsilon_d \varepsilon_a$. The problem that remains is how to efficiently find such a function g. In [2], this is done by first identifying key bits that have little influence on the result of $f(k, v, t, Z, Z')$, these are called *probabilistic neutral bits* (PNBs). This is done by defining the *neutrality measure* $\gamma_{i,j}$ of a key bit $k_{i,j}$. After computing $\gamma_{i,j}$ (see [2] for a method of estimation), for all $i = (0, 1, ..., 7)$ and $j = (0, 1, ..., 31)$, we can define the set of significant key bits as $\Psi = \{(i, j) : \gamma_{i,j} \leq \gamma\}$ where γ is a threshold value, and then define our approximation g as $g(k_\Psi, v, t, Z, Z') =$

$f(k^*, v, t, Z, Z')$ where k_Ψ is defined as the subkey with key bits in the set Ψ and k^* is computed from k_Ψ by setting $k_{i,j} = 0$ for all $(i, j) \notin \Psi$.

We refer to [2] for further information about the estimation of the data and time complexity of the attack and for further details on the described technique. We also note that Dey et al. [14] provided new formulas to compute the complexities, correcting some problems with previous formulas.

2.5 Linear approximations for ARX ciphers

To attack Salsa and ChaCha, only two simple approximations to the carry function have been used. Let $\Theta(x, y) = x \oplus y \oplus (x + y)$ be the carry function of the sum $x + y$. Define $\Theta_i(x, y)$ as the i-th bit of $\Theta(x, y)$. By definition, we have $\Theta_0(x, y) = 0$. Using Theorem 3 of Wallén [29], we can generate all possible linear approximations with a given correlation. In particular, at Eurocrypt 2021, Coutinho and Souza [11] used the following linear approximations:

$$\Pr(\Theta_i(x, y) = y_{i-1}) = \frac{1}{2}\left(1 + \frac{1}{2}\right), i > 0. \tag{7}$$

$$\Pr(\Theta_i(x, y) \oplus \Theta_{i-1}(x, y) = 0) = \frac{1}{2}\left(1 + \frac{1}{2}\right), i > 0. \tag{8}$$

As Coutinho and Souza explained, by combining Eqs. 7 and 8 when attacking ARX ciphers we can create a strategy to improve linear approximations when considering more rounds. The main idea is that when using Eq. 7 in one round we will create consecutive terms that can be expanded together using Eq. 8.

Next, we review previous linear approximations for Salsa and ChaCha.

Linear Approximations for Salsa In the following, we review the work of [9] using the notation of Coutinho and Souza [11]. We can write the QRF equations of Salsa (Eq. 2) as

$$x_{b,i}^{(m)} = x_{b,i}^{(m-1)} \oplus x_{a,i-7}^{(m-1)} \oplus x_{d,i-7}^{(m-1)} \oplus \Theta_{i-7}(x_d^{(m-1)}, x_a^{(m-1)}) \tag{9}$$

$$x_{c,i}^{(m)} = x_{c,i}^{(m-1)} \oplus x_{b,i-9}^{(m)} \oplus x_{a,i-9}^{(m-1)} \oplus \Theta_{i-9}(x_a^{(m-1)}, x_b^{(m)}) \tag{10}$$

$$x_{d,i}^{(m)} = x_{d,i}^{(m-1)} \oplus x_{c,i-13}^{(m)} \oplus x_{b,i-13}^{(m)} \oplus \Theta_{i-13}(x_c^{(m)}, x_b^{(m)}) \tag{11}$$

$$x_{a,i}^{(m)} = x_{a,i}^{(m-1)} \oplus x_{d,i-18}^{(m)} \oplus x_{c,i-18}^{(m)} \oplus \Theta_{i-18}(x_d^{(m)}, x_c^{(m)}) \tag{12}$$

Inverting these equations and changing to positive indexes, we get:

$$x_{a,i}^{(m-1)} = \mathcal{L}_{a,i}^{(m)} \oplus \Theta_{i+14}(x_d^{(m)}, x_c^{(m)}) \tag{13}$$

$$x_{d,i}^{(m-1)} = \mathcal{L}_{d,i}^{(m)} \oplus \Theta_{i+19}(x_c^{(m)}, x_b^{(m)}) \tag{14}$$

$$x_{c,i}^{(m-1)} = \mathcal{L}_{c,i}^{(m)} \oplus \Theta_{i+23}(x_a^{(m-1)}, x_b^{(m)}) \oplus \Theta_{i+5}(x_d^{(m)}, x_c^{(m)}) \tag{15}$$

$$x_{b,i}^{(m-1)} = \mathcal{L}_{b,i}^{(m)} \oplus \Theta_{i+25}(x_d^{(m-1)}, x_a^{(m-1)}) \oplus \Theta_{i+7}(x_d^{(m)}, x_c^{(m)}) \oplus \Theta_{i+12}(x_c^{(m)}, x_b^{(m)}) \tag{16}$$

where

$$\mathcal{L}_{a,i}^{(m)} = x_{a,i}^{(m)} \oplus x_{d,i+14}^{(m)} \oplus x_{c,i+14}^{(m)} \tag{17}$$

$$\mathcal{L}_{b,i}^{(m)} = x_{b,i}^{(m)} \oplus x_{a,i+25}^{(m)} \oplus x_{d,i+7}^{(m)} \oplus x_{c,i+7}^{(m)} \oplus x_{d,i+25}^{(m)} \oplus x_{c,i+12}^{(m)} \oplus x_{b,i+12}^{(m)} \tag{18}$$

$$\mathcal{L}_{c,i}^{(m)} = x_{c,i}^{(m)} \oplus x_{b,i+23}^{(m)} \oplus x_{a,i+23}^{(m)} \oplus x_{d,i+5}^{(m)} \oplus x_{c,i+5}^{(m)} \tag{19}$$

$$\mathcal{L}_{d,i}^{(m)} = x_{d,i}^{(m)} \oplus x_{c,i+19}^{(m)} \oplus x_{b,i+19}^{(m)} \tag{20}$$

From Eq. (7) and these equations is possible to derive the following result:

Lemma 1. *For Salsa's QRF, the following linear approximations hold*

Equation	*Probability*	*Condition*
$x_{a,18}^{(m-1)} = \mathcal{L}_{a,18}^{(m)}$	1	-
$x_{a,i}^{(m-1)} = \mathcal{L}_{a,i}^{(m)} \oplus x_{c,i+13}^{(m)}$	$\frac{1}{2}(1+\frac{1}{2})$	$i \neq 18$
$x_{d,13}^{(m-1)} = \mathcal{L}_{d,13}^{(m)}$	1	-
$x_{d,i}^{(m-1)} = \mathcal{L}_{d,i}^{(m)} \oplus x_{b,i+18}^{(m)}$	$\frac{1}{2}(1+\frac{1}{2})$	$i \neq 13$
$x_{c,9}^{(m-1)} = \mathcal{L}_{c,9}^{(m)} \oplus x_{c,13}^{(m)}$	$\frac{1}{2}(1+\frac{1}{2})$	-
$x_{c,27}^{(m-1)} = \mathcal{L}_{c,27}^{(m)} \oplus x_{b,17}^{(m)}$	$\frac{1}{2}(1+\frac{1}{2})$	-
$x_{c,i}^{(m-1)} = \mathcal{L}_{c,i}^{(m)} \oplus x_{a,i+22}^{(m)}$	$\frac{1}{2}(1-\frac{1}{4})$	$i \neq 9, 27$
$x_{b,7}^{(m-1)} = \mathcal{L}_{b,7}^{(m)} \oplus x_{c,13}^{(m)} \oplus x_{b,18}^{(m)}$	$\frac{1}{2}(1+\frac{1}{4})$	-
$x_{b,20}^{(m-1)} = \mathcal{L}_{b,20}^{(m)} \oplus x_{a,12}^{(m)}$	$\frac{1}{2}(1-\frac{1}{4})$	-
$x_{b,25}^{(m-1)} = \mathcal{L}_{b,25}^{(m)} \oplus x_{d,17}^{(m)}$	$\frac{1}{2}(1-\frac{1}{4})$	-
$x_{b,i}^{(m-1)} = \mathcal{L}_{b,i}^{(m)} \oplus x_{a,i+24}^{(m)} \oplus x_{b,i+11}^{(m)}$	$\frac{1}{2}(1-\frac{1}{8})$	$i \neq 7, 20, 25$

Proof. See Lemmas 2 and 7 of [9]. □

Linear approximations for ChaCha In this section, we review the work presented in [9], [10], and in [11]. Since there are many results presented in these papers, here we focus only on the linear approximations that we will need throughout this paper.

Lemma 2. *(Lemma 9 of [9] combined with Lemma 6 of [11]) For one active input bit in round $m-1$ and multiple active output bits in round m, the following holds for $i > 0$.*

$$x_{b,i}^{(m-1)} = x_{b,i+19}^{(m)} \oplus x_{c,i}^{(m)} \oplus x_{c,i+12}^{(m)} \oplus x_{d,i}^{(m)} \oplus x_{d,i-1}^{(m)}, \quad w.p.\ \tfrac{1}{2}\left(1+\tfrac{1}{2}\right)$$

$$x_{a,i}^{(m-1)} = x_{a,i}^{(m)} \oplus x_{b,i+7}^{(m)} \oplus x_{b,i+19}^{(m)} \oplus x_{c,i+12}^{(m)} \oplus x_{d,i}^{(m)} \oplus x_{b,i+6}^{(m)} \oplus x_{b,i+18}^{(m)} \oplus x_{c,i+11}^{(m)} \oplus x_{d,i-1}^{(m)}, \quad w.p.\ \tfrac{1}{2}\left(1+\tfrac{1}{2^3}\right)$$

$$x_{c,i}^{(m-1)} = x_{a,i}^{(m)} \oplus x_{c,i}^{(m)} \oplus x_{d,i}^{(m)} \oplus x_{d,i+8}^{(m)} \oplus x_{a,i-1}^{(m)} \oplus x_{d,i+7}^{(m)} \oplus x_{d,i-1}^{(m)}, \quad w.p.\ \tfrac{1}{2}\left(1+\tfrac{1}{2^2}\right)$$

$$x_{d,i}^{(m-1)} = x_{a,i}^{(m)} \oplus x_{a,i+16}^{(m)} \oplus x_{b,i+7}^{(m)} \oplus x_{c,i}^{(m)} \oplus x_{d,i+24}^{(m)} \oplus x_{c,i-1}^{(m)} \oplus x_{b,i+6}^{(m)}, \quad w.p.\ \tfrac{1}{2}\left(1+\tfrac{1}{2}\right)$$

Proof. See [9] and [11]. We provide an alternative proof of this lemma in Section 3. □

Lemma 3. *(Lemma 10 of [11]) The following linear approximation holds with probability* $\frac{1}{2}\left(1+\frac{1}{2^8}\right)$

$$\begin{aligned} x_{3,0}^{(3)} \oplus x_{4,0}^{(3)} = {} & x_0^{(6)}[0,16] \oplus x_1^{(6)}[0,6,7,11,12,22,23] \oplus x_2^{(6)}[0,6,7,8,16,18,\\ & 19,24] \oplus x_4^{(6)}[7,13,19] \oplus x_5^{(6)}[7] \oplus x_6^{(6)}[7,13,14,19] \oplus \\ & x_7^{(6)}[6,7,14,15,26] \oplus x_8^{(6)}[0,7,8,19,31] \oplus x_9^{(6)}[0,6,12,26] \oplus \\ & x_{10}^{(6)}[0] \oplus x_{11}^{(6)}[6,7] \oplus x_{12}^{(6)}[0,11,12,19,20,30,31] \oplus \\ & x_{13}^{(6)}[0,14,15,24,26,27] \oplus x_{14}^{(6)}[8,25,26] \oplus x_{15}^{(6)}[24]. \end{aligned}$$

Proof. See [11]. □

3 A more effective approach to derive linear approximations for ChaCha

In this section, we propose a new approach to the derivation of linear approximations for ChaCha. To do so, instead of considering the QRF as in previous works, here we will consider the SRF, as defined in Eq. (5). We point out that we used the techniques of this section to implement automatic linear expansions of ChaCha in *CryptDances*.

3.1 New framework: linear approximations to the SRF

From Eq. (5), we can write the SRF equations of ChaCha as

$$\begin{array}{ll} x_{a,i}^{[s]} = x_{a,i}^{[s-1]} \oplus x_{b,i}^{[s-1]} \oplus \Theta_i(x_a^{[s-1]}, x_b^{[s-1]}); & x_{d,i+r_1}^{[s]} = x_{d,i}^{[s-1]} \oplus x_{a,i}^{[s]}; \\ x_{c,i}^{[s]} = x_{c,i}^{[s-1]} \oplus x_{d,i}^{[s]} \oplus \Theta_i(x_c^{[s-1]}, x_d^{[s]}); & x_{b,i+r_2}^{[s]} = x_{b,i}^{[s-1]} \oplus x_{c,i}^{[s]}; \end{array} \tag{21}$$

Inverting these equations, we get:

$$x_{b,i}^{[s-1]} = x_{b,i+r_2}^{[s]} \oplus x_{c,i}^{[s]} \tag{22}$$

$$x_{c,i}^{[s-1]} = x_{c,i}^{[s]} \oplus x_{d,i}^{[s]} \oplus \Theta_i(x_c^{[s-1]}, x_d^{[s]}) \tag{23}$$

$$x_{d,i}^{[s-1]} = x_{a,i}^{[s]} \oplus x_{d,i+r_1}^{[s]} \tag{24}$$

$$x_{a,i}^{[s-1]} = x_{a,i}^{[s]} \oplus x_{b,i+r_2}^{[s]} \oplus x_{c,i}^{[s]} \oplus \Theta_i(x_a^{[s-1]}, x_b^{[s-1]}) \tag{25}$$

Note that the expansions for $x_{b,i}^{[s-1]}$ and $x_{d,i}^{[s-1]}$ are deterministic. Therefore, we only need to focus on expansions for $x_{a,i}^{[s-1]}$ and $x_{c,i}^{[s-1]}$. To this end, consider the following three lemmas:

Lemma 4. *Consider the* SR_{ChaCha} *with rotation distances* r_1 *and* r_2*. Then we have that* $x_{c,0}^{[s-1]} = x_{c,0}^{[s]} \oplus x_{d,0}^{[s]}$ *and* $x_{a,0}^{[s-1]} = x_{a,0}^{[s]} \oplus x_{b,r_2}^{[s]} \oplus x_{c,0}^{[s]}$.

Proof. The proof follows from Eqs. (23) and (25) and using $\Theta_0(.) = 0$. □

Lemma 5. *For one active input bit in subround $s-1$ and multiple output bits in subround s, the following linear approximations hold with probability $\frac{1}{2}(1+\frac{1}{2})$ for the function SR_{ChaCha} with rotation distances r_1 and r_2 when $i > 0$*

$$x_{c,i}^{[s-1]} = x_{c,i}^{[s]} \oplus x_{d,i}^{[s]} \oplus x_{d,i-1}^{[s]},$$
$$x_{a,i}^{[s-1]} = x_{a,i}^{[s]} \oplus x_{b,i+r_2}^{[s]} \oplus x_{c,i}^{[s]} \oplus x_{b,i+r_2-1}^{[s]} \oplus x_{c,i-1}^{[s]}.$$

Proof. The proof follows directly from the application of Eq. (7) in Eqs. (23) and (25). □

Lemma 6. *For two active input bits in subround $s-1$ and multiple output bits in subround s, the following linear approximations hold with probability $\frac{1}{2}(1+\frac{1}{2})$ for the function SR_{ChaCha} with rotation distances r_1 and r_2*

$$x_{c,i}^{[s-1]} \oplus x_{c,i-1}^{[s-1]} = x_{c,i}^{[s]} \oplus x_{d,i}^{[s]} \oplus x_{c,i-1}^{[s]} \oplus x_{d,i-1}^{[s]},$$
$$x_{a,i}^{[s-1]} \oplus x_{a,i-1}^{[s-1]} = x_{a,i}^{[s]} \oplus x_{b,i+r_2}^{[s]} \oplus x_{c,i}^{[s]} \oplus x_{a,i-1}^{[s]} \oplus x_{b,i+r_2-1}^{[s]} \oplus x_{c,i-1}^{[s]}.$$

Proof. The proof follows directly from the application of Eq. (8) after expanding the left side of the equations with Eqs. (23) and (25). □

As we will show, from these three Lemmas it is possible to reproduce previous works. Before that, we show an additional lemma that we use to improve previous results.

Lemma 7. *For two active input bits in subround $s-1$ and multiple output bits in subround s, the following linear approximations hold with probability $\frac{1}{2}(1+\frac{1}{2})$ for the function SR_{ChaCha} with rotation distances r_1 and r_2*

$$x_{c,i}^{[s-1]} \oplus x_{c,i-1}^{[s-1]} = x_{c,i}^{[s]} \oplus x_{d,i}^{[s]},$$
$$x_{a,i}^{[s-1]} \oplus x_{a,i-1}^{[s-1]} = x_{a,i}^{[s]} \oplus x_{b,i+r_2}^{[s]} \oplus x_{c,i}^{[s]}.$$

Proof. See the extended version of this paper. □

Strategies As the reader may have noticed, Lemmas 6 and 7 are actually expanding the same pair of bits. Then we may ask which is the best choice. However, it depends on the situation. As a general rule, we always look for minimizing the number of active bits in the equations. That is because fewer terms means fewer expansions which means a higher correlation (usually). From this assertion, the reader might conclude that Lemma 7 is better. Notice, however, that adjacent bits are always expanded together (due to Lemma 6) and should be counted as one. Therefore, the best rule will be the one that results in other bits being canceled (see the extended version of this paper for a complete example). We conclude that each situation needs to be evaluated individually by considering all options to reach the best possible linear approximation.

3.2 Deriving linear approximations of previous works using the new approach

The new framework proposed in Section 3 is simpler to understand and to use when compared with previous works. For example, the methods of Coutinho and Souza [11] encompasses at least 18 different rules to derive linear approximations for ChaCha. Of course, being simpler is not enough, as the proposed framework should also be at least as effective. Our claim is that using Lemmas 4, 5, and 6 is possible to derive most of the linear approximations (if not all) of previous works. Of course, proving that to each one of them individually would be an extremely tedious task. Therefore, here we will just prove this result to Lemma 2 that is the base to generate almost all linear approximations of ChaCha in the literature, we leave the rest as a conjecture.

Proposition 1. *Lemma 2 is a consequence of Lemmas 5 and 6.*

Proof. See the extended version of this paper. □

3.3 Improve linear approximations and differential-linear distinguisher for ChaCha

In this section, we improve the best differential-linear distinguisher against ChaCha by improving its linear part by using the framework of Section 3.1. We highlight that the improvements are achieved through an intelligent use of Lemma 7. The new result is given by the following lemma

Lemma 8. *The following linear approximation holds with probability* $\frac{1}{2}\left(1+\frac{1}{2^{53}}\right)$

$$x_{3,0}^{[6]} \oplus x_{4,0}^{[6]} = x_0^{[14]}[0,3,4,7,8,11,12,14,15,18,20,27,28] \oplus x_1^{[14]}[0,5,7,8,10,14,15,16,22,23,24,25,27,30,31] \oplus x_2^{[14]}[7,9,10,16,19,25,26] \oplus x_3^{[14]}[6,7,8,24] \oplus x_4^{[14]}[0,2,3,5,18,22,23,27] \oplus x_5^{[14]}[1,2,9,10,13,14,18,21,22,25,29] \oplus x_6^{[14]}[0,2,3,7,10,11,13,14,19,22,23,25,27,31] \oplus x_7^{[14]}[1,2,13,25,26,30,31] \oplus x_8^{[14]}[8,11,13,20,25,27,28,30,31] \oplus x_9^{[14]}[2,3,6,7,11,14,15,18,23,27] \oplus x_{10}^{[14]}[0,3,4,6,8,12,13,14,18,20,23,25,27,28] \oplus x_{11}^{[14]}[6,14,15,18,19,23,24,27] \oplus x_{12}^{[14]}[3,4,6,11,13,22,23,24,26,27,30,31] \oplus x_{13}^{[14]}[1,2,6,7,8,13,14,16,18,20,22,23,24,25,26] \oplus x_{14}^{[14]}[0,7,13,14,15,16,17,18,23,24] \oplus x_{15}^{[14]}[16,25,26]$$

Proof. We present just a sketch of the proof, for the complete proof see the extended version of this paper. We start from the linear approximation of Lemma 3. Notice that since we are transitioning from round 6 to 7 (subrounds 12 to 14), we have $(a,b,c,d) \in \{(0,4,8,12),(1,5,9,13),(2,6,10,14),(3,7,11,15)\}$. Therefore, we can divide the bits of the equation in 4 distinct groups:

- Group I - $x_0^{[12]}[0,16], x_4^{[12]}[7,13,19], x_8^{[12]}[0,7,8,19,31], x_{12}^{[12]}[0,11,12,19,20,30,31]$.
- Group II - $x_1^{[12]}[0,6,7,11,12,22,23], x_5^{[12]}[7], x_9^{[12]}[0,6,12,26], x_{13}^{[12]}[0,14,15,24,26,27]$.

– Group III - $x_2^{[12]}[0,6,7,8,16,18,19,24], x_6^{[12]}[7,13,14,19]$, $x_{10}^{[12]}[0]$, $x_{14}^{[12]}[8,25,26]$.
– Group IV - $x_7^{[12]}[6,7,14,15,26]$, $x_{11}^{[12]}[6,7], x_{15}^{[12]}[24]$.

We divide the proof for each group, and the proof for Group I and Group IV is identical as the one of Lemma 11 of [11], with probabilities $\frac{1}{2}\left(1+\frac{1}{2^{12}}\right)$ and $\frac{1}{2}\left(1+\frac{1}{2^4}\right)$, respectively. For Group II, it is possible to show that

$$\begin{aligned} &x_1^{[12]}[0,6,7,11,12,22,23] \oplus x_5^{[12]}[7] \oplus x_9^{[12]}[0,6,12,26] \oplus \\ &x_{13}^{[12]}[0,14,15,24,26,27] = x_1^{[14]}[0,5,7,8,10,14,15,16,22,23,24,25,27, \\ &30,31] \oplus x_5^{[14]}[1,2,9,10,13,14,18,21,22,25,29] \oplus x_9^{[14]}[2,3,6,7,11,14, \\ &15,18,23,27] \oplus x_{13}^{[14]}[1,2,6,7,8,13,14,16,18,20,22,23,24,25,26], \end{aligned} \tag{26}$$

with probability $\frac{1}{2}\left(1+\frac{1}{2^{14}}\right)$. And for Group III, we get

$$\begin{aligned} &x_2^{[12]}[0,6,7,8,16,18,19,24] \oplus x_6^{[12]}[7,13,14,19] \oplus x_{10}^{[12]}[0] \oplus x_{14}^{[12]}[8,25,26] = \\ &x_2^{[14]}[7,9,10,16,19,25,26] \oplus x_6^{[14]}[0,2,3,7,10,11,13,14,19,22,23,25,27,31] \oplus \\ &x_{10}^{[14]}[0,3,4,6,8,12,13,14,18,20,23,25,27,28] \oplus \\ &x_{14}^{[14]}[0,7,13,14,15,16,17,18,23,24], \end{aligned} \tag{27}$$

with probability $\frac{1}{2}\left(1+\frac{1}{2^{15}}\right)$. Aggregating the correlation via the Piling-up Lemma completes the proof. □

Computational Result 1 *The linear approximations of Eqs.* (26) *and* (27) *hold computationally with* $\varepsilon_{L_2} = 0.000201 \approx 2^{-12.31}$ *and* $\varepsilon_{L_3} = 0.000141 \approx 2^{-12.813}$*, respectively. These correlations were verified using* 2^{42} *random samples.*

Finally, we compute the differential-linear distinguisher. For that, we use the differential correlation $\varepsilon_d = 0.00048$ for $(a,b) = (3,4)$ described in [10], and the Computational Results 1, 2 and 5 of [11] for linear correlations $\varepsilon_{L_0} = 0.006942$, $\varepsilon_{L_1} = 0.000301$, and $\varepsilon_{L_4} = 0.0625$, respectively. Additionally, we use our Computational Result 1 for the linear correlations ε_{L_2} and ε_{L_3}. From that, we get $\varepsilon_d(\varepsilon_{L_0}\varepsilon_{L_1}\varepsilon_{L_2}\varepsilon_{L_3}\varepsilon_{L_4})^2 \approx 2^{-107}$ which gives us a distinguisher for 7 rounds of ChaCha with complexity approximately 2^{214}.

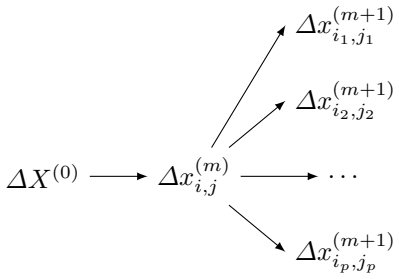
4 Bidirectional Linear Expansions

In this section, we propose a new technique called Bidirectional Linear Expansions (BLE). This section is divided in three parts: in Section 4.1 we present BLE. In Sections 4.2 and 4.3, we use BLE to study Salsa and ChaCha, respectively.

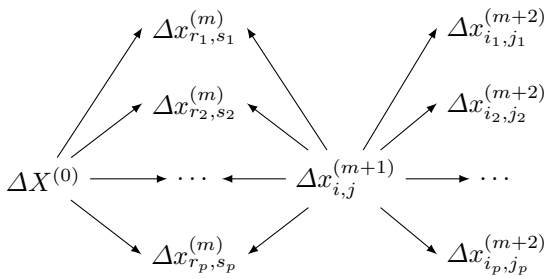
4.1 Proposed technique

Previous works on the cryptanalysis of Salsa and ChaCha used an intensive computational approach to find significant correlations for the differential part

of the attacks. To do so, authors considered an $\mathcal{ID}$ $\Delta X^{(0)}$ and used several random simulations to estimate a correlation for a single bit $\Delta x_{i,j}^{(m)}$. From this point, this single bit was expanded into several bits using linear approximations, like in the following diagram:



In this work, we propose a different approach. More precisely, we expand a single bit in both forward and backward directions. Therefore, in the differential part we need to find a correlation for a combination of bits instead of just one. This approach leads to the worst differential correlations, however it improves the linear correlations. Since the linear part has a higher weight on the complexity of the attack, the proposed technique leads to better results overall. We illustrate the proposed technique in the following diagram:



This technique is useful to find differentials that reach more rounds. The reason is that when we try to find differentials experimentally we have two parameters to set: (1) the number of differentials to be tested D; and (2) the number of random samples N to estimate the differential. Since for each differential we need to execute the algorithm two times, then we need $2DN$ executions to try to find successful differential correlations. However, as the number of rounds increases, the correlations decreases, then we have to increase N accordingly. Because of that, the computation quickly becomes infeasible.

Using BLE we can leverage the backward linear approximation to search for correlations in the previous round. For example, suppose that we compute all possible single bit differentials for m rounds of ChaCha and that we have a backward linear approximation $x_{i,j}^{(m+1)} = x_{r_1,s_1}^{(m)} \oplus x_{r_2,s_2}^{(m)} \oplus ... \oplus x_{r_p,s_p}^{(m)}$. Then, we can use the Piling-up Lemma to aggregate the correlation for each single bit from the

previous round, achieving a differential correlation for further rounds. Mathematically, if we define $\Pr(\Delta x^{(m)}_{r_k,s_k}|\mathcal{ID}) = \frac{1}{2}(1+\varepsilon_k)$, and $\Pr(\Delta x^{(m+1)}_{i,j}|\mathcal{ID}) = \frac{1}{2}(1+\varepsilon_d)$, then we can estimate $\varepsilon_d = \prod_{k=1}^{p} \varepsilon_k$.

4.2 Applying BLE to Salsa

Next, we use the techniques proposed in Section 4.1 to improve the attacks against Salsa. This section is divided in three parts: first we present the first single bit differential reaching 5 rounds of Salsa. Then, we present new linear approximations for Salsa, starting from the proposed differential. Finally, we use these results to improve attacks against Salsa.

Proposed differential for 5 rounds of Salsa In this section, we present a new single bit differential correlation for 5 rounds of Salsa, constructed by applying the technique proposed in the previous section. To do so, first notice from Eq. (9), that we can write $x^{(5)}_{b,7} = x^{(4)}_{b,7} \oplus x^{(4)}_{a,0} \oplus x^{(4)}_{d,0}$, with probability 1, where $(a, b, d) \in \{(0, 4, 12), (5, 13, 1), (10, 2, 6), (15, 7, 11)\}$. Using this relationship, we will find a correlation for a bit in the fifth round $x^{(5)}_{b,7}$ by combining the correlation of three other bits in the fourth round.

To achieve this result, we start from the single bit $\mathcal{ID}$ of $\Delta x^{(0)}_{7,31} = 1$, proposed by Aumasson et al. [2], which is the one that provides the highest correlations presented in the literature. However, instead of relying on computational results only, we expanded the first round theoretically and used the techniques proposed by Beierle et al. [3] (see Section 2.3) to find differentials with amplified probabilities. Here, we apply the techniques proposed by Lipmaa and Moriai on efficient algorithms for computing differential properties of addition [22]. In the referred work, the authors define the Differential Probability of Addition (DPA) modulo 2^n as a triplet of two input and one output differences, denoted as $(\alpha, \beta \rightarrow \gamma)$, where $\alpha, \beta, \gamma \in \mathbb{F}_2^n$, and is defined as $\mathrm{DP}^+(\delta) = \mathrm{DP}^+(\alpha, \beta \rightarrow \gamma) := \Pr_{x,y}[(x + y) \oplus ((x \oplus \alpha) + (y \oplus \beta)) = \gamma]$.

One important question is how to find γ such that $\mathrm{DP}^+(\delta)$ is maximum given α and β. In other words, we want to find $\mathrm{DP}^+_{\max}(\alpha, \beta) := \max_\gamma \mathrm{DP}^+(\alpha, \beta \rightarrow \gamma)$. In [22], the authors provide two important algorithms to compute $\mathrm{DP}^+_{\max}(\alpha, \beta)$. Specifically, Algorithm 3 of [22] returns all (α, β)-optimal output differences γ, and Algorithm 4 of [22] finds an (α, β)-optimal γ in log-time.

Thus, starting from the $\mathcal{ID}$ given by $\Delta X^{(0)}$, we propagated the differential using the algorithms from [22] and chose the one that minimized the hamming weight, from this we get (in hexadecimal notation):

$$\Psi = \Delta X^{(1)} = \begin{pmatrix} 0\ 0\ 0 & 0x00000000 \\ 0\ 0\ 0 & 0x80000000 \\ 0\ 0\ 0 & 0x00001000 \\ 0\ 0\ 0 & 0x40020000 \end{pmatrix}.$$

The probability that $\Delta X^{(0)}$ leads to $\Delta X^{(1)}$ is 2^{-1}. To compute this probability, we used Algorithm 2 of [22]. At this point, we used the strategy of Beierle

et al. [3] (see Section 2.3) to find differentials with amplified probabilities. We may apply this technique because, as with ChaCha, the QRF of Salsa is independently applied to each column in the first round. Therefore, when the output difference of one QRF is restricted, the input of the other three QR functions is trivially independent of the output difference. It implies that we have 96 independent bits, and we can easily amplify the probability of the differential-linear distinguisher.

We summarize the differential part combined with the backward linear expansion of the proposed attacks in the diagram of Figure 2.

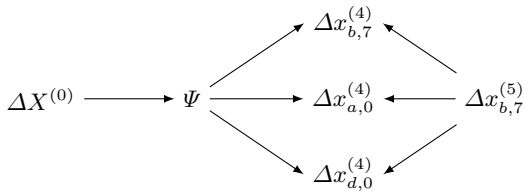


Fig. 2: Differential part of the proposed attack.

Considering Figure 2, we need to estimate the transition probability from Ψ to $\Delta x^{(5)}_{b,7}$. We performed this task computationally, and we achieved the best results when considering $b = 4$. Thus, consider the following computational result:

Computational Result 2 *The following differentials were found computationally using* 2^{45} *random samples.*

$\mathcal{ID}$	$\mathcal{OD}$	*Correlation*
$\Delta X^{(1)} = \Psi$	$\Delta x^{(4)}_{0,0}$	-0.00000159
$\Delta X^{(1)} = \Psi$	$\Delta x^{(4)}_{4,7}$	-0.00085
$\Delta X^{(1)} = \Psi$	$\Delta x^{(4)}_{12,0}$	0.000167

From this result, we can use the Piling-Up Lemma to reach a differential correlation from round 1 to round 5 of Salsa. More precisely, we can write

$$\Pr(\Delta x^{(5)}_{4,7} = 0 | \Delta X^{(1)} = \Psi) = \frac{1}{2}(1 + \varepsilon_d), \tag{28}$$

where $\varepsilon_d \approx 2^{-42.01}$. Unfortunately, checking this correlation is computationally infeasible as it would require approximately 2^{84} samples. We note, however, that we tested if the Piling-up Lemma holds using this technique for ChaCha and Salsa for smaller correlations in fewer rounds. In our tests, the observed correlation was always higher than predicted, therefore, our attack using this correlation is probably better than what we report in this paper.

In the next section, we will present the linear expansion for the bit $x^{(5)}_{b,7}$ to complete the differential-linear distinguisher.

New linear approximations for Salsa First, we propose the following Lemma:

Lemma 9. *For two active input bits in round $m-1$ and multiple active output bits in round m of Salsa, the following holds for $i \notin \mathcal{I}$*

$$x_{\lambda,i}^{(m-1)} \oplus x_{\lambda,i-1}^{(m-1)} = \mathcal{L}_{\lambda,i}^{(m)} \oplus \mathcal{L}_{\lambda,i-1}^{(m)}, \text{ w.p. } \frac{1}{2}\left(1+\frac{1}{2^{\sigma}}\right),$$

where $(\lambda, \sigma, \mathcal{I}) \in \{(a,1,\{18\}), (b,3,\{7,20,25\}), (c,2,\{9,27\}), (d,1,\{13\})\}$ *and* $\mathcal{L}$ *is given in Eqs.* (17)-(20).

Proof. This proof follows from Eqs. (13)-(16) by noting that always we have pair with the form $\Theta_i(x) \oplus \Theta_{i-1}(x)$. When $i > 1$ we apply the approximation of Eq. (8) to get $\Theta_i(x) \oplus \Theta_{i-1}(x) = 0$ with probability $\frac{1}{2}(1+\frac{1}{2})$. When $i = 1$ we use the fact that $\Theta_0(x) = 0$ to get $\Theta_1(x) \oplus \Theta_0(x) = \Theta_1(x) = 0$ again with probability $\frac{1}{2}(1+\frac{1}{2})$. When $i = 0$, $\Theta_i(x) \oplus \Theta_{i-1}(x) \neq 0$, thus we exclude these indexes. All that is left is to use the Piling-Up Lemma to combine the probabilities. □

Next, we consider new linear approximations to the bit $x_{4,7}^{(5)}$.

Lemma 10. *The following linear approximation holds with probability* $\frac{1}{2}\left(1-\frac{1}{2^6}\right)$

$$\begin{aligned} x_{4,7}^{(5)} = {} & x_0^{(7)}[0] \oplus x_2^{(7)}[12,13] \oplus x_3^{(7)}[17] \oplus x_4^{(7)}[7,18,19] \oplus x_6^{(7)}[25,26] \oplus x_7^{(7)}[26,31] \oplus \\ & x_8^{(7)}[13,14,19] \oplus x_{11}^{(7)}[31] \oplus x_{12}^{(7)}[0,14] \oplus x_{14}^{(7)}[12,13] \oplus x_{15}^{(7)}[16,17]. \end{aligned}$$

Proof. From $x_{4,7}^{(5)}$ we use the expansion for $x_{d,i}$ of Lemma 1 to get $x_{4,7}^{(5)} = x_{4,7}^{(6)} \oplus x_{6,25}^{(6)} \oplus x_{6,26}^{(6)} \oplus x_{7,26}^{(6)}$, with probability $\frac{1}{2}\left(1+\frac{1}{2}\right)$. Then, we use the expansion for $x_{b,7}$ and $x_{c,i}$ of Lemma 1 to get $x_{4,7}^{(6)} = \mathcal{L}_{4,7}^{(m)} \oplus x_{8,13}^{(m)} \oplus x_{4,18}^{(m)}$ with probability $\frac{1}{2}\left(1+\frac{1}{4}\right)$, and $x_{7,26}^{(6)} = \mathcal{L}_{7,26}^{(m)} \oplus x_{15,16}^{(m)}$ with probability $\frac{1}{2}\left(1-\frac{1}{4}\right)$. Additionally, using Lemma 9 we get $x_{6,25}^{(6)} \oplus x_{6,26}^{(6)} = \mathcal{L}_{6,25}^{(7)} \oplus \mathcal{L}_{6,26}^{(7)}$, with probability $\frac{1}{2}\left(1+\frac{1}{2}\right)$. Finally, using the Piling-Up Lemma to combine the probabilities completes the proof. □

Lemma 11. *The following linear approximation holds with probability* $\frac{1}{2}\left(1+\frac{1}{2^{34}}\right)$

$$\begin{aligned} x_{4,7}^{(5)} = {} & x_0^{(8)}[0,3,4] \oplus x_2^{(8)}[4,12,14,17,18] \oplus x_3^{(8)}[14,18] \oplus x_4^{(8)}[0,1,4,7,31] \oplus \\ & x_5^{(8)}[16,17,18,19,21,22] \oplus x_6^{(8)}[17,22] \oplus x_7^{(8)}[0,1,4] \oplus \\ & x_8^{(8)}[6,11,13,14,18,24] \oplus x_9^{(8)}[6,18,19] \oplus x_{10}^{(8)}[4,5,9,10,23,24] \oplus \\ & x_{11}^{(8)}[4,5,11,31] \oplus x_{12}^{(8)}[11,12,14,25,26,30,31] \oplus x_{13}^{(8)}[0,7,12,21,26,30] \oplus \\ & x_{14}^{(8)}[12,13,21,25,30,31] \oplus x_{15}^{(8)}[6,7,16,17,24,25]. \end{aligned}$$

Proof. See the extended version of this paper. □

Additionally, we verified the theoretical results of Lemmas 10 and 11 computationally. In particular, for Lemma 11 the experiment is divided in 4 parts leading to the correlations $\varepsilon_{L_1}, \varepsilon_{L_2}, \varepsilon_{L_3}$ and ε_{L_4} (for more details, see the extended version of the paper).

Computational Result 3 *The linear approximation of Lemma 10 holds computationally with* $\varepsilon_{L_0} = -0.015627 \approx -2^{-5.999}$*. This correlation was verified using* 2^{38} *random samples.*

Computational Result 4 *The linear approximations for Lemma 11 hold computationally with correlations* $\varepsilon_{L_1} = 0.083980 \approx 2^{-3.57}$*,* $\varepsilon_{L_2} = 0.007814 \approx 2^{-6.99}$*,* $\varepsilon_{L_3} = 0.006368 \approx 2^{-7.29}$*,* $\varepsilon_{L_4} = 0.002234 \approx 2^{-8.81}$*, respectively. These correlations were verified using* 2^{38} *random samples.*

New Attacks against Salsa Using the linear approximations of Lemma 10 and Lemma 11, the differential correlation $\varepsilon_d \approx -2^{-42.01}$ given in Eq. (28), and the estimated correlations from the Computational Results 3 and 4, we get $\varepsilon_d(\varepsilon_{L_0})^2 \approx 2^{-53.99}$ and $\varepsilon_d(\varepsilon_{L_0}\varepsilon_{L_1}\varepsilon_{L_2}\varepsilon_{L_3}\varepsilon_{L_4})^2 \approx 2^{-107.31}$ which gives us a distinguisher for 7 and 8 rounds of Salsa with complexity less than $2^{-107.98}$ and $2^{-214.62}$, respectively. As in [3], we have to repeat this attack 2 times on average because of the transition probability from $\Delta X^{(0)}$ to $\Delta X^{(1)} = \Psi$. Therefore, we have a distinguisher with data and time complexity of $2^{108.98}$ for Salsa20/7 and $2^{215.62}$ for Salsa20/8.

Additionally, it is straightforward to combine the new differential-linear distinguisher for 5 rounds presented in Eq. 28 with the technique of PNB presented in Section 2.4. More precisely, to use the differential correlation for $\Delta x_{4,7}^{(5)}$, we used the variation of PNB attack described by Beierle in [3]. Thus, consider $(x_{4,7}^{(5)} | \Delta X^{(1)} = \Psi)$. To attack 8 rounds, we need to go back 3 rounds to reach the desired differential. In this case, using $\gamma = 0.3$ we found 152 PNBs , and we obtained $\varepsilon_a = 0.000305$. As in [3], we have to repeat this attack 2 times on average because of the transition probability from $\Delta X^{(0)}$ to $\Delta X^{(1)} = \Psi$. Thus, the final attack has data complexity of $2^{113.14}$ and time complexity $2^{217.14}$.

4.3 Applying BLE against ChaCha

Finding differentials for 3.5 rounds of ChaCha experimentally is very difficult, only a few have been presented in the literature [3,11]. By searching for all possible single bit differentials for 3 rounds of ChaCha we were able to find more than 1000 new differentials using the backward expansion. Unfortunately, we were not able to improve attacks in this case.

5 Forró: a Novel Latin Dance

Although they have a very similar structure, the literature (including this work) suggests that ChaCha is safer than Salsa. Therefore, a natural question that arises is if we can do better with fewer operations, it turns out the answer is yes, and we show how with the design of a new stream cipher named Forró. To do that, in Section 5.1 we will introduce a new concept which we call *Pollination*. Then, in Sections 5.2, 5.3, and 5.4 we present the design, security, and performance of Forró, respectively.

5.1 Pollination

In this section, we propose a new technique that we call *Pollination*. We chose this name as an analogy to the real Pollination in nature: when a bee collects nectar from a flower, the pollen sticks to the hairs of her body. When she visits the next flower, some of this pollen is rubbed off onto the stigma, making fertilization possible. Here, our idea is to use the element that is likely to maximize confusion and diffusion (we call this best element *pollen*) to bring non-linearity and confusion to other elements in the state matrix.

Actually, one of the reasons behind the improved diffusion of ChaCha when compared to Salsa is, in fact, pollination. Since the QRF function updates one element after the other, using the previously updated element as input, then it is a natural consequence that the element updated last ($x_b^{(r)}$) has higher diffusion. In ChaCha, the pattern of application of the QRF actually means that the elements in the second row (which are the parameter $x_b^{(r)}$ for each QRF application), are used to update the first element in the next round. Salsa does not have such a property, hence the improved diffusion of ChaCha.

ChaCha achieves pollination from one round to another, however, it fails to do so within each round because the QRF is applied independently in each column or diagonal. Thus, it is possible to have more diffusion with fewer operations if we create a chain of pollination from one application of the QRF to the other. It can be argued that we will lose parallelism in each round, however, as we will show later, the improved diffusion will allow the same security in fewer rounds, reducing the total number of operations. Also, in Section 5.4, we show that there is another way to explore concurrency inside the processor to achieve better performance.

5.2 Design

Forró's Round Function To deliver pollination from one round to the other we propose to include an extra parameter into the QRF. Nevertheless, we want to maintain (or to decrease) the number of arithmetic operations to achieve competitive performance. Notice that each rotation in Eq. (4) actually makes the same element be updated twice in a row, thus we could update more elements if we had fewer rotations.

Actually, in [6], Bernstein asked the question of whether there should be fewer rotations in the QRF, because rotations account for about 1/3 of the integer operations in Salsa (and also in ChaCha), he wrote:

> "If rotations are simulated by shift-shift-xor (as they are on the UltraSPARC and with XMM instructions) then they account for about 1/2 of the integer operations in Salsa20. Replacing some of the rotations with a comparable number of additions might achieve comparable diffusion in less time."

With those ideas in mind, we define the subround function $X^{[m]} = SR_{forro}(a, b, c, d, e, X^{[m-1]})$ as the following set of operations over indexes a, b, c, d and e

$$\begin{aligned}
x_d'^{(m-1)} &= x_d^{(m-1)} + x_e^{(m-1)}; \quad x_c'^{(m-1)} = x_c^{(m-1)} \oplus x_d'^{(m-1)};\\
x_b'^{(m-1)} &= \left(x_b^{(m-1)} + x_c'^{(m-1)}\right) \lll r_1;\\
x_a'^{(m-1)} &= x_a^{(m-1)} + x_b'^{(m-1)}; \quad x_e^{(m)} = x_e^{(m-1)} \oplus x_a'^{(m-1)};\\
x_d^{(m)} &= \left(x_d'^{(m-1)} + x_e^{(m)}\right) \lll r_2;\\
x_c^{(m)} &= x_c'^{(m-1)} + x_d^{(m)}; \qquad x_b^{(m)} = x_b'^{(m-1)} \oplus x_c^{(m)};\\
x_a^{(m)} &= \left(x_a'^{(m-1)} + x_b^{(m)}\right) \lll r_3;
\end{aligned} \tag{29}$$

where $r_1 = 10, r_2 = 27$ and $r_3 = 8$.

Notice that SR_{forro} has a total of 12 operations, just like QR_{ChaCha}, but fewer rotations. Also, notice that SR_{forro} is asymmetric in the sense that of all elements there is one, namely $x_e^{(r)}$ that is updated less frequently than the others. However, this behavior is actually acceptable since $x_e^{(r)}$ is the element used for pollination, thus its job is to provide non-linearity and confusion and not to gain more necessarily. In addition, except in the first subround, $x_e^{(r)}$ is always updated in the previous subround. Finally, notice that as the element $x_a^{(r)}$ is the last to be updated, then it will likely have the more complex boolean functions in comparison to $x_b^{(r)}, x_c^{(r)}, x_d^{(r)}$ and $x_e^{(r)}$, therefore $x_a^{(r)}$ will become the pollen for the next application of SR_{forro}.

We define each round of Forró in terms of its subrounds. More precisely, each round has 4 subrounds, thus we have $X^{(r)} = X^{[4r]}$ (see Section 2.2). Then, in an odd round, when $r \in \{1, 3, 5, 7, ...\}$, $X^{(r)}$ is defined from $X^{(r-1)}$ in the following manner

$$\begin{aligned}
X^{[4r-3]} &= SR(0, 4, 8, 12, 3, X^{[4r-4]}); \quad X^{[4r-2]} = SR(1, 5, 9, 13, 0, X^{[4r-3]});\\
X^{[4r-1]} &= SR(2, 6, 10, 14, 1, X^{[4r-2]}); \quad X^{[4r]} = SR(3, 7, 11, 15, 2, X^{[4r-1]});
\end{aligned} \tag{30}$$

and for even rounds $r \in \{2, 4, 6, 8, , ...\}$ from

$$\begin{aligned}
X^{[4r-3]} &= SR(0, 5, 10, 15, 3, X^{[4r-4]}); \quad X^{[4r-2]} = SR(1, 6, 11, 12, 0, X^{[4r-3]});\\
X^{[4r-1]} &= SR(2, 7, 8, 13, 1, X^{[4r-2]}); \quad X^{[4r]} = SR(3, 4, 9, 14, 2, X^{[4r-1]});
\end{aligned} \tag{31}$$

Initialization To initialize the state matrix we have 16 integers available, being 8 key words, 2 nonce words, 2 counter words and 4 constants. All positions in the state matrix are different in terms of diffusion and whether it is used sooner or later. Forró's initialization matrix is defined by

$$X^{(0)} = \begin{pmatrix} x_0^{(0)} & x_1^{(0)} & x_2^{(0)} & x_3^{(0)} \\ x_4^{(0)} & x_5^{(0)} & x_6^{(0)} & x_7^{(0)} \\ x_8^{(0)} & x_9^{(0)} & x_{10}^{(0)} & x_{11}^{(0)} \\ x_{12}^{(0)} & x_{13}^{(0)} & x_{14}^{(0)} & x_{15}^{(0)} \end{pmatrix} = \begin{pmatrix} k_0 & k_1 & k_2 & k_3 \\ t_0 & t_1 & c_0 & c_1 \\ k_4 & k_5 & k_6 & k_7 \\ v_0 & v_1 & c_2 & c_3 \end{pmatrix}. \tag{32}$$

When comparing Eqs. (3) and (32), one can notice that Forró's initialization is different from ChaCha's. In differential cryptanalysis usually the attacker is allowed to choose arbitrary values to t_0, t_1, v_0 and v_1, thus it is a good idea to update these values as soon as possible allowing the differential to be propagated faster decreasing the probability of a differential characteristic. Thus, we defined the initialization in such a way that t_0, t_1, v_0 and v_1 are used in the first two columns, however, separated by the application of parts of the key.

Rotations The rotation distances for Forró are set as $r_1 = 10, r_2 = 27$ and $r_3 = 8$. Most authors of ARX algorithms in the literature do not justify the choice of the rotation distances with a numerical argument. It is generally argued that it is difficult to find bad rotation distances for ARX. Therefore, authors tend to choose aligned rotation distances (multiple of 8) because these are much faster than unaligned rotation distances on many non-64-bit architectures. For example, many 8-bit microcontrollers have only 1-bit shifts of bytes, so rotation by 3 bits is particularly expensive. Even 64-bit systems can benefit from alignment, for example, when a sequence of shift-shift-xor can be replaced by SSSE3's `pshufb` byte-shuffling instruction [1].

On the other hand, it may be possible to improve the security of the algorithm by carefully studying the behavior of the cipher when each combination of rotation distances is evaluated. This approach could allow for a reduced number of rounds to achieve the desired security. Hence, this approach could also improve performance. For example, in [12] authors showed that changing the rotation distances of ChaCha to $(19, 17, 25, 11)$ improved the resistance of ChaCha against known attacks.

Here, the rotation distances were defined following a similar approach as proposed in [12], with some adaptations. First, we define $\mathcal{R}$ as the set of all combinations of rotation distances (note that $|\mathcal{R}| = 32^3$). Next, we define Algorithm 1, which returns the maximum observed differential correlation among all single bit differentials $(\mathcal{ID}, \mathcal{OD})$ for a given combination of rotation distances $\mathbf{r} = (r_1, r_2, r_3) \in \mathcal{R}$ when considering N random trials. Then, to define the optimal rotation distances we executed the following steps:

1. Execute Algorithm 1 for all $\mathbf{r}_i \in \mathcal{R}$, obtaining a list $L = \{\delta_{\mathbf{r}_i}\}$.
2. Compute $\delta_{\min} = \min(L)$.
3. For each $\delta_{\mathbf{r}_i} \in L$, test the hypothesis $H_i : \delta_{\mathbf{r}_i} = \delta_{\min}$. More precisely, we used the standard statistical test to compare two proportions by converting the correlation to a probability $p_{\mathbf{r}_i} = (\delta_{\mathbf{r}_i} + 1)/2$. In addition, since we are dealing with multiple hypothesis tests, we used the Family-Wise Error Rate (FWER) technique to guard against type-I errors.
4. Discard all rotations distances $\mathbf{r}_i \in \mathcal{R}$ that lead to the hypothesis H_i being rejected. Thus, we are left with a subset of rotation distances $\mathcal{R}^* \subset \mathcal{R}$.
5. For each $\mathbf{r}_j \in \mathcal{R}^*$, compute the average neutrality measure $\bar{\gamma}_{\mathbf{r}_j}$ using Algorithm 1 of [2]. In this case, we considered an encryption with 5 rounds of Forró and 3 rounds executed backwards.
6. For each $\mathbf{r}_j \in \mathcal{R}^*$, define the metric $\mu_{\mathbf{r}_j} = \delta_{\mathbf{r}_j} \times \bar{\gamma}_{\mathbf{r}_j}$.

Algorithm 1 Returns the maximum observed differential correlation for all possible single bit differentials.

INPUT: rotation distances (r_1, r_2, r_3), the number of trials N.
Setup Forró with rotation distances (r_1, r_2, r_3).
for each single bit input difference $\mathcal{ID}$ **do**
    for $i \in \{1, 2, ..., N\}$ **do**
        Generate random key k, nonce v, and counter t.
        Initialize Forró's state matrix X.
        Execute 2 rounds of Forró from X, obtaining Y.
        Compute $X' = \mathcal{ID} \oplus X$.
        Execute 2 rounds of Forró from X', obtaining Y'.
        Compute $\mathcal{OD} = Y \oplus Y'$.
        Update the differential correlation $\delta_{\mathcal{ID},j}$ for each bit of $\mathcal{OD}$, where $j \in \{0, 1, ..., 512\}$.
return $\max(|\delta_{\mathcal{ID},j}|)$

7. Define the rotation distances for Forró as $\arg\min_{\mathbf{r}_j}\{\mu_{\mathbf{r}_j}\}$.

We executed these steps using a cluster of 24 *NVIDIA GPUs RTX 2080ti*. This setup allowed us to run Algorithm 1 with $N = 24 \times 2^{20}$, for all $\mathbf{r} \in \mathcal{R}$, in two days of computation. From these, we defined Forró's rotation distances as $(r_1, r_2, r_3) = (10, 27, 8)$. See the extended version of this paper for some interesting patterns that could be observed.

Constants Since the choice of the constants does not impact security or performance, we decided to go through a cultural route: the constants correspond to the ASCII string "*voltadaasabranca*", little-endian encoded. "*A volta da asa branca*" is the name of a song of the Brazilian singer Luiz Gonzaga. It is a continuation of the song "*asa branca*", one of the greatest classics of Brazilian music, composed more than 70 years ago. In "*asa branca*", Luiz Gonzaga and Humberto Teixeira tell us the story of a man who lost everything due to the drought in the Brazilian northeast region and had to leave his home in search of better living conditions. In "*a volta da asa branca*", he returns to his home and is reunited with his love with whom he intends to marry.

Number of rounds From Table 1, we know that we can attack a maximum of 7 rounds of ChaCha and 5 rounds of Forró. Therefore, we know that we do not need 20 rounds of Forró to achieve the security of ChaCha20 against known attacks. That said, it is not easy to quantify exactly how many rounds would give that security margin. Assuming that for every 7 rounds of ChaCha we can save 2 rounds in Forró, we recommend using Forró with a total of 14 rounds (Forro14) to achieve a security margin comparable with ChaCha20. Also, we recommend Forró with 10 rounds (Forro10) to achieve higher security than ChaCha12.

5.3 Security

In the extended version of this paper, we present a complete analysis of the security of Forró when considering the same techniques that are applied against ChaCha and Salsa. In this version, we only present the main results.

Distinguishers We constructed distinguishers for Forró by following the best techniques used against ChaCha in the literature [9,11]. More precisely, we looked for single bit differentials ranging 2 and 3 rounds of Forró. To do so, we tested all possible single bit input differences (128 possibilities) combined with every possible single bit output difference (512 possibilities). Hence, we tested a total of 2^{15} differentials. In each case, we estimated the correlation experimentally with a total of 2^{34} random samples. We present some examples in Table 3.

$\mathcal{ID}$	$\mathcal{OD}$	Correlation
$\Delta X_5^{(0)} = 2^{18}$	$\Delta X_{15}^{(2)} = 2^7$	-0.00379
$\Delta X_5^{(0)} = 2^{18}$	$\Delta X_{10}^{(2)} = 2^7$	-0.00221
$\Delta X_5^{(0)} = 2^{11}$	$\Delta X_{15}^{(2)} = 1$	-0.00139
$\Delta X_5^{(0)} = 2^{11}$	$\Delta X_{10}^{(2)} = 1$	-0.00053

Table 3: Some of the best single bit differentials for 2 rounds of Forró.

Next, using *CryptDances* we expanded the linear equations of Forró automatically. Using *CryptDances* functionalities, we also constructed distinguishers against Forró for every single differential that had a statistically significant correlation. From this study, we derived the best distinguisher for 3, 4, 5 and 5.25 rounds of Forró. We could not find any distinguishers against 5.5 rounds of Forró or more. In the following, we present more information about these distinguishers.

Distinguisher against 3 rounds of Forró. In this case, consider that single bit differential with $\mathcal{OD} = \Delta X_{15}^{(2)} = 1$ presented in Table 3. Thus, we have $\varepsilon_d = 0.00139$. For the linear part, we have to expand the bit $x_{15,0}^{(2)}$ (or, considering subrounds, $x_{15,0}^{[8]}$), obtaining $x_{15,0}^{[8]} = x_{15,27}^{[12]} \oplus x_{3,8}^{[12]} \oplus x_{7,0}^{[12]}$, with probability 1. Clearly, $\varepsilon_L = 1$, then the complexity of the differential-linear distinguisher for 3 rounds of Forró is $\frac{1}{\varepsilon_d^2} \approx 2^{18.9814}$.

Distinguisher against 4 rounds of Forró. In this case, consider that single bit differential with $\mathcal{OD} = \Delta X_{10}^{(2)} = 1$ presented in Table 3. Thus, we have $\varepsilon_d = 0.00053$. For the linear part, we have to expand the bit $x_{10,0}^{(2)} = x_{10,0}^{[8]}$, which results in the following Lemma:

Lemma 12. *The following linear approximation holds with probability* $\frac{1}{2}\left(1 + \frac{1}{2^5}\right)$

$$x_{10,0}^{[8]} = x_1^{[16]}[8] \oplus x_2^{[16]}[16] \oplus x_3^{[16]}[2,3,24] \oplus x_4^{[16]}[0,15,16,26,27] \oplus$$
$$x_7^{[16]}[7,8] \oplus x_9^{[16]}[0] \oplus x_{10}^{[16]}[0] \oplus x_{11}^{[16]}[0] \oplus x_{14}^{[16]}[22,27] \oplus x_{15}^{[16]}[0,27].$$

Proof. See the extended version of this paper. □

Computational Result 5 *The linear approximation of Lemma 12 holds computationally with* $\varepsilon_{L_0} = 0.0476 \approx 2^{-4.39}$. *This correlation was verified using* 2^{38} *random samples.*

We conclude that the complexity of the differential-linear distinguisher for 4 rounds of Forró is $\frac{1}{\varepsilon_d^2 \varepsilon_{L_0}^4} \approx 2^{36.55}$.

Distinguisher against 5 and 5.25 rounds of Forró. For these distinguishers, we just keep expanding the equation from Lemma 12. This will lead to differential-linear distinguishers with complexities $2^{129.68}$ and $2^{176.81}$ for 5 and 5.25 rounds of Forró, respectively. See the extended version of this paper for a complete description and proof of these distinguishers.

Attacks Using PNBs In this section, we use the techniques developed by [2] and later improved by [9] to attack Forró, see Section 2.4. We tested several different attacks for different values of γ for all differentials presented in Table 3. With this approach, the best attack we found against 5 rounds of Forró uses 2 rounds forward and 3 rounds backwards. The attack uses the differential $(\Delta_{10,0}^{(2)} | \Delta_{5,11}^{(0)})$, thus, from Table 3 we get $\varepsilon_d = -0.00053$. Using $\gamma = 0.25$ we get a total of 155 PNBs. From that, we estimated $\varepsilon_a = 0.000068$ which leads to an attack with data complexity of 2^{57} and time complexity of 2^{158}.

5.4 Performance

By design, Forró achieves the same security with less operations than ChaCha, the implication being that on embedded devices with limited concurrency capabilities, such as the Raspberry Pi and others used in IoT, Forró naturally has better performance, see Table 4 for measurements. However, in more advanced processors, where speculative execution and out-of-order execution are empowered by large caches, such as modern x86, ChaCha still has an advantage. It is possible, however, to work around this apparent limitation with a clever implementation.

In order to pipeline instructions, the processor detects (or speculates) instructions that don't have dependencies on each others output and are nearby to anticipate them, so while one executes, the other can be fetching, for example. In ChaCha, the QRF is applied independently inside a round, and pipelining occurs without much impediment. In Forró, because of Pollination, every operation in a round has a dependency on the previous output, causing a serial data dependency. Meaning that the processor can't detect independent instructions to pipeline, or if it guesses the instructions are likely to not retire.

However, just like ChaCha, in order to get the next 512 bits of keystream, the algorithm needs to be executed from the start with an increment on the counter. This execution is completely independent of the previous one. Unfortunately, the processor doesn't have the foresight to anticipate that, since the code for it is far

into the future, but that can be bypassed. To take full advantage of pipelining, whenever there is a need for more than 512 bits of keystream, we implement it so that the code for the two executions of Forró is in the same scope, a technique that for this specific use case we kindly named, "Xote". This strategy permits that Forró continues to leverage it's better diffusion to produce better performance on such processors, which can be seen in our measurements that are available on Table 4. For reference, the measurements also contain ChaCha with Xote. We make these implementations available at (`https://github.com/MurCoutinho/forro_cipher`).

	ARMv7	**ARMv7 NEON**	**ARMv8**	**Intel x86-64**	**Intel x86-64 SIMD**
Algorithm	**Cycles**	**Cycles**	**Cycles**	**Cycles**	**Cycles**
Salsa20	83689	-	24622	20542	4418
Chacha20	89495	51914	35100	20118	3934
Chacha20 (Xote)	138284	-	36214	19362	4480
Forro14	73230	49575	46700	34472	6244
Forro14 (Xote)	76236	-	31666	20748	4826

Table 4: Performance comparison generating a 4096 bytes keystream between Salsa, ChaCha and Forró on ARMv7, ARMv7 using NEON, ARMv8 (64 bits), Intel x86-64 and Intel x86-64 using SIMD (AVX2 and SSE as available).

6 CryptDances: a new tool for cryptanalysis of ARX ciphers

The final contribution of this work is a tool to perform cryptanalysis of ChaCha, Salsa and Forró in high performance environments. As a brief summary, in the current version of CryptDances we have:

- Implementation of most attacks from the literature for Salsa and ChaCha, in particular from [2,9,3,10,11] (attacks for [14] are not yet available).
- It is easy to test any new differential or linear approximation for Salsa, ChaCha or Forró.
- Automatic linear expansions for ChaCha and Forró (Salsa in development).
- Given a differential and linear expansion, CryptDances can compute the complexity of distinguishers and PNB attacks.

CryptDances is available at `https://github.com/MurCoutinho/cryptDances`.

7 Conclusion

In this work, we provided several contributions for ARX ciphers. In particular, we provided a new way to derive linear approximations for ChaCha, improving

the complexity of the best differential-linear distinguisher from 2^{224} to 2^{214}. In addition, using the proposed BLE, we improved attacks against Salsa. More precisely, we presented the first distinguishers against 7 and 8 rounds of Salsa with complexities 2^{109} and 2^{216}, and improved key recovery attacks achieving a complexity of 2^{212} for 8 rounds when the best know attack so far had complexity of $2^{244.9}$.

Another contribution of this work is a new stream cipher called Forró. We showed that Forró can achieve the same security as ChaCha with fewer operations. Because of that, Forró can achieve faster performance in certain platforms, specially in constrained devices. Finally, we developed *CryptDances*, a new tool for the cryptanalysis of Salsa, ChaCha, and Forró designed to be used in high performance environments with several GPUs, making it available for the community at `https://github.com/MurCoutinho/cryptDances`.

For future works, the techniques developed in this paper may be used to improve cryptanalysis against other ARX primitives, such as Chaskey or the hash function Blake. Also, the security of Forró should be analyzed further, specially against other types of attacks, such as rotational cryptanalysis. Finally, the tool *CryptDances* can be used by researchers to try to improve further attacks against Salsa, ChaCha, and Forró.

Acknowledgements. This work is supported in part by FAPDF - Brazilian Federal District Research Support Foundation, in part by CNPq - Brazilian National Research Council (Grants 312180/2019-5 PQ-2 and 465741/2014-2 INCT on Cybersecurity), in part by the Ministry of Justice and Public Security (Grant MJSP 01/2019), in part by the Administrative Council for Economic Defense (Grant CADE 08700.000047/2019-14), in part by the General Attorney of the Union (Grant AGU 697.935/2019), in part by the National Auditing Department of the Brazilian Health System (Grant DENASUS 23106.118410/2020-85), and in part by the General Attorney's Office for the National Treasure (Grant PGFN 23106.148934/2019-67).

References

1. Aumasson, J., Bernstein, D.J.: Siphash: A fast short-input PRF. In: Galbraith, S.D., Nandi, M. (eds.) Progress in Cryptology - INDOCRYPT 2012, 13th International Conference on Cryptology in India. Lecture Notes in Computer Science, vol. 7668, pp. 489–508. Springer (2012). https://doi.org/10.1007/978-3-642-34931-7_28, `https://doi.org/10.1007/978-3-642-34931-7_28`
2. Aumasson, J., Fischer, S., Khazaei, S., Meier, W., Rechberger, C.: New features of latin dances: Analysis of salsa, chacha, and rumba. In: Nyberg, K. (ed.) Fast Software Encryption, 15th International Workshop, FSE 2008, Lausanne, Switzerland, February 10-13, 2008, Revised Selected Papers. Lecture Notes in Computer Science, vol. 5086, pp. 470–488. Springer (2008). https://doi.org/10.1007/978-3-540-71039-4_30, `https://doi.org/10.1007/978-3-540-71039-4_30`
3. Beierle, C., Leander, G., Todo, Y.: Improved differential-linear attacks with applications to ARX ciphers. In: Micciancio, D., Ristenpart, T. (eds.) Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020,

Proceedings, Part III. Lecture Notes in Computer Science, vol. 12172, pp. 329–358. Springer (2020). https://doi.org/10.1007/978-3-030-56877-1_12, https://doi.org/10.1007/978-3-030-56877-1_12
4. Bernstein, D.J.: The poly1305-aes message-authentication code. In: Gilbert, H., Handschuh, H. (eds.) Fast Software Encryption: 12th International Workshop, FSE 2005, Paris, France, February 21-23, 2005, Revised Selected Papers. Lecture Notes in Computer Science, vol. 3557, pp. 32–49. Springer (2005). https://doi.org/10.1007/11502760_3, https://doi.org/10.1007/11502760_3
5. Bernstein, D.J.: Chacha, a variant of salsa20. In: Workshop Record of SASC. vol. 8, pp. 3–5 (2008)
6. Bernstein, D.J.: The salsa20 family of stream ciphers. In: Robshaw, M.J.B., Billet, O. (eds.) New Stream Cipher Designs - The eSTREAM Finalists, Lecture Notes in Computer Science, vol. 4986, pp. 84–97. Springer (2008). https://doi.org/10.1007/978-3-540-68351-3_8, https://doi.org/10.1007/978-3-540-68351-3_8
7. Blondeau, C., Leander, G., Nyberg, K.: Differential-linear cryptanalysis revisited. J. Cryptol. **30**(3), 859–888 (2017). https://doi.org/10.1007/s00145-016-9237-5, https://doi.org/10.1007/s00145-016-9237-5
8. Castro, J.C.H., Estévez-Tapiador, J.M., Quisquater, J.: On the salsa20 core function. In: Nyberg, K. (ed.) Fast Software Encryption, 15th International Workshop, FSE 2008, Lausanne, Switzerland, February 10-13, 2008, Revised Selected Papers. Lecture Notes in Computer Science, vol. 5086, pp. 462–469. Springer (2008). https://doi.org/10.1007/978-3-540-71039-4_29, https://doi.org/10.1007/978-3-540-71039-4_29
9. Choudhuri, A.R., Maitra, S.: Significantly improved multi-bit differentials for reduced round salsa and chacha. IACR Trans. Symmetric Cryptol. **2016**(2), 261–287 (2016). https://doi.org/10.13154/tosc.v2016.i2.261-287, https://doi.org/10.13154/tosc.v2016.i2.261-287
10. Coutinho, M., Neto, T.C.S.: New multi-bit differentials to improve attacks against chacha. IACR Cryptol. ePrint Arch. **2020**, 350 (2020), https://eprint.iacr.org/2020/350
11. Coutinho, M., Neto, T.C.S.: Improved linear approximations to ARX ciphers and attacks against chacha. In: Canteaut, A., Standaert, F. (eds.) Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part I. Lecture Notes in Computer Science, vol. 12696, pp. 711–740. Springer (2021). https://doi.org/10.1007/978-3-030-77870-5_25, https://doi.org/10.1007/978-3-030-77870-5_25
12. Coutinho, M., Passos, I., de Sousa Jr, R.T., Borges, F.: Improving the security of chacha against differential-linear cryptanalysis (2020)
13. Crowley, P.: Truncated differential cryptanalysis of five rounds of salsa20. IACR Cryptol. ePrint Arch. **2005**, 375 (2005), http://eprint.iacr.org/2005/375
14. Dey, S., Garai, H.K., Sarkar, S., Sharma, N.K.: Revamped differential-linear cryptanalysis on reduced round chacha. Springer-Verlag (2022)
15. Dey, S., Sarkar, S.: Improved analysis for reduced round salsa and chacha. Discret. Appl. Math. **227**, 58–69 (2017). https://doi.org/10.1016/j.dam.2017.04.034, https://doi.org/10.1016/j.dam.2017.04.034
16. Ding, L.: Improved related-cipher attack on salsa20 stream cipher. IEEE Access **7**, 30197–30202 (2019). https://doi.org/10.1109/ACCESS.2019.2892647, https://doi.org/10.1109/ACCESS.2019.2892647

17. Fischer, S., Meier, W., Berbain, C., Biasse, J., Robshaw, M.J.B.: Non-randomness in estream candidates salsa20 and TSC-4. In: Barua, R., Lange, T. (eds.) Progress in Cryptology - INDOCRYPT 2006, 7th International Conference on Cryptology in India, Kolkata, India, December 11-13, 2006, Proceedings. Lecture Notes in Computer Science, vol. 4329, pp. 2–16. Springer (2006). https://doi.org/10.1007/11941378_2, https://doi.org/10.1007/11941378_2
18. IANIX: ChaCha usage & deployment. https://ianix.com/pub/chacha-deployment.html (2020), accessed: 2020-01-13
19. IANIX: Salsa20 usage & deployment. https://ianix.com/pub/salsa20-deployment.html (2021), accessed: 2021-02-02
20. Langford, S.K., Hellman, M.E.: Differential-linear cryptanalysis. In: Desmedt, Y. (ed.) Advances in Cryptology - CRYPTO '94, Proceedings. Lecture Notes in Computer Science, vol. 839, pp. 17–25. Springer (1994). https://doi.org/10.1007/3-540-48658-5_3, https://doi.org/10.1007/3-540-48658-5_3
21. Langley, A., Chang, W., Mavrogiannopoulos, N., Strömbergson, J., Josefsson, S.: Chacha20-poly1305 cipher suites for transport layer security (TLS). RFC **7905**, 1–8 (2016). https://doi.org/10.17487/RFC7905, https://doi.org/10.17487/RFC7905
22. Lipmaa, H., Moriai, S.: Efficient algorithms for computing differential properties of addition. In: Matsui, M. (ed.) Fast Software Encryption, 8th International Workshop, FSE 2001 Yokohama, Japan, April 2-4, 2001, Revised Papers. Lecture Notes in Computer Science, vol. 2355, pp. 336–350. Springer (2001). https://doi.org/10.1007/3-540-45473-X_28, https://doi.org/10.1007/3-540-45473-X_28
23. Maitra, S.: Chosen IV cryptanalysis on reduced round chacha and salsa. Discret. Appl. Math. **208**, 88–97 (2016). https://doi.org/10.1016/j.dam.2016.02.020, https://doi.org/10.1016/j.dam.2016.02.020
24. Maitra, S., Paul, G., Meier, W.: Salsa20 cryptanalysis: New moves and revisiting old styles. IACR Cryptol. ePrint Arch. **2015**, 217 (2015), http://eprint.iacr.org/2015/217
25. Mouha, N., Preneel, B.: A proof that the ARX cipher salsa20 is secure against differential cryptanalysis. IACR Cryptol. ePrint Arch. **2013**, 328 (2013), http://eprint.iacr.org/2013/328
26. Niu, Z., Sun, S., Liu, Y., Li, C.: Rotational differential-linear distinguishers of arx ciphers with arbitrary output linear masks. Cryptology ePrint Archive (2022)
27. Robshaw, M.J.B., Billet, O. (eds.): New Stream Cipher Designs - The eSTREAM Finalists, Lecture Notes in Computer Science, vol. 4986. Springer (2008). https://doi.org/10.1007/978-3-540-68351-3, https://doi.org/10.1007/978-3-540-68351-3
28. Shi, Z., Zhang, B., Feng, D., Wu, W.: Improved key recovery attacks on reduced-round salsa20 and chacha. In: Kwon, T., Lee, M., Kwon, D. (eds.) Information Security and Cryptology - ICISC 2012. Lecture Notes in Computer Science, vol. 7839, pp. 337–351. Springer (2012). https://doi.org/10.1007/978-3-642-37682-5_24, https://doi.org/10.1007/978-3-642-37682-5_24
29. Wallén, J.: Linear approximations of addition modulo 2^n. In: Johansson, T. (ed.) Fast Software Encryption, 10th International Workshop, FSE 2003, Lund, Sweden, February 24-26, 2003, Revised Papers. Lecture Notes in Computer Science, vol. 2887, pp. 261–273. Springer (2003). https://doi.org/10.1007/978-3-540-39887-5_20, https://doi.org/10.1007/978-3-540-39887-5_20