

Enhanced Differential-Linear Cryptanalysis of ChaCha Based on Bit Puncturing

Xinhai Wang, Lin Ding, Zhengting Li, Honglei Wang, Jiang Wan and Bin Hu

Abstract—ChaCha is one of the most extensively deployed symmetric ciphers. The security margin of ChaCha is directly related to the safety of many widely used lightweight security protocols and operating systems for constrained devices, such as TLS 1.3, SSH, Noise, WireGuard, S/MIME 4.0, Linux, Android, Chromium/Chrome, Firefox and Safari. This paper introduces a novel *guessed key covering technique*. The objective of this technique is to identify partitioning-based functions whose required key bits are covered by those guessed for bit puncturing-based functions, enabling them to be processed jointly. Building on this, we propose a refined framework for differential-linear cryptanalysis of ChaCha called **ReBitP**, which combines the ideas of the bit puncturing technique, the partitioning technique and a two-phase distillation strategy. The key insight of **ReBitP** is to incorporate a carefully selected set of functions that are evaluated using the partitioning technique with different tail lengths into the first phase, without requiring additional key bits to be guessed. This early filtering via parity checks simultaneously lowers the time cost of the first phase and the time complexity of constructing the distillation table in the second phase. As applications, enhanced key recovery attacks on 7- and 7.5-round ChaCha256 are presented, achieving time complexities of $2^{142.08}$ and $2^{242.02}$, respectively. The cryptanalytic results are $2^{6.12}$ and $2^{1.58}$ times faster than the existing attacks, respectively. So far as we know, these are the best known key recovery attacks on 7- and 7.5-round ChaCha256. This definitely demonstrates the superiority of the refined framework **ReBitP**.

Index Terms—Stream cipher, ChaCha, Differential-linear attack, Bit puncturing

I. INTRODUCTION

SYMMETRIC key cryptosystems play an important role in secure communication. There are many ways to design symmetric key cryptosystems, and the ARX-based design is an important one due to its good security properties and efficiency in software. Here, ARX is an acronym for modular Addition, Rotation and XOR. There are a lot of famous cryptographic ciphers that utilize the ARX-based design approach, such as the stream ciphers Salsa [1] and ChaCha [2], the block cipher Sparx [3], the hash function BLAKE2 [4], the cryptographic permutation Sparkle [5] and the MAC Chaskey [6].

Salsa [1], designed in 2005, is a well-known ARX-based stream cipher, and was submitted as a competitor to the eSTREAM competition [7], the ECRYPT Stream Cipher Project. The original cipher consists of 20 rounds in total, and the 12-round version of Salsa, denoted as Salsa20/12, was selected

into the final portfolio of eSTREAM in 2008. In the same year, Bernstein [2] made some modifications to the round function of Salsa and proposed a new variant called ChaCha to enhance diffusion and resistance against cryptanalysis without compromising the encryption speed. Like Salsa, ChaCha also comprises 20 rounds in total and supports two key sizes, i.e., 256 bits (ChaCha256) and 128 bits (ChaCha128). This paper concentrates primarily on ChaCha256, due to its extensive real-world deployment.

Despite Salsa being one of the seven ciphers selected as an eSTREAM winner, ChaCha has received significantly greater attention recently because of its widespread practical adoption. In particular, the ChaCha stream cipher paired with the Poly1305 message authentication code [8] forms part of the cipher suites standardized in the TLS 1.3 [9]. Google has integrated this combination into both Chrome and Android. Moreover, ChaCha is employed in numerous important lightweight security protocols, such as SSH, Noise, WireGuard and S/MIME 4.0, and used in the random number generation of many operating systems, such as Linux, Android, OpenBSD and Dragonfly BSD. It is also proposed to be used in IKEv2 and IPsec by RFC 7634 [10]. Now, ChaCha is being utilized in a vast array of lightweight security protocols, operating systems, cryptographic libraries, and applications (a comprehensive list can be found in [11]). Consequently, the ChaCha stream cipher has emerged as one of the most widely deployed symmetric ciphers nowadays. The security margin of ChaCha is directly related to the safety of these widely used lightweight security protocols, operating systems and softwares.

Related Works. Since ChaCha was proposed in 2008, it has attracted great interest from many cryptanalysts in recent years, and several cryptanalytic results of the cipher had been published. At FSE 2008, Aumasson et al. [12] proposed the *Probabilistic Neutral Bits* (PNBs) technique and mounted the first PNB-based key recovery attacks on the round-reduced version of ChaCha256. In 2012, Shi et al. [13] followed the idea of PNB and presented improved key recovery attacks on 6- and 7-round ChaCha256 by introducing a new class of distinguishers. Maitra [14] later advanced these efforts by leveraging carefully chosen initialization vectors (IVs) tied to the secret key, thereby achieving a more efficient attack on 7-round ChaCha256. Subsequently, Choudhuri and Maitra [15] explored multi-bit differential properties and proposed several enhanced key recovery attacks. In 2017, Dey and Sarkar [16] reported an improvement for 7-round ChaCha256 with time complexity $2^{235.22}$. In 2020, the attacks on 6- and

Manuscript received February 8, 2026; revised April 9, 2026; accepted June 10, 2026. (Corresponding author: Lin Ding.)

Xinhai Wang, Lin Ding, Zhengting Li, Honglei Wang, Jiang Wan and Bin Hu are with the Information Engineering University, Zhengzhou 450001, China (E-mails: wxh1066559569@126.com, dinglin_cipher@163.com, lizhengting0225@163.com, WangHonglei_1007@163.com, wanjiang0626@163.com, hb2110@126.com).

TABLE I: A summary of the key recovery attacks on round-reduced version of ChaCha256

Rounds	Time complexity	Data complexity	Memory complexity	Ref.
6	2^{139} (corrected: 2^{147})	2^{30}	–	[12]
	2^{136} (corrected: 2^{139})	2^{28}	–	[13]
	$2^{131.40}$ (corrected: 2^{159})	$2^{34.39}$	–	[15]
	$2^{129.53}$ (corrected: 2^{161})	$2^{34.51}$	–	[15]
	$2^{127.5}$ (corrected: 2^{166})	$2^{37.5}$	–	[15]
	$2^{102.2}$ (corrected: 2^{210})	2^{56}	–	[17]
	$2^{77.4}$ (partial key recovery)**	2^{58}	–	[18]
	$2^{99.48}$	$2^{41.47}$	–	[22]
	$2^{75.7}$ (partial key recovery)***	$2^{73.7}$	–	[24]
	$2^{57.4}$	$2^{55.7}$	–	[29]
7	2^{248}	2^{27}	–	[12]
	$2^{246.5}$	2^{27}	–	[13]
	$2^{238.94}$	2^{96}	2^{96}	[14]
	$2^{237.65}$	2^{96}	2^{96}	[15]
	$2^{235.22}$	–	–	[16]
	$2^{231.9}$	2^{50}	–	[17]
	$2^{231.63}$	$2^{49.58}$	–	[19]
	$2^{230.86}$	$2^{48.83}$	–	[18]
	$2^{228.51}$	$2^{80.51}$	–	[20]
	$2^{221.95}$	$2^{90.20\dagger}$	$2^{47.31}$	[21]
	$2^{216.9}$	$2^{68.9}$	–	[24]
	$2^{210.3}$	$2^{103.3}$	–	[24]
	$2^{206.8}$	$2^{110.81}$	–	[25]
	$2^{192.89}$	$2^{93.79}$	2^{68}	[27]
	$2^{189.7}$	$2^{102.63}$	–	[26]
	$2^{178.12}$	$2^{101.09}$	–	[28]
	$2^{154.2}$	$2^{102.9}$	–	[29]
	$2^{152.3}$	$2^{100.91}$	–	This paper
	$2^{148.2}$	$2^{127.7}$	–	[29]
	$2^{142.08}$	$2^{127.99}$	–	This paper
7.25 (or $7.5^{\oplus}$) [‡]	$2^{255.62}$	$2^{48.36}$	–	[19]
	$2^{244.9}$	$2^{104.9}$	–	[24]
	$2^{242.9}$	$2^{125.8}$	–	[24]
	$2^{238.34}$	$2^{122.34}$	–	[25]
	$2^{228.24}$	$2^{100.9}$	2^{91}	[27]
	$2^{223.9}$	$2^{100.8}$	–	[26]
	$2^{212.43}$	$2^{100.56}$	–	[28]
7.5	$2^{255.24}$	$2^{32.64}$	2^{217}	[27]
	$2^{253.23}$	$2^{34.47}$	–	[28]
	$2^{250.2}$	$2^{127.1}$	–	[29]
	$2^{245.76}$	$2^{127.99}$	–	This paper
	$2^{243.6}$	$2^{127.1}$	–	[31]
	$2^{242.02}$	$2^{125.04}$	–	This paper

* At FSE 2023, Dey et al. [22] corrected the time complexities.

** The attack could recover only 36 (out of 256) key bits.

*** The attack could recover only 45 (out of 256) key bits.

† The data complexity can be cut down to $2^{75.89}$ according to the evaluation in [24].

‡ It was proved in [26] that 7.25-round ChaCha and ChaCha7.5[⊕] provide the same security.

7-round ChaCha256 were further improved by Coutinho and Souza [17].

At CRYPTO 2020, the *right pair* technique was introduced by Beierle, Leander and Todo [18] to amplify the correlation of differential-linear (DL) distinguishers and presented new attacks on ChaCha256. Later, Miyashita et al. [19] proposed a PNB-focused differential attack on ChaCha and gave attacks on 7- and 7.25-round versions of ChaCha256. Coutinho and Souza [20] devised a novel method for identifying better linear approximations in ARX-based ciphers, which they applied to improve the attack on 7-round version of ChaCha256 at EUROCRYPT 2021. The following year, Dey et al. [21] enhanced the existing attacks on 7-round version of ChaCha256 by partitioning all key bits into two subsets, i.e., memory-dependent and memory-independent. Notably, they also proposed a more

precise formula for complexity estimation, refining the original model of Aumasson et al. [12].

At FSE 2023, the time complexities of the previous attacks on ChaCha256 were theoretically revisited by Dey et al. [22]. As a result, the time complexities of several previous attacks on 6-round ChaCha256 were corrected, and an improved on 6-round ChaCha256 with time/data $2^{99.48}/2^{41.47}$ was proposed. Soon after, new attacks on 7- and 7.25-round ChaCha256 were proposed by Dey et al. in [23]. At CRYPTO 2023, Wang et al. [24] proposed the *syncopation* technique to refine the PNB-based cryptanalysis attacks. Using this approach, they presented new attacks on 6- and 7-round ChaCha256, and gave an attack on ChaCha7.5[⊕] that is a 7.5-round version of ChaCha lacking the final XOR and rotation operations. At the same time, Bellini et al. [25] gave improved attacks on 7-

and 7.25-round ChaCha256 with the automated MILP tool.

More recently, Xu et al. [26] incorporated the differential-linear hull effect into their cryptanalysis and presented enhanced attacks on ChaCha256. They also proved that 7.25-round ChaCha and ChaCha7.5[⊕] offer equivalent security under the chosen (or known) plaintext setting. In [27], Dey introduced a new framework for the PNB-based attack and gave the first key recovery attack on 7.5-round ChaCha256. Recently, a follow-up work of [27] was published in [28].

At EUROCRYPT 2025, Flórez-Gutiérrez and Todo [29] introduced a new technique called *bit puncturing* to replace the PNB technique, and obtained significantly improved attacks on ChaCha256. Specifically, key recovery attacks on 7- and 7.5-round ChaCha256 with time $2^{148.2}$ and $2^{250.2}$ were obtained, with data $2^{127.7}$ and $2^{127.1}$, respectively. The proposed technique *bit puncturing* is a specific variant of *Walsh spectrum puncturing* [30] proposed at EUROCRYPT 2024. Very recently, Flórez-Gutiérrez and Todo [31] proposed a divide-and-conquer technique for trail enumeration puncturing, improving the key recovery attack on 7.5-round ChaCha256. However, our work is independent of theirs.

Our Contributions. The security margin of ChaCha is directly related to the safety of many widely used protocols, operating systems and softwares, such as TLS, SSH, Noise, WireGuard, Linux, Android, Chromium/Chrome, Firefox and Safari, this paper focuses on analyzing the security of ChaCha against differential-linear cryptanalysis, to obtain a deeper understanding of the security margin of ChaCha. The contributions are summarized below.

- *Technical Contributions.* This paper introduces a novel *guessed key covering technique*. The objective of this technique is to identify partitioning-based functions whose required key bits are covered by those guessed for bit puncturing-based functions, enabling them to be processed jointly. Building on this, we propose a refined framework for differential-linear cryptanalysis of ChaCha called ReBitP, which integrates the bit puncturing technique, the partitioning technique, and a two-phase distillation strategy. The key insight of ReBitP is to incorporate a carefully selected set of functions that are evaluated using the partitioning technique with different tail lengths into the first phase, without requiring additional key bits to be guessed. This early filtering via parity checks simultaneously lowers the time cost of the first phase and the time complexity of constructing the distillation table in the second phase. Overall, ReBitP combines the ideas of the bit puncturing technique, the partitioning technique, a two-phase distillation strategy and the guessed key covering technique introduced in this paper. Thus, ReBitP is a refined framework of the existing approach. Furthermore, this paper formally analyzes the trade-off relationship of the bit puncturing-based attack between the time complexity, the data complexity and the success probability, which is not addressed in the previous work. Undoubtedly, this will give a better understanding of the existing approach, and also offer the attacker a wider choice of the cryptanalytic results.

- *Applications to ChaCha256.* By applying ReBitP to ChaCha256, a key recovery attack on 7-round ChaCha256 with a time complexity of $2^{142.08}$ is presented, which improves the existing attack proposed at EUROCRYPT 2025 by a factor of $2^{6.12}$. As for 7.5-round ChaCha256, a key recovery attack with a time complexity of $2^{242.02}$ is obtained, which improves the existing attack proposed at ASIACRYPT 2025 by a factor of $2^{1.58}$. As shown in Table I, these are the best known key recovery attacks on 7- and 7.5-round ChaCha256 up to now. This definitely demonstrates the superiority of the refined framework ReBitP.

Highlight. At EUROCRYPT 2025 [29], the authors also considered incorporating functions utilizing the partitioning technique into the first phase. Since all functions using the bit puncturing technique must be evaluated in the first phase, the corresponding key bits must be guessed. However, their selection of functions utilizing the partitioning technique increases the number of key bits required to be guessed in the first phase. To achieve a time-data trade-off (e.g., in their key recovery attack on 6-round ChaCha256), their approach relied on using partitioning with smaller tail lengths in the first phase, which reduced the number of guessed key bits and consequently lowered the time complexity. However, for key recovery attacks on 7 rounds and beyond, the punctured functions are already sufficiently complex, which prevents us from arbitrarily incorporating functions using the partitioning technique into the first phase. Consequently, their 7-round and 7.5-round key recovery attacks fail to achieve an effective time-data trade-off.

To address this issue, we introduce a novel *guessed key covering technique*. The objective of this technique is to identify partitioning-based functions whose required key bits are covered by those guessed for bit puncturing-based functions, enabling them to be processed jointly. Building on this, we propose a refined framework for differential-linear cryptanalysis of ChaCha called ReBitP, which integrates the bit puncturing technique, the partitioning technique, and a two-phase distillation strategy. By applying our refined framework ReBitP, we can carefully select specific functions to be evaluated using the partitioning technique in the first phase, without requiring additional key bits to be guessed. Consequently, while maintaining the number of key bits guessed in the first phase, we reduce the time complexity by decreasing the number of functions requiring the partitioning technique in the second phase.

Furthermore, building upon the time-data trade-off, we incorporate a success probability trade-off, which is not addressed in the previous work. Due to data complexity constraints in ChaCha, certain functions that could otherwise be evaluated using the partitioning technique in the first phase are forced to be evaluated in the second phase. Benefiting from our framework ReBitP, we can free up space by adjusting the value of N' . Since N' is correlated with the success probability, we can achieve optimal results by extending the conventional time-data trade-off to a time-data-success probability trade-off.

In summary, our framework ReBitP differs significantly from previous works, representing a refined framework for differential-linear cryptanalysis of ChaCha based on bit puncturing.

Organization. The organization of this paper is outlined as follows. Section II offers a brief overview of the ChaCha256 stream cipher and introduces the cryptanalytic methods and techniques employed in this paper. The refined framework ReBitP is introduced in Section III. By applying ReBitP to ChaCha256, Sections IV and V propose enhanced key recovery attacks on ChaCha256. Section VI concludes this paper.

II. PRELIMINARIES

This section will briefly describe the ChaCha256 stream cipher and introduce the cryptanalytic methods and techniques employed in this paper.

A. A Brief Description of the ChaCha256 Stream Cipher

ChaCha utilizes an ARX-based design approach and supports two versions, i.e., ChaCha256 and ChaCha128, which offer 256- and 128-bit security levels respectively. This paper focuses on ChaCha256 because of its widely application. For a complete description of ChaCha, we refer to [2]. The internal state of ChaCha256 consists of 512 bits in total, and is arranged as a 4×4 word matrix where each word consists of 32 bits. Let $k = (k_0, k_1, \dots, k_7)$ be the 256-bit key of ChaCha256, and $X^{(r)} = (x_0^{(r)}, \dots, x_{15}^{(r)})$ denote the state matrix of ChaCha256 at round r . The ChaCha256 stream cipher is initialized as follows:

$$X^{(0)} = \begin{pmatrix} x_0^{(0)} & x_1^{(0)} & x_2^{(0)} & x_3^{(0)} \\ x_4^{(0)} & x_5^{(0)} & x_6^{(0)} & x_7^{(0)} \\ x_8^{(0)} & x_9^{(0)} & x_{10}^{(0)} & x_{11}^{(0)} \\ x_{12}^{(0)} & x_{13}^{(0)} & x_{14}^{(0)} & x_{15}^{(0)} \end{pmatrix} = \begin{pmatrix} c_0 & c_1 & c_2 & c_3 \\ k_0 & k_1 & k_2 & k_3 \\ k_4 & k_5 & k_6 & k_7 \\ t_0 & v_0 & v_1 & v_2 \end{pmatrix},$$

where t_0 is a 32-bit counter, $v = (v_0, v_1, v_2)$ is a 96-bit nonce, and $c_0 = 0 \times 61707865$, $c_1 = 0 \times 3320646e$, $c_2 = 0 \times 79622d32$ and $c_3 = 0 \times 6b206574$ are four 32-bit constants. In this paper, t_0 and $v = (v_0, v_1, v_2)$ are referred to as IV words. Let $IV = (iv_0, \dots, iv_{127})$ denote the 128-bit IV of ChaCha, and the IV loading can be defined as $x_{12+i,j}^{(0)} = iv_{32i+j}$, $i = 0, \dots, 3$, $j = 0, \dots, 31$.

The state matrix of the ChaCha256 stream cipher is updated using the round function *Round* that is made up of four parallel *quarterround* function. Given a 4-word input vector $(x_a^{(r-1)}, x_b^{(r-1)}, x_c^{(r-1)}, x_d^{(r-1)})$, the *quarterround* function generates a 4-word vector $(x_a^{(r)}, x_b^{(r)}, x_c^{(r)}, x_d^{(r)})$ as output via an intermediate vector $(x_a^{(r-0.5)}, x_b^{(r-0.5)}, x_c^{(r-0.5)}, x_d^{(r-0.5)})$:

$$\begin{aligned} x_a^{(r-0.5)} &= x_a^{(r-1)} + x_b^{(r-1)}; \\ x_d^{(r-0.5)} &= (x_d^{(r-1)} \oplus x_a^{(r-0.5)}) \lll 16; \\ x_c^{(r-0.5)} &= x_c^{(r-1)} + x_d^{(r-0.5)}; \\ x_b^{(r-0.5)} &= (x_b^{(r-1)} \oplus x_c^{(r-0.5)}) \lll 12; \\ x_a^{(r)} &= x_a^{(r-0.5)} + x_b^{(r-0.5)}; \\ x_d^{(r)} &= (x_d^{(r-0.5)} \oplus x_a^{(r)}) \lll 8; \\ x_c^{(r)} &= x_c^{(r-0.5)} + x_d^{(r)}; \\ x_b^{(r)} &= (x_b^{(r-0.5)} \oplus x_c^{(r)}) \lll 7. \end{aligned}$$

For odd rounds, the *quarterround* function is applied to four columns (x_0, x_4, x_8, x_{12}) , (x_1, x_5, x_9, x_{13}) , $(x_2, x_6, x_{10}, x_{14})$ and $(x_3, x_7, x_{11}, x_{15})$. For even rounds, the *quarterround* function is applied to four diagonals $(x_0, x_5, x_{10}, x_{15})$, $(x_1, x_6, x_{11}, x_{12})$, (x_2, x_7, x_8, x_{13}) and (x_3, x_4, x_9, x_{14}) .

In the keystream generation, the r -round ChaCha256 can generate the 512-bit keystream block Z as $Z = X^{(0)} + X^{(r)}$. Here, $X^{(r)}$ denotes the state matrix after r rounds, i.e., $X^{(r)} = \text{Round}^r(X^{(0)})$. Note that the function *Round* of ChaCha256 is invertible, i.e., $X^{(0)} = \text{Round}^{-r}(X^{(r)})$, where Round^{-1} is the inverse of *Round*. For the convenience of description, we define a variant of the 512-bit keystream block Z as $\bar{Z}$, where

$$\bar{z}_i = \begin{cases} x_i^{(r)}, & \text{if } i \in \{0, 1, 2, 3, 12, 13, 14, 15\}, \\ z_i, & \text{otherwise.} \end{cases}$$

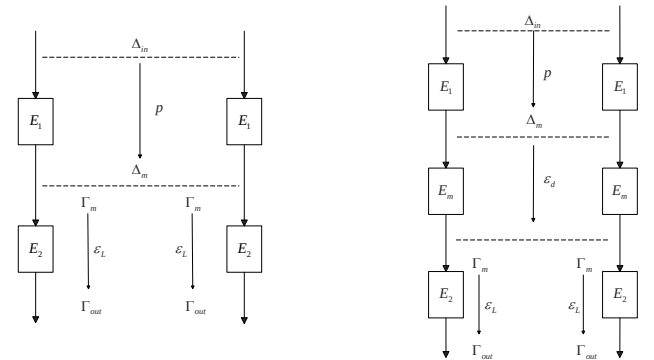


Fig. 1: A differential-linear distinguisher with two sub-ciphers (left) and a differential-linear distinguisher with three sub-ciphers (right).

B. Differential-Linear (DL) Cryptanalysis

Differential-linear attack is a powerful and general cryptanalytic technique proposed by Langford and Hellman [33], which is a combination of differential cryptanalysis [34] and linear cryptanalysis [35]. Given an iterative symmetric cipher E , the approach divides E into two consecutive sub-ciphers E_1 and E_2 satisfying $E = E_2 \circ E_1$. In this attack, a differential distinguisher is applied to the first part E_1 , while a linear distinguisher is used for the second part E_2 , as shown in the left side of Fig. 1. Let Δ_{in} denote the input differential (ID),

where X and $X' = X \oplus \Delta_{\text{in}}$ are two input states. Assume that for the sub-cipher E_1 , the differential trail $\Delta_{\text{in}} \rightarrow \Delta_m$ holds with probability

$$p = \Pr_{x \in \mathbb{F}_2^n} [E_1(x) \oplus E_1(x \oplus \Delta_{\text{in}}) = \Delta_m],$$

where Δ_m denotes the output differential ($\mathcal{OD}$) of E_1 . For the linear part, assume the linear approximation $\langle \Gamma_m, x \rangle \oplus \langle \Gamma_{\text{out}}, E_2(x) \rangle$ has correlation

$$\varepsilon_L = \mathbf{Cor}_{x \in \mathbb{F}_2^n} [\langle \Gamma_m, x \rangle \oplus \langle \Gamma_{\text{out}}, E_2(x) \rangle],$$

where Γ_m and Γ_{out} are the input and output linear masks for E_2 , respectively. By merging these two parts, a differential-linear distinguisher $\Delta_{\text{in}} \rightarrow \Gamma_{\text{out}}$ is constructed, whose correlation is approximated as $p\varepsilon_L^2$, under the heuristic assumption that the outputs of E_1 and the inputs to E_2 behave as independent random variables. Thus, the cipher E could be successfully distinguished from a random permutation with complexity $\mathcal{O}(p^{-2}\varepsilon_L^{-4})$.

However, the independence assumption may lead to an inaccurate estimation of the actual correlation. To mitigate this problem, a common solution employed in many prior works as well as in our attacks is to split the cipher into three parts as $E = E_2 \circ E_m \circ E_1$, as illustrated in the right side of Fig. 1. Now, the distinguisher has a correlation of $p\varepsilon_d\varepsilon_L^2$, where $\varepsilon_d = \mathbf{Cor}_{\mathcal{S}} [\langle \Gamma_m, E_m(x) \rangle \oplus \langle \Gamma_m, E_m(x \oplus \Delta_m) \rangle]$ is usually evaluated experimentally over a sample set $\mathcal{S}$. Therefore, the cipher E could be successfully distinguished from a random permutation with complexity $\mathcal{O}(p^{-2}\varepsilon_d^{-2}\varepsilon_L^{-4})$.

C. The Right Pair Technique

To enhance the cryptanalytic effect of a differential-linear distinguisher, Beierle, Leander and Todo [18] introduced the right pair technique for ARX ciphers. Formally, two initial states $(X, X \oplus \Delta_{\text{in}})$ are defined as a *right pair* if they satisfy the desired output differential Δ_m after E_1 . After then, the probability that any two initial states $(X, X \oplus \Delta_{\text{in}})$ form a right pair can be described as $p = \Pr_{x \in \mathbb{F}_2^n} [E_1(x) \oplus E_1(x \oplus \Delta_{\text{in}}) = \Delta_m]$. When the right pair technique is used, the attacker assumes that once a right pair is found, many other right pairs can be generated for free. This assumption holds true for ChaCha, as the column containing the input differential is independent of the remaining three columns. Specifically, once a right pair is identified, additional right pairs can be generated by varying the IV words of the other three columns. Since only one right pair is expected among p^{-1} random pairs, the actual attack must be repeated about p^{-1} times on average to obtain a right pair, and the remaining right pairs are freely available. Consequently, the time/data complexity should be multiplied by p^{-1} to get the final time/data complexity.

D. The Partitioning Technique

Partitioning is a new cryptanalytic technique introduced by Beierle, Leander and Todo [18] at CRYPTO 2020, which exploits structural properties of modular addition. The following theorem can be used to handle a d -bit tail.

Theorem 2 [29]. Let $z = a \boxplus b$, $s = a \oplus b$ and $\text{tr}_i(s)$ denote the i rightmost bits of s , where $a, b \in \mathbb{F}_2^n$. For $i > d$, we have

$$z[i] = \begin{cases} a[i] \oplus b[i] \oplus a[i-1] & \text{if } \text{tr}_{i-1}(s) = 0 | *^{i-1}, \\ a[i] \oplus b[i] \oplus a[i-2] & \text{if } \text{tr}_{i-1}(s) = 1 | 0 | *^{i-2}, \\ \vdots & \\ a[i] \oplus b[i] \oplus a[i-d] & \text{if } \text{tr}_{i-1}(s) = 1^{d-1} | 0 | *^{i-d}, \end{cases}$$

$$\Delta z[i] = \begin{cases} a[i] \oplus b[i] & \text{if } \text{tr}_{i-1}(s) = 1 | *^{i-1}, \\ a[i] \oplus b[i] \oplus a[i-1] \oplus a[i-2] & \text{if } \text{tr}_{i-1}(s) = 0 | 0 | *^{i-2}, \\ \vdots & \\ a[i] \oplus b[i] \oplus a[i-1] \oplus a[i-d] & \text{if } \text{tr}_{i-1}(s) = 0 | 1^{d-2} | 0 | *^{i-d}, \end{cases}$$

where $\Delta z[i] = z[i] \oplus z[i-1]$.

As stated in **Theorem 2**, when partitioning is applied with a d -bit tail, either $z[i]$ or $z[i] \oplus z[i-1]$ can be determined with probability $1 - 2^{-d}$. If the attacker need to guess either a or b , a d -bit guess will be required to determine the correct partition. The theorem is used in the key recovery attacks proposed in [29] and also in our improved attacks. For a comprehensive exposition of the partitioning technique, we refer the readers to [18].

E. The Bit Puncturing Technique

At EUROCRYPT 2024, Flórez-Gutiérrez and Todo [30] introduced the *Walsh spectrum puncturing* technique to strengthen linear key recovery attacks. In this attack, the attacker should find a map g which approximates the target function f from the key recovery attack, and use g instead of f to refine the key recovery process. The main idea of Walsh spectrum puncturing is given by the following proposition.

Proposition 1 [29]. Let $f : \mathbb{F}_2^r \times \mathbb{F}_2^s \rightarrow \mathbb{F}_2$ be the key recovery map for one ciphertext in a differential-linear attack, and let $g : \mathbb{F}_2^r \times \mathbb{F}_2^s \rightarrow \mathbb{R}$. Then f can be swapped for g by increasing the data complexity by a factor of ρ^{-4} , where $\rho = \frac{|(f,g)|_1}{\|f\|_2 \cdot \|g\|_2}$.

Given a set of indices $\mathcal{P} \subseteq \mathbb{F}_2^r \times \mathbb{F}_2^s$, let g be a *punctured* version of f , so that $\hat{g}(u, v) = \hat{f}(u, v)$ if $(u, v) \notin \mathcal{P}$, and $\hat{g}(u, v) = 0$ otherwise. Then the data complexity increases by $(1 - \epsilon)^{-2}$, where $\epsilon = \sum_{(u,v) \in \mathcal{P}} \hat{f}(u, v)^2$.

Here, the inner product $\langle f, g \rangle$ and 2-norm $\|f\|_2$ are defined as

$$\langle f, g \rangle = \frac{1}{2^n} \sum_{x \in \mathbb{F}_2^n} f(x)g(x),$$

$$\|f\|_2 = \sqrt{\langle f, f \rangle} = \sqrt{\frac{1}{2^n} \sum_{x \in \mathbb{F}_2^n} f(x)^2}.$$

At EUROCRYPT 2025, Flórez-Gutiérrez and Todo [29] applied the *bit puncturing* technique which is one of several puncturing strategies to ChaCha. The formal definition of bit puncturing is recalled as follows. This approach operates by discarding specific input bits of f to improve the attack complexity.

Definition 1 [29]. Let $f : \mathbb{F}_2^n \rightarrow \mathbb{R}$ be a pseudo-Boolean function, and let $\mathcal{B} \subseteq \{0, \dots, n-1\}$ be a subset of indices. The

bit puncturing of f according to $\mathcal{B}$ is a function $g : \mathbb{F}_2^n \rightarrow \mathbb{R}$ whose Walsh spectrum $\hat{g}$ is given by

$$\hat{g}(u) = \begin{cases} \hat{f}(u) & \text{if } u_i = 0 \text{ for all } i \in \mathcal{B}, \\ 0 & \text{otherwise.} \end{cases}$$

By the definition above, $g(x) = g(x \oplus y)$ holds for any $y \in \text{span}\{e_i : i \in \mathcal{B}\}$, which means that $g(x)$ corresponds to the average of f over all inputs gained by flipping the bits indexed by $\mathcal{B}$.

Proposition 2 [29]. Let g be the bit puncturing of $f : \mathbb{F}_2^n \rightarrow \mathbb{R}$ according to $\mathcal{B} \subseteq \{0, \dots, n-1\}$. Then

$$g(x) = \frac{1}{2^{|\mathcal{B}|}} \sum_{t \in \mathbb{F}_2^{|\mathcal{B}|}} f\left(x \oplus \bigoplus_{i \in \mathcal{B}} t_i \cdot e_i\right).$$

At EUROCRYPT 2025 [29], the main difference of bit puncturing from the previous techniques are summarized. In the context of the PNB technique, the non-significant bits are assumed to be fixed values (usually zeros), whereas bit puncturing accounts for all possible assignments to the punctured bits and uses their average in the analysis. Furthermore, the partitioning technique is limited to handling a single modular addition per key recovery phase, whereas bit puncturing can handle the entire ARX structure.

III. REBITP: A REFINED FRAMEWORK OF DIFFERENTIAL-LINEAR CRYPTANALYSIS

This section will introduce a novel *guessed key covering technique*. The objective of this technique is to identify partitioning-based functions whose required key bits are covered by those guessed for bit puncturing-based functions, enabling them to be processed jointly. Building on this, we propose a refined framework for differential-linear cryptanalysis of ChaCha called ReBitP, which integrates bit puncturing and partitioning with a two-phase distillation strategy. The key insight of ReBitP is to incorporate a carefully selected set of functions that are evaluated using the partitioning technique with different tail lengths into the first phase, without increasing the number of guessed key bits. Since these functions enable early filtering of invalid keystream pairs via parity checks in the first phase, this will simultaneously lower the time cost of the first phase and the time complexity of constructing the distillation table in the second phase.

A. The Basic Framework

Suppose that the key recovery is decomposed into a sum of m functions, among which:

- a functions are evaluated using the bit puncturing technique, with a correlation coefficient $\rho_{\text{puncturing}}$.
- b functions are evaluated using the partitioning technique, with a corresponding probability $p_{\text{partitioning}}$.

Let $\mathcal{T}$ be the set of distinct tail lengths for the b functions, and for each $i \in \mathcal{T}$, suppose there are b_i functions that use

an i -bit tail. By definition, we have $b = \sum_{i \in \mathcal{T}} b_i$. The overall partitioning probability is therefore

$$p_{\text{partitioning}} = \prod_{i \in \mathcal{T}} (1 - 2^{-i})^{b_i},$$

since each i -bit tail contributes a factor of $1 - 2^{-i}$.

The total correlation of the combined key recovery mapping can be expressed as

$$\rho^2 = \rho_{\text{puncturing}}^2 \cdot p_{\text{partitioning}}.$$

Let N' denote the number of effective samples utilized in the attack, and ϵ denote the correlation of the differential-linear distinguisher. The experimental correlation for a given key guess will follow one of the following two distributions:

- For the correct key, the experimental correlation follows the normal distribution $\mathcal{N}(\epsilon, 1/N')$.
- For an incorrect key, the experimental correlation follows the normal distribution $\mathcal{N}(0, 1/N')$.

According to **Proposition 1**, the total data complexity should be calculated as

$$N = N' \cdot \rho^{-4}.$$

To ensure the attack feasible, the data complexity N must satisfy:

- $N \leq 2^{96}$ if the right pair technique is employed. When the right pair technique is employed, only three of the four IV words (i.e., a total of 96 IV bits) can be freely used to construct samples. Thus, the data complexity is bounded above by 2^{96} .
- $N \leq 2^{128}$ if the right pair technique is not employed. When the right pair technique is not employed, all four IV words (i.e., a total of 128 IV bits) can be freely used to construct samples. Thus, the data complexity must be not bigger than 2^{128} .

In this framework, we will employ the distillation table technique to accelerate the key recovery process. However, the keystream block is large in size for ChaCha (i.e., 512 bits), making a direct distillation table impractical. To overcome this limitation, we use a two-phase distillation strategy, which divides the key recovery process into two phases. In the first phase, we will estimate the selected functions from the decomposition to reduce the involved keystream bits. Once the set of involved keystream bits becomes sufficiently small, we advance to the second phase for distillation.

Since the a functions evaluated with the puncturing technique are not suitable for direct integration into a distillation table, they are all processed in the first phase. Assuming that evaluating these a functions requires guessing n_{gk_1} key bits, the time complexity of the first phase for N keystream pairs is $T_1 = 2N \cdot 2^{n_{gk_1}}$. Given that we need to construct a distillation table from the output of the first phase, the table construction complexity is $T_2 = N \cdot 2^{n_{gk_1}}$. After the table is built, let T_3 denote the complexity of the step-by-step evaluation of the remaining functions. Therefore, the total time complexity is:

$$T = T_1 + T_2 + T_3 = 3N \cdot 2^{n_{gk_1}} + T_3$$

Algorithm 1 Guessed Key Covering Technique

Require: a functions that use puncturing, b functions that use partitioning, tail length set $\mathcal{T}$, guessed key bits n_{gk_1}

Ensure: Selected function count b_{phase_1} , tail length subset $\mathcal{T}_1 \subseteq \mathcal{T}$

- 1: Initialize $b_{\text{phase}_1} \leftarrow 0$, $\mathcal{T}_1 \leftarrow \emptyset$
- 2: **for** each function f_j in the b functions that use partitioning **do**
- 3: **for** each tail length $i \in \mathcal{T}$ **do**
- 4: Determine the set of key bits $K_{f_j,i}$ involved in f_j with tail length i
- 5: **if** $K_{f_j,i} \subseteq K_{n_{gk_1}}$ **then**
- 6: Select f_j with tail length i
- 7: $b_{\text{phase}_1} \leftarrow b_{\text{phase}_1} + 1$
- 8: $\mathcal{T}_1 \leftarrow \mathcal{T}_1 \cup \{i\}$
- 9: **break**
- 10: **end if**
- 11: **end for**
- 12: **end for**
- 13: **return** b_{phase_1} , $\mathcal{T}_1$

To reduce the overall attack complexity, we introduce a novel *guessed key covering technique* to optimize the processing of the m functions. Since the bit puncturing technique is more complex than the partitioning technique, evaluating a single function via bit puncturing requires guessing significantly more key bits than via partitioning. We observe that the key bits required for certain functions evaluated using the partitioning technique are covered by the key bits guessed for the bit puncturing technique. Consequently, we can employ our *guessed key covering technique* to identify these partitioning-based functions and process them jointly with the puncturing-based functions.

Guessed Key Covering Technique: For the a functions evaluated using the bit puncturing technique, a total of n_{gk_1} key bits must be guessed. We then examine the b functions evaluated using the partitioning technique. For each function, we test different tail sizes in descending order. If the key bits involved in a function (under a specific tail configuration) are fully covered by the n_{gk_1} key bits, we select this function. This process continues until all b functions have been assessed. The detailed procedure is outlined in Algorithm 1.

Following the application of the *guessed key covering technique*, we employ a two-phase distillation strategy. In the first phase, all a functions evaluated using the bit puncturing technique are processed jointly with the b_{phase_1} selected functions. Benefiting from the *guessed key covering technique*, incorporating these b_{phase_1} functions does not increase the number of key bits to be guessed. **Phase 1** consists of the following operations:

Phase 1:

- 1) Evaluate all a functions using the bit puncturing technique, which requires guessing n_{gk_1} key bits.
- 2) Select b_{phase_1} functions from the b candidates using the *guessed key covering technique*, whose tail lengths form

a subset $\mathcal{T}_1 \subseteq \mathcal{T}$.

- 3) Evaluate these b_{phase_1} functions using the partitioning technique with their corresponding tail lengths.
- 4) Filter out invalid keystream pairs and retain only those satisfying all partitioning conditions.

Phase 1 outputs a set of valid keystream pairs by filtering out invalid ones, which are used to construct the distillation table in **Phase 2**.

Phase 2:

- 1) Let the tail lengths of the remaining $b_{\text{phase}_2} = b - b_{\text{phase}_1}$ functions form a subset $\mathcal{T}_2 \subseteq \mathcal{T}$.
- 2) Process these b_{phase_2} functions using the partitioning technique with their corresponding tail lengths in $\mathcal{T}_2$, which requires guessing n_{gk_2} key bits.
- 3) Before evaluating any of these functions, construct the distillation table from the output of **Phase 1**.
- 4) Evaluate the remaining b_{phase_2} functions one by one in any order, merging table entries based on each function's output parity. After evaluating each function, reduce the dimension of the distillation table accordingly.
- 5) After all functions have been evaluated, compute the statistical counter for each guessed key. If the counter exceeds a threshold Θ , add the corresponding guessed key into the key candidate list. Otherwise, try the next guessed key.

By **Phase 2**, the attacker can obtain the key candidate list and then find the correct key by verifying each of the list.

B. Complexity Analysis

Phase 1: For each tail length $i \in \mathcal{T}_1$, let b'_i denote the number of functions using an i -bit tail. Note that $b_{\text{phase}_1} = \sum_{i \in \mathcal{T}_1} b'_i$. Thus, the probability that all partitioning conditions are satisfied for a single keystream can be calculated as

$$p_{\text{partitioning}_1} = \prod_{i \in \mathcal{T}_1} (1 - 2^{-i})^{b'_i},$$

since each i -bit tail contributes a factor of $1 - 2^{-i}$.

Since partitioning should be applied independently to each of the keystreams in a pair, the fraction of valid pairs is $p_{\text{partitioning}_1}^2$. Given N keystream pairs and guessing n_{gk_1} key bits, **Phase 1** has a time complexity of

$$T_1 = 2N \cdot p_{\text{partitioning}_1}^2 \cdot 2^{n_{gk_1}}.$$

Phase 2: We construct a distillation table from the output of **Phase 1**. The construction processes each valid pair once, yielding a time cost of

$$T_2 = N \cdot p_{\text{partitioning}_1}^2 \cdot 2^{n_{gk_1}}.$$

For the remaining $b_{\text{phase}_2} = \sum_{i \in \mathcal{T}_2} b''_i$ functions, where b''_i denotes the number of functions with an i -bit tail, the distillation table's index contains the n_{gk_1} key bits guessed in **Phase 1**, and for each remaining function, i bits from each keystream pair (i.e., $2i$ bits per i -bit tail function). Summing over all tail lengths $i \in \mathcal{T}_2$, the total amount of keystream bits

in the index is $\sum_{i \in \mathcal{T}_2} 2i \cdot b_i''$. Thus, the initial table dimension is

$$D = n_{gk_1} + \sum_{i \in \mathcal{T}_2} 2i \cdot b_i'',$$

and the partitioning probability for **Phase 2** is

$$p_{\text{partitioning}_2} = \prod_{i \in \mathcal{T}_2} (1 - 2^{-i})^{b_i''}.$$

Then, each function is evaluated sequentially. When evaluating a function with an i -bit tail, we exhaustively guess the involved i key bits, which results in a time cost of $2^D \cdot 2^i$ operations. After evaluating a function, the table dimension is reduced. Let C_i denote the total cost for all functions with i -bit tails. The evaluation complexity can be calculated as

$$T_3 = \sum_{i \in \mathcal{T}_2} C_i.$$

Therefore, the total time complexity is

$$T = T_1 + T_2 + T_3 = 3N \cdot p_{\text{partitioning}_1}^2 \cdot 2^{n_{gk_1}} + \sum_{i \in \mathcal{T}_2} C_i,$$

and the total data complexity remains $N = N' \cdot \rho^{-4}$.

Success Probability: Let n_{gk} denote the total amount of key bits that have to be guessed, k_{wrong} the number of incorrect key candidates, and p the right pair probability. Note that $p = 1$ holds when the right pair technique is not used in the attack. Let Φ represent the cumulative distribution function of the standard normal distribution. Following the analysis in [18], we know the attacker expects to find one right pair with probability 0.5 when the attack is repeated p^{-1} times, and thus the success probability p_{success} can be calculated as

$$p_{\text{success}} = \frac{1}{2} \left(1 - \Phi \left(\frac{\Theta - N'\epsilon}{\sqrt{N'}} \right) \right),$$

where the decision threshold Θ is given by

$$\Theta = \sqrt{N'} \cdot \Phi^{-1} \left(1 - \frac{k_{\text{wrong}} \cdot p}{2^{n_{gk}}} \right).$$

In other words, once the right pair has been successfully obtained, the success probability of our attack is $2 \cdot p_{\text{success}}$.

C. Trade-off Analysis

Substituting $N = N' \cdot \rho^{-4}$ into the time complexity formula, we obtain:

$$\begin{aligned} T &= 3N \cdot p_{\text{partitioning}_1}^2 \cdot 2^{n_{gk_1}} + \sum_{i \in \mathcal{T}_2} C_i \\ &= 3N' \cdot \rho^{-4} \cdot p_{\text{partitioning}_1}^2 \cdot 2^{n_{gk_1}} + \sum_{i \in \mathcal{T}_2} C_i \\ &= 3N' \cdot \rho_{\text{puncturing}}^{-4} \cdot p_{\text{partitioning}_2}^{-2} \cdot 2^{n_{gk_1}} + \sum_{i \in \mathcal{T}_2} C_i \end{aligned}$$

First, consider the scenario where N' remains constant. Within our framework, to maintain the number of key bits guessed in **Phase 1**, we typically select functions that use

partitioning with smaller tail lengths for processing in **Phase 1**. According to

$$p_{\text{partitioning}} = \prod_{i \in \mathcal{T}} (1 - 2^{-i})^{b_i},$$

smaller tail lengths correspond to a lower partitioning probability. Consequently, assigning more functions to **Phase 1** reduces $p_{\text{partitioning}}$. Given the relationship

$$N = N' \cdot \rho^{-4} = N' \cdot \rho_{\text{puncturing}}^{-4} \cdot p_{\text{partitioning}}^{-2},$$

with $\rho_{\text{puncturing}}$ fixed, a smaller $p_{\text{partitioning}}$ correspond to a larger N . Since the total number of functions that use partitioning remains constant, increasing the count in **Phase 1** decreases the count in **Phase 2**. This results in a larger $p_{\text{partitioning}_2}$, thereby reducing the time complexity T . In summary, by controlling the number of functions that use partitioning in **Phase 1**, we can achieve a time-data trade-off.

In practice, cryptanalytic attacks often prioritize optimizing time complexity at the expense of data complexity. Consequently, we aim to maximize the number of functions that use partitioning in **Phase 1** without increasing the number of guessed key bits. However, given data complexity constraints (e.g., $N \leq 2^{96}$ or $N \leq 2^{128}$), certain partitioning-based functions eligible for **Phase 1** are forced into **Phase 2**. To address this limitation, we consider a second optimization scenario: achieving further improvements by adjusting the value of N' . Since N' is correlated with the success probability, this enables us to explore a time-data-success probability trade-off.

Building on the analysis of the first scenario, incorporating more functions that use partitioning into **Phase 1** reduces $p_{\text{partitioning}}$. According to the relation $N = N' \cdot \rho^{-4}$, this causes the required data complexity N to increase. When N approaches its upper bound, we can further reduce time complexity by decreasing N' . This adjustment compensates for the increase in ρ^{-4} , thereby freeing up capacity to include additional functions that use partitioning in **Phase 1** without exceeding the data limit. Furthermore, based on the time complexity formula, T is positively correlated with N' ; consequently, a smaller N' directly results in a lower time complexity.

However, since p_{success} is positively correlated with N' , reducing N' inevitably lowers the success probability. This establishes a three-way trade-off among time complexity, data complexity, and success probability:

- To minimize time complexity, more functions that use partitioning with small tail lengths (e.g., $i = 1, 2, 3$) should be prioritized for processing in **Phase 1**; additionally, a smaller N' should be selected to free up capacity when N approaches its upper bound.
- To minimize data complexity, all functions that use partitioning should be evaluated in **Phase 2**, preferably employing large tail lengths.
- To maximize success probability, a larger N' should be chosen.

Therefore, the attacker can use the framework ReBitP to obtain a meaningful trade-off by balancing the time complexity, the data complexity and the success probability. This

will be demonstrated by applying the framework ReBitP to ChaCha256 in the following sections.

D. Differences of ReBitP from the Existing Work

At EUROCRYPT 2025, the bit puncturing and partitioning techniques were utilized to propose key recovery attacks on ChaCha256 in [29]. The new framework ReBitP significantly generalizes their approach in the following aspects.

- The framework ReBitP generalizes their approach. By carefully selecting functions that employ the partitioning technique with small tail lengths for inclusion in **Phase 1**, ReBitP can achieve not only the cryptanalytic results obtained in [29], but also better cryptanalytic results through this refined framework. Overall, ReBitP combines the ideas of the bit puncturing technique, the partitioning technique, a two-phase distillation strategy and the guessed key covering technique proposed in this paper. Thus, ReBitP is a refined framework of the existing work [29] and is significantly different from it.
- We formally analyze the trade-off relationship of the bit puncturing-based attack between the time complexity, the data complexity and the success probability, which is not addressed in [29]. Undoubtedly, this provides a better understanding of the existing approach in [29] and offers the attacker greater flexibility in selecting cryptanalytic parameters.

Therefore, ReBitP is a refined framework of differential-linear cryptanalysis compared with the existing work. The idea of ReBitP is general and applicable to ARX-based ciphers. We will demonstrate the advantages of ReBitP by applying it to ChaCha256 in the following sections.

IV. ENHANCED ATTACK ON 7-ROUND CHACHA256

In this section, we will present an enhanced key recovery attack on 7-round ChaCha256 by applying the ReBitP framework.

A. Linear Decomposition in Key Recovery

In [25], Bellini et al. obtained a 4-round differential-linear distinguisher for ChaCha with correlation $2^{-34.15}$, i.e.,

$$\Delta^{(1)} \rightarrow \Gamma^{(5)},$$

where

$$\begin{aligned} \Delta^{(1)} &= (\Delta_{3,25}^{(1)}, \Delta_{3,5}^{(1)}, \Delta_{7,28}^{(1)}, \Delta_{7,12}^{(1)}, \Delta_{11,25}^{(1)}, \Delta_{11,21}^{(1)}, \Delta_{15,21}^{(1)}, \Delta_{15,13}^{(1)}), \\ \Gamma^{(5)} &= x_2^{(5)}[0] \oplus x_6^{(5)}[7] \oplus x_6^{(5)}[19] \oplus x_{10}^{(5)}[12] \oplus x_{14}^{(5)}[0]. \end{aligned}$$

Subsequently, Xu et al. [26] further improved the correlation from $2^{-34.15}$ to $2^{-32.2}$ by exploiting the differential-linear hull effect. By using the right pair technique, a 1-round differential trail $\Delta_{in} \rightarrow \Delta^{(1)}$ with probability $2^{-5.2}$ was found in [32], where $\Delta_{in} = (\Delta x_{15,9}^{(0)}, \Delta x_{15,29}^{(0)})$.

In our enhanced attack on 7-round ChaCha256, we utilize a 5-round distinguisher for ChaCha256, which is obtained through combining the 1-round differential trail $\Delta_{in} \rightarrow \Delta^{(1)}$ with the 4-round distinguisher $\Delta^{(1)} \rightarrow \Gamma^{(5)}$ described above. Like the attack on 7-round ChaCha256 proposed in [29], we

also express the key recovery mapping as a sum of some functions f_i in our attack, and thus obtain the same 24 functions:

$$\begin{aligned} f_1 &= (-1)^{v_{10}^{5,5}[12] \oplus v_7^{7,7}[19] \oplus v_{11}^{7,7}[12]}, \\ f_2 &= (-1)^{v_{11}^{5,5}[19] \oplus v_{11}^{6,6}[19] \oplus v_4^{7,7}[26] \oplus v_8^{7,7}[19]}, \\ f_3 &= (-1)^{v_{11}^{5,5}[7]}, \quad f_4 = (-1)^{v_3^{6,6}[16]}, \quad f_5 = (-1)^{v_2^{6,5}[24]}, \end{aligned}$$

each of which involves more than one modular subtraction, and

$$\begin{aligned} f_6 &= (-1)^{v_5^{7,7}[7]}, & f_7 &= (-1)^{v_6^{7,7}[25]}, & f_8 &= (-1)^{v_6^{7,7}[19]}, \\ f_9 &= (-1)^{v_6^{7,7}[13]}, & f_{10} &= (-1)^{v_7^{7,7}[7]}, & f_{11} &= (-1)^{v_7^{7,7}[26]}, \\ f_{12} &= (-1)^{v_7^{7,7}[7] \oplus v_7^{7,7}[6]}, & f_{13} &= (-1)^{v_{10}^{7,7}[18]}, & f_{14} &= (-1)^{v_{10}^{7,7}[12]}, \\ f_{15} &= (-1)^{v_{10}^{7,7}[6]}, & f_{16} &= (-1)^{v_{11}^{7,7}[31]}, & f_{17} &= (-1)^{v_{11}^{7,7}[19]}, \\ f_{18} &= (-1)^{v_8^{6,6}[12]}, & f_{19} &= (-1)^{v_{11}^{6,6}[31]}, & f_{20} &= (-1)^{v_8^{6,5}[7]}, \\ f_{21} &= (-1)^{v_{10}^{6,5}[26]}, & f_{22} &= (-1)^{v_{10}^{6,5}[6]}, & f_{23} &= (-1)^{v_{11}^{6,5}[19]}, \\ f_{24} &= (-1)^{v_{11}^{6,5}[7]}, \end{aligned}$$

each of which involves only one single modular subtraction.

B. Puncturing the Functions of the Decomposition

We utilize the partitioning technique for f_6 to f_{24} , and employ the bit puncturing technique for f_1 to f_5 . For f_1 to f_5 , we use the punctured map proposed in Section 6.3 of [29]. As shown in Table V of [29], the puncturing correlation for these five functions can be calculated as

$$\rho_{\text{puncturing}}^2 = 2^{-(7.14+2.39+2.57+3.07+1.74)} \approx 2^{-16.91}.$$

To achieve better results, we utilize ReBitP to handle the functions f_6 to f_{24} . Specifically, unlike [29], we employ the partitioning technique with a 2-bit tail for f_{17} , f_{18} , f_{23} and a 1-bit tail for f_{14} , f_{20} in **Phase 1**. For the remaining 14 functions, we continue to use the partitioning technique with a 3-bit tail in **Phase 2**. Thus, these parities can be determined with probability

$$p_{\text{partitioning}} = \left(\frac{7}{8}\right)^{14} \cdot \left(\frac{3}{4}\right)^3 \cdot \left(\frac{1}{2}\right)^2 \approx 2^{-5.94}.$$

In this process, we need to guess 54 bits totally.

C. Enhanced Key Recovery Using ReBitP

We will apply our framework ReBitP to 7-round ChaCha256 in two distinct scenarios. In the first scenario, the right pair technique is not considered. In other words, $\Delta_{in} \rightarrow \Gamma^{(5)}$ is regarded as a 5-round DL distinguisher with correlation $2^{-32.2-5.2} = 2^{-37.4}$. In the second scenario, the right pair technique is considered. In this subsection, we focus on the first scenario.

According to our framework, the experimental correlation is distributed as $\mathcal{N}(2^{-37.4}, 1/N')$ for the correct key, while it follows $\mathcal{N}(0, 1/N')$ for an incorrect key. Thus, the data complexity of our enhanced attack can be estimated as $N = N' \cdot \rho^{-4}$, where $\rho^2 = \rho_{\text{puncturing}}^2 \cdot p_{\text{partitioning}}$ is the puncturing correlation for f_1 to f_{24} . In [29], the authors chose $N' = 2^{85.5}$ and presented an attack on 7-round ChaCha256

with a time/data complexity of $2^{148.23}/2^{127.65}$. However, this is not optimal under our framework. Benefiting from ReBitP, we can carefully select the functions that use the partitioning technique in **Phase 1** to obtain a better attack. In our enhanced attack, we choose $N' = 2^{81.29}$ and then we can achieve a success probability of 88.27% with 54-bit advantage. Consequently, we have

$$N = N' \cdot 2^{(16.91+5.94) \times 2} \approx 2^{126.99}.$$

Hence, the data complexity of our enhanced attack is $2^{127.99}$, which is not bigger than 2^{128} .

Phase 1. By applying ReBitP, we evaluate all 5 punctured functions f_1 to f_5 in the first phase. We also use the partitioning technique with a 2-bit tail for f_{17} , f_{18} , f_{23} and a 1-bit tail for f_{14} , f_{20} in the first phase. In this process, we need to guess $n_{gk_1} = 20$ key bits, i.e.,

$$\begin{aligned} & k_4[30, 18, 13], k_6[30], k_7[26, 22, 2], \\ & k_8[23, 11, 10, 6], k_{10}[23, 11], \\ & k_{11}[27, 19, 18, 17, 15, 14, 6], \end{aligned}$$

and the correlation is $p_{\text{partitioning}_1} = \left(\frac{3}{4}\right)^3 \cdot \left(\frac{1}{2}\right)^2 \approx 2^{-3.245}$. The time cost of this phase can be computed as

$$T_1 = 2N \cdot p_{\text{partitioning}_1}^2 \cdot 2^{20} \approx 2N \cdot 2^{-6.49} \cdot 2^{20} \approx 2^{141.5}.$$

Phase 2. This phase applies the partitioning technique with a 3-bit tail to the remaining functions, utilizing the following 36 key bits:

$$\begin{aligned} & k_5[6, 5, 4], k_6[24, 23, 22, 18, 17, 16, 12, 11, 10, 6, 5, 4], \\ & k_7[25, 24, 23, 6, 5, 4], k_{10}[25, 24, 23, 17, 16, 15, 5, 4, 3], \\ & k_{11}[30, 29, 28, 6, 5, 4]. \end{aligned}$$

It should be noted that some key bits are already present in **Phase 1**, so we only need to guess $n_{gk} = 54$ bits totally.

Since each function employs a 3-bit tail, the size of the distillation table is given by

$$D = n_{gk_1} + \sum_{i \in T_2} 2i \cdot b_i'' = 20 + 2 \times 3 \times 14 = 104.$$

In this phase, we first build the distillation table through traversing $2^{140.5}$ pairs, and thus the table construction complexity is $T_2 = 2^{140.5}$. Subsequently, the remaining functions are evaluated sequentially, and no more than 3 key bits have to be guessed at each step. Specifically, the evaluation of f_{24} requires an additional 2-bit guess, leading to a time cost of $2^{104+2} = 2^{106}$ for this step. After this step, the dimension of this distillation table can be reduced to be $22+2 \times 3 \times 13 = 100$. Since $2^{140.5} \gg 2^{106}$, the total time cost of evaluating the remaining functions $T_3 = 2^{106}$ is so small that can be ignored. Hence, the overall time complexity of our enhanced attack in the first scenario can be calculated as

$$T \approx T_1 + T_2 = 2^{141.5} + 2^{140.5} \approx 2^{142.08}.$$

Remark. We summarize the parameter settings of our attack alongside those of [29] in Table II. As evident from the

results, using our ReBitP framework allows us to reduce the time complexity by a factor of $2^{6.2}$ compared to [29]. This improvement is achieved by carefully selecting partitioning-based functions for inclusion in **Phase 1** and optimizing the value of N' . Notably, while we explored various combinations of functions and values for N' , we also included a comparative experiment, as shown in the last row of Table II. Our final parameter selection represents the optimal trade-off, achieving minimal time complexity while maintaining a high success probability.

D. Reducing the Data Complexity

Now, we consider the second scenario, i.e., the right pair technique is considered. In [29], the authors chose $N' = 2^{72}$ and obtained the other attack on 7-round ChaCha256 with time/data complexity $2^{154.2}/2^{102.85}$. However, this is also not optimal under our framework. Benefiting from ReBitP, we can choose a smaller value for N' to achieve a better attack. We follow the same strategy as in [29]. Specifically, the puncturing correlation of **Phase 1** is estimated to be approximately $2^{-8.54}$, and the correlation of **Phase 2** is about $2^{-2.89}$. When $N' = 2^{71.85}$, the attack achieves a success probability of 99.32% with 83-bit advantage. Consequently,

$$N = N' \cdot 2^{(8.54+2.89) \times 2} \approx 2^{94.71}.$$

Hence, the data complexity of our enhanced attack is $2^{95.71}$, which is not bigger than 2^{96} . In **Phase 1**, we apply the partitioning technique with a 7-bit tail for f_{17} and f_{23} , a 9-bit tail for f_{18} , and a 4-bit tail for f_{20} , requiring a total of $n_{gk_1} = 51$ key bits to be guessed. Therefore, these parities can be determined with probability

$$p_{\text{partitioning}_1} = \left(\frac{15}{16}\right) \cdot \left(\frac{511}{512}\right) \cdot \left(\frac{127}{128}\right)^2 \approx 2^{-0.12}.$$

The time complexity is

$$T_1 = 2N \cdot p_{\text{partitioning}_1}^2 \cdot 2^{51} \approx 2N \cdot 2^{-0.24} \cdot 2^{51} \approx 2^{146.47}.$$

In **Phase 2**, the partitioning technique is applied with a 3-bit tail to the remaining 15 functions. Taking into consideration the overlapping key bits between **Phase 1** and **Phase 2**, the total amount of distinct key bits which have to be guessed is $n_{gk} = 83$. The dimension of the required distillation table can be computed as

$$D = n_{gk_1} + \sum_{i \in T_2} 2i \cdot b_i'' = 51 + 2 \times 3 \times 15 = 141.$$

We first construct the distillation table through traversing $2^{145.47}$ pairs, resulting in a table construction cost of $T_2 = 2^{145.47}$. Subsequently, the remaining functions are evaluated sequentially, and no more than 3 key bits have to be guessed at each step. Specifically, the evaluation of f_{24} requires an additional 1-bit guess, yielding a time cost of $2^{141+1} = 2^{142}$ for this step. After this evaluation, the dimension of this distillation table can be reduced to be $52+2 \times 3 \times 14 = 136$. Since the cost of this initial evaluation dominates the whole process, the total time cost of evaluating the remaining functions is approximately $T_3 = 2^{142}$.

TABLE II: Parameter Selection for Key Recovery Attack on 7-Round ChaCha

Phase 1 Functions	Phase 2 Functions	N'	Time	Data	Success Probability	Ref.
f_1-f_5	f_6-f_{24}	$2^{85.5}$	$2^{148.23}$	$2^{127.65}$	99.79%	[29]
$f_1-f_5, f_{14}, f_{17}, f_{18}, f_{23}$	$f_6-f_{24} \setminus \{f_{14}, f_{17}, f_{18}, f_{23}\}$	$2^{82.9}$	$2^{144.08}$	$2^{127.99}$	99.99%	This paper (comparative)
$f_1-f_5, f_{14}, f_{17}, f_{18}, f_{20}, f_{23}$	$f_6-f_{24} \setminus \{f_{14}, f_{17}, f_{18}, f_{20}, f_{23}\}$	$2^{81.29}$	$2^{142.08}$	$2^{127.99}$	88.27%	This paper

Therefore, the time complexity of our enhanced attack in the second scenario can be calculated as

$$T = T_1 + T_2 + T_3 = 2^{146.47} + 2^{145.47} + 2^{142} \approx 2^{147.1}.$$

Since the 1-round differential trail $\Delta_{in} \rightarrow \Delta^{(1)}$ holds with a probability of $2^{-5.2}$, the attack must be repeated approximately $2^{5.2}$ times to obtain a sufficient number of valid right pairs. Consequently, our enhanced attack on 7-round ChaCha256 in the second scenario has a final time complexity of $2^{147.1} \cdot 2^{5.2} = 2^{152.3}$, with a final data complexity of $2^{95.71} \cdot 2^{5.2} = 2^{100.91}$.

V. ENHANCED ATTACK ON 7.5-ROUND CHACHA256

In this section, we will present an enhanced key recovery attack on 7.5-round ChaCha256 by applying the ReBitP framework.

A. Linear Decomposition in Key Recovery

Our attack on 7.5-round ChaCha256 utilizes the same 5-round distinguisher as in the attack on 7-round ChaCha256 and expresses the key recovery mapping as a sum of some functions f_i . We use the same puncturing strategy as in [29], perform a full 32-bit guess of k_4 and a partial 20-bit guess of k_9 [19–0], and thus obtain the same 28 functions as follows:

$$\begin{aligned} f_1 &= (-1)^{v_{10}^{5.5}[12] \oplus v_{11}^{7.5}[19] \oplus v_{11}^{7.5}[12] \oplus v_7^{7.5}[31] \oplus v_7^{7.5}[6] \oplus v_8^{7.5}[26] \oplus v_8^{7.5}[19]}, \\ f_2 &= (-1)^{v_{11}^{5.5}[19] \oplus v_{11}^{6.5}[19] \oplus v_7^{7.5}[26] \oplus v_8^{7.5}[19]}, \\ f_3 &= (-1)^{v_{11}^{5.5}[7] \oplus v_{11}^{6.5}[7] \oplus v_5^{7.5}[19] \oplus v_{10}^{7.5}[7]}, \\ f_4 &= (-1)^{v_3^{6.5}[16]}, \quad f_5 = (-1)^{v_8^{6.5}[12]}, \quad f_6 = (-1)^{v_{11}^{6.5}[31]}, \\ f_7 &= (-1)^{v_2^{6.5}[24]}, \quad f_8 = (-1)^{v_1^{7.5}[16]}, \quad f_9 = (-1)^{v_2^{7.5}[8]} \end{aligned}$$

each of which involves more than one modular subtraction, and

$$\begin{aligned} f_{10} &= (-1)^{v_6^{7.5}[31]}, \quad f_{11} = (-1)^{v_6^{7.5}[25]}, \quad f_{12} = (-1)^{v_6^{7.5}[19]}, \\ f_{13} &= (-1)^{v_6^{7.5}[5]}, \quad f_{14} = (-1)^{v_7^{7.5}[19] \oplus v_7^{7.5}[18]}, \\ f_{15} &= (-1)^{v_7^{7.5}[12]}, \quad f_{16} = (-1)^{v_8^{7.5}[7] \oplus v_8^{7.5}[6]}, \\ f_{17} &= (-1)^{v_{11}^{7.5}[25]}, \quad f_{18} = (-1)^{v_{11}^{7.5}[19]}, \quad f_{19} = (-1)^{v_{11}^{7.5}[13]}, \\ f_{20} &= (-1)^{v_{11}^{7.5}[7]}, \quad f_{21} = (-1)^{v_8^{6.5}[7]}, \quad f_{22} = (-1)^{v_{10}^{6.5}[26]}, \\ f_{23} &= (-1)^{v_{10}^{6.5}[6]}, \quad f_{24} = (-1)^{v_{11}^{6.5}[19]}, \quad f_{25} = (-1)^{v_{10}^{7.5}[18]}, \\ f_{26} &= (-1)^{v_{10}^{7.5}[12]}, \quad f_{27} = (-1)^{v_{11}^{7.5}[6]}, \quad f_{28} = (-1)^{v_{11}^{7.5}[31]}, \end{aligned}$$

each of which involves only one single modular subtraction.

B. Puncturing the Functions of the Decomposition

We utilize the partitioning technique for f_{10} to f_{28} , and employ the puncturing technique for f_1 to f_9 . For f_1 to f_9 , we use the punctured map proposed in Section F.2 of [29], which shows the details of the trail enumeration. The estimated puncturing correlation for f_1 to f_9 is about $2^{-20.21}$.

To achieve better results, we utilize ReBitP to handle the functions f_{10} to f_{28} . Specifically, in [29], the authors employ 6-, 6-, 4-, and 5-bit tails for f_{16} , f_{21} , f_{26} , and f_{28} , respectively, and use 4-bit tails for the remaining 15 functions. In our enhanced attack, we apply the partitioning technique with a 2-bit tail for f_{16} , f_{21} , f_{26} , f_{28} and a 1-bit tail for f_{25} in **Phase 1**. For the remaining 14 functions, we continue to use the partitioning technique with a 4-bit tail in **Phase 2**. Thus, these parities can be determined with probability

$$p_{\text{partitioning}} = \left(\frac{3}{4}\right)^4 \cdot \left(\frac{1}{2}\right) \cdot \left(\frac{15}{16}\right)^{14} \approx 2^{-3.96}.$$

C. Enhanced Key Recovery Using ReBitP

In our enhanced attack, we utilize the 5-round distinguisher with a correlation of $2^{-32.2-5.2} = 2^{-37.4}$. In [29], the authors chose $N' = 2^{82.5}$ and obtained an attack on 7.5-round ChaCha256 with time/data complexity $2^{250.17}/2^{127.08}$. However, this is also not optimal under our framework. Benefiting from ReBitP, we can carefully select the functions that use the partitioning technique in **Phase 1** to obtain a better attack. In our enhanced attack, we choose $N' = 2^{78.65}$ and then we can achieve a success probability of 62.18% with 12-bit advantage. Consequently, we have

$$N = N' \cdot 2^{(20.21+3.96) \times 2} \approx 2^{126.99},$$

which implies a data complexity of $2^{127.99}$ for our enhanced attack. This is valid, since it is not bigger than 2^{128} .

Phase 1. By applying ReBitP, we first evaluate the nine punctured functions $f_1, \dots, f_9$. Additionally, since the functions f_{16} , f_{21} , f_{25} , f_{26} , and f_{28} share numerous active key bits with those involved in $f_1, \dots, f_9$, we also evaluate them in **Phase 1**. In total, this phase guesses $n_{gk_1} = 70$ key bits, i.e.,

$$\begin{aligned} &k_5[23-21, 30, 29, 14, 13, 2, 1], \quad k_6[27, 26, 10, 9], \\ &k_7[23, 22, 19, 18, 14, 13, 6, 5, 3-0], \\ &k_8[26, 25, 23-20, 11-1], \quad k_9[30-26], \\ &k_{10}[23-21, 18, 17, 11-8, 2, 1], \\ &k_{11}[30-26, 19, 18, 15-13, 6, 5]. \end{aligned}$$

In addition to these 70 bits, we fully guess the 32-bit word k_4 and the lower 20 bits of k_9 , i.e., $k_9[19-0]$, resulting in 122 key bits totally that have to be guessed. The fraction of input pairs that survive the partitioning in this phase is

$$p_{\text{partitioning}_1} = \left(\frac{1}{2}\right) \cdot \left(\frac{3}{4}\right)^4 \approx 2^{-2.66}.$$

Hence, the time cost of this phase can be calculated as

$$T_1 = 2N \cdot p_{\text{partitioning}_1}^2 \cdot 2^{122} \approx 2N \cdot 2^{-5.32} \cdot 2^{122} \approx 2^{244.67}.$$

Phase 2. We should evaluate the remaining functions in this phase. Since the partitioning technique is applied to each function with a 4-bit tail, the dimension of the required distillation table can be calculated as

$$D = n_{gk_1} + \sum_{i \in T_2} 2i \cdot b_i'' = 122 + 2 \times 4 \times 14 = 234.$$

We first construct the distillation table through traversing $2^{243.67}$ pairs, resulting in a table construction cost of $T_2 = 2^{243.67}$. Subsequently, the remaining functions are evaluated sequentially, and no more than 4 key bits have to be guessed at each step. Specifically, the evaluation of f_{20} requires an additional 2-bit guess, yielding a time cost of $2^{234+2} = 2^{236}$ for this step. However, since $2^{243.67} \gg 2^{236}$, the total time cost of evaluating the remaining functions $T_3 = 2^{236}$ is small that can be ignored.

Since the attack takes a 12-bit advantage, we need to further guess $256 - 12 = 244$ key bits to finish the entire key recovery. Therefore, our enhanced attack on 7.5-round ChaCha256 has a final time complexity of $T \approx 2^{244.67} + 2^{243.67} + 2^{244} \approx 2^{245.76}$, with a data complexity of $2^{127.99}$.

D. Further Reduction of Time Complexity

The enhanced key recovery attack on 7.5-round ChaCha256 presented in Subsection V-C is obtained using our framework ReBitP. Very recently, Flórez-Gutiérrez and Todo [31] proposed a divide-and-conquer technique for trail enumeration puncturing, which presented a new key recovery attack on 7.5-round ChaCha256. In [31], the authors used a 5-round distinguisher with a correlation of $2^{-39.2}$. Based on this distinguisher, they selected $N' = 2^{83}$ and achieved time and data complexities of $2^{243.62}$ and $2^{127.05}$ respectively, with a success probability of 82%. This is the optimal result they can obtain using this 5-round distinguisher. However, this is also not optimal under our framework, and we can choose a smaller value for N' to achieve a better attack.

In our enhanced attack, we use the 5-round distinguisher with a correlation of $2^{-37.4}$ proposed in IV-A. We choose $N' = 2^{81}$, and then we obtain a 15-bit advantage with a success probability of 99.99%. The data complexity is

$$2 \cdot 2^{81} \cdot 2^{21.52 \times 2} \approx 2^{125.04}.$$

The total time complexity of the two phases is approximately $2^{125.04} \cdot 2^{115} + 2^{125.04} \cdot 2^{115} \approx 2^{241.04}$, when 115 key bits are guessed. Therefore, the overall time complexity of our attack is reduced to be $2^{241.04} + 2^{241} \approx 2^{242.02}$.

The key reason why we can propose a better key recovery attack on 7.5-round ChaCha256 than [31] is that the 5-round distinguisher we use has a higher correlation. Benefiting from this improvement, we can choose a smaller value for N' , and then obtain the optimal trade-off by balancing time complexity, data complexity and success probability using our framework ReBitP. This completely demonstrates the advantage of our framework ReBitP, compared with the existing techniques.

VI. CONCLUSIONS

The security margin of ChaCha is directly related to the safety of many widely used protocols, operating systems and softwares, such as TLS, SSH, Noise, WireGuard, Linux, Android, Chromium/Chrome, Firefox and Safari. This paper introduces a refined framework of differential-linear cryptanalysis of ChaCha called ReBitP, which combines the ideas of the bit puncturing technique, the partitioning technique, a two-phase distillation strategy and the guessed key covering technique proposed in this paper. In contrast to prior work, our ReBitP framework enables the incorporation of a carefully selected set of functions that are evaluated using the partitioning technique with different tail lengths into the first phase, without requiring additional key bits to be guessed. This early filtering via parity checks simultaneously lowers the time cost of the first phase and the time complexity of constructing the distillation table in the second phase. Furthermore, building upon the conventional time-data trade-off, we account for ChaCha's data complexity constraints and introduce a novel time-data-success probability trade-off, which is not addressed in prior work. Undoubtedly, this provides a better understanding of the existing approach and offers the attacker greater flexibility in selecting cryptanalytic parameters.

As applications, enhanced key recovery attacks on 7- and 7.5-round ChaCha256 are presented, achieving time complexities of $2^{142.08}$ and $2^{242.02}$, respectively. The cryptanalytic results are $2^{6.12}$ and $2^{1.58}$ times faster than the existing attacks, respectively. So far as we know, these are the best known key recovery attacks on 7- and 7.5-round ChaCha256. This definitely demonstrates the superiority of the refined framework ReBitP. For future work, we will apply the refined framework ReBitP to other similar ARX-based ciphers, e.g., Salsa, Chaskey and Forró.

REFERENCES

- [1] Bernstein, D.J.: The Salsa20 family of stream ciphers. In: Robshaw, M., Billet, O. (eds.) New Stream Cipher Designs. LNCS, vol. 4986, pp. 84-97. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-68351-3_8
- [2] Bernstein, D.J.: Chacha, a variant of salsa20. In: Workshop Record of SASC, vol.8, pp. 3-5. (2008)
- [3] Dinu, D., Perrin, L., Udovenko, A., Velichkov, V., Großschädl, J., Biryukov, A.: Design strategies for ARX with provable bounds: Sparx and LAX. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10031, pp. 484-513. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53887-6_18
- [4] Aumasson, J.-P., Neves, S., Wilcox-O'Hearn, Z., Winnerlein, C.: BLAKE2: simpler, smaller, fast as MD5. In: Jacobson, M., Locasto, M., Mohassel, P., Safavi-Naini, R. (eds.) ACNS 2013. LNCS, vol. 7954, pp. 119-135. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38980-1_8

- [5] Beierle, C., et al.: Lightweight AEAD and Hashing using the sparkle permutation family. *IACR Transactions on Symmetric Cryptology*, vol. 2020, no. S1, pp. 208-261, 2020. <https://doi.org/10.13154/tosc.v2020.iS1.208-261>
- [6] Mouha, N., Mennink, B., Van Herrewege, A., Watanabe, D., Preneel, B., Verbauwhede, I.: Chaskey: an efficient MAC algorithm for 32-bit microcontrollers. In: Joux, A., Youssef, A. (eds.) *SAC 2014*. LNCS, vol. 8781, pp. 306-323. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-13051-4_19
- [7] eSTREAM: The ecrypt stream cipher project., <https://cr.yp.to/streamciphers.html>
- [8] Bernstein, D.J.: The poly1305-AES message-authentication code. In: Gilbert, H., Handschuh, H. (eds.) *FSE 2005*. LNCS, vol. 3557, pp. 32-49. Springer, Heidelberg (2005). https://doi.org/10.1007/11502760_3
- [9] Langley, A., Chang, W., Mavrogianopoulos, N., Strömbergson, J., Josefsson, S.: Chacha20-poly1305 cipher suites for transport layer security (TLS). RFC 7905(1-8), 10 (2016). <https://doi.org/10.17487/RFC7905>
- [10] Nir, Y.: Chacha20, poly1305, and their use in the internet key exchange protocol (IKE) and IPsec. RFC 7634(1-13), 10 (2015). <https://doi.org/10.17487/RFC7634>
- [11] IANIX: ChaCha usage & deployment (2024). <https://ianix.com/pub/chacha-deployment.html>. Accessed 20 May 2024
- [12] Aumasson, J.-P., Fischer, S., Khazaei, S., Meier, W., Rechberger, C.: New features of Latin dances: analysis of salsa, chacha, and rumba. In: Nyberg, K. (eds.) *FSE 2008*. LNCS, vol. 5086, pp. 470-488. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-71039-4_30
- [13] Shi, Z., Zhang, B., Feng, D., Wu, W.: Improved key recovery attacks on reducedround Salsa20 and ChaCha. In: Kwon, T., Lee, M.-K., Kwon, D. (eds.) *ICISC 2012*. LNCS, vol. 7839, pp. 337-351. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-37682-5_24
- [14] Maitra, S.: Chosen IV cryptanalysis on reduced round chacha and salsa. *Discrete Applied Mathematics*, vol. 208, pp. 88-97, 2016. <https://doi.org/10.1016/j.dam.2016.02.020>
- [15] Choudhuri, A.R., Maitra, S.: Significantly improved multi-bit differentials for reduced round salsa and chacha. *IACR Transactions on Symmetric Cryptology*, vol. 2016, no. 2, pp. 261-287, 2016. <https://doi.org/10.13154/tosc.v2016.i2.261-287>
- [16] Dey, S., Sarkar, S.: Improved analysis for reduced round salsa and chacha. *Discrete Applied Mathematics*, vol. 227, pp. 58-69, 2017. <https://doi.org/10.1016/j.dam.2017.04.034>
- [17] Coutinho, M., Souza, T.C.N.: New multi-bit differentials to improve attacks against ChaCha. *IACR Cryptology ePrint Archive 2020/350* (2020). <https://eprint.iacr.org/2020/350>
- [18] Beierle, C., Leander, G., Todo, Y.: Improved differential-linear attacks with applications to ARX ciphers. In: Micciancio, D., Ristenpart, T. (eds.) *CRYPTO 2020*. LNCS, vol. 12172, pp. 329-358. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-56877-1_12
- [19] Miyashita, S., Ito, R., Miyaji, A.: PNB-focused differential cryptanalysis of chacha stream cipher. In: Nguyen, K., Yang, G., Guo, F., Susilo, W. (eds.) *ACISP 2022*. LNCS, vol. 13494, pp. 46-66, 2022. Springer International Publishing, Cham (2022). https://doi.org/10.1007/978-3-031-22301-3_3
- [20] Coutinho, M., Souza, T.C.N.: Improved linear approximations to ARX ciphers and attacks against chacha. In: Canteaut, A., Standaert, F.-X. (eds.) *EUROCRYPT 2021*. LNCS, vol. 12696, pp. 711-740. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-77870-5_25
- [21] Dey, S., Garai, H.K., Sarkar, S., Sharma, N.K.: Revamped differential-linear cryptanalysis on reduced round chacha. In: Dunkelman, O., Dziembowski, S. (eds.) *EUROCRYPT 2022*. LNCS, vol. 13277, pp. 86-114. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-07082-2_4
- [22] Dey, S., Garai, H. K., Maitra, S.: Cryptanalysis of Reduced Round ChaCha-New Attack & Deeper Analysis. *IACR Transactions on Symmetric Cryptology*, vol. 2023, no. 1, pp. 89-110, 2023. <https://doi.org/10.46586/tosc.v2023.i1.89-110>
- [23] Dey, S., Garai, H.K., Sarkar, S., Sharma, N.K.: Enhanced Differential-Linear Attacks on Reduced Round ChaCha. *IEEE Transactions on Information Theory*, vol. 69, no. 8, pp. 5318-5336, 2023. <https://doi.org/10.1109/TIT.2023.3269790>
- [24] Wang, S., Liu, M., Hou, S. and Lin, D.: Moving a Step of ChaCha in Syncopated Rhythm. In: H. Handschuh and A. Lysyanskaya (eds.) *CRYPTO 2023*. LNCS, vol. 14083, pp. 273-304. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-38548-3_10
- [25] E. Bellini, D. Gerault, J. Grados, R. H. Makarim, and T. Peyrin. Boosting differential-linear cryptanalysis of chacha7 with MILP. *IACR Transactions on Symmetric Cryptology*, vol. 2023, no. 2, pp. 189-223, 2023. <https://tosc.iacr.org/index.php/ToSC/article/view/10983>
- [26] Xu, Z., Xu, H., Tan, L., Qi, W.: Differential-Linear Cryptanalysis of Reduced Round ChaCha. *IACR Transactions on Symmetric Cryptology*, vol. 2024, no. 2, pp. 166-189, 2024. <https://doi.org/10.46586/tosc.v2024.i2.166-189>
- [27] Dey, S.: Advancing the idea of probabilistic neutral bits: first key recovery attack on 7.5 round ChaCha. *IEEE Transactions on Information Theory*, vol. 70, no. 8, pp. 6091-6106, 2024. <https://doi.org/10.1109/TIT.2024.3389874>
- [28] Sharma, N.K., Dey, S., Sarkar, S., Maitra, S.: On Improved Cryptanalytic Results Against ChaCha for Reduced Rounds ≥ 7 . In: Mukhopadhyay, S., Stănică, P. (eds.) *INDOCRYPT 2024*, LNCS, vol. 15496, pp. 29-52. Springer, Cham (2025). https://doi.org/10.1007/978-3-031-80311-6_2
- [29] Flórez-Gutiérrez, A., Todo, Y.: Improved Cryptanalysis of ChaCha: Beating PNBs with Bit Puncturing. In: Fehr, S., Fouque, P.A. (eds.) *EUROCRYPT 2025*, LNCS, vol. 15601, pp. 427-457. Springer, Cham (2025). https://doi.org/10.1007/978-3-031-91107-1_15
- [30] Flórez-Gutiérrez, A., Todo, Y.: Improving linear key recovery attacks using walsh spectrum puncturing. In: Joye, M., Leander, G. (eds.) *EUROCRYPT 2024*, LNCS, vol. 14651, pp. 187-216. Springer, Cham (2024). https://doi.org/10.1007/978-3-031-58716-0_7
- [31] Flórez-Gutiérrez, A., Todo, Y. (2026). Divide-and-Conquer Trail Enumeration Puncturing: Application to Salsa and ChaCha. In: Hanaoka, G., Yang, B.Y. (eds.) *Advances in Cryptology-ASIACRYPT 2025*. ASIACRYPT 2025. Lecture Notes in Computer Science, vol 16245. Springer, Singapore. https://doi.org/10.1007/978-981-95-5018-0_2
- [32] S. Sahoo, D. Chakraborty and S. Sarkar, "Chasing Shadows: Advancements in Differential-Linear Cryptanalysis for ChaCha," in *IEEE Transactions on Information Theory*, vol. 71, no. 8, pp. 6451-6469, Aug. 2025, <https://doi.org/10.1109/TIT.2025.3577784>.
- [33] Langford, S.K., Hellman, M.E.: Differential-linear cryptanalysis. In: Desmedt, Y.G. (eds.) *CRYPTO 1994*. LNCS, vol. 839, pp. 17-25. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-48658-5_3
- [34] Biham, E., Shamir, A.: Differential cryptanalysis of DES-like cryptosystems. In: Menezes, A.J., Vanstone, S.A. (eds.) *CRYPTO 1990*. LNCS, vol. 537, pp. 2-21. Springer, Heidelberg (1991). https://doi.org/10.1007/3-540-38424-3_1
- [35] Matsui, M.: Linear cryptanalysis method for DES cipher. In: Hellese, T. (eds.) *EUROCRYPT 1993*. LNCS, vol. 765, pp. 386-397. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-48285-7_33
- [36] Blondeau, C., Leander, G., Nyberg, K.: Differential-Linear Cryptanalysis Revisited. In: Cid, C., Rechberger, C. (eds.) *FSE 2014*. LNCS, vol. 8540, pp. 411-430, 2015. Springer International Publishing, Cham (2015). https://doi.org/10.1007/978-3-662-46706-0_21
- [37] Blondeau, C., Leander, G., Nyberg, K.: Differential-Linear Cryptanalysis Revisited. *Journal of Cryptology*, vol. 30, pp. 859-888, 2017. <https://doi.org/10.1007/s00145-016-9237-5>