Improved Key-Recovery Attack on ChaCha Using Carry-Lock Method

Anonymous Submission

Abstract. In this work, we introduce the *carry-lock* technique to enhance the probabilistic neutral bit-based differential attacks on ChaCha. Existing attacks rely on probabilistic neutral bits (PNBs) by partitioning key bits into significant bits and PNBs and recovering them in two stages. We observe that the correlation in these attacks is significantly influenced by carry propagation in the backward subtraction operation. The proposed *carry-lock* method restricts carry propagation in specific segments, effectively mimicking XOR behavior in those segments. By leveraging the *carry-lock* method, we first increase the count of PNBs and achieve the same correlation value for a PNB block as with the XOR operation in the key-stream generation equation. Secondly, this method introduces dependencies among significant key bits, reducing the search space in the first stage of the attack by limiting the number of possible key candidates. With these contributions, we present the first-ever attack on ChaCha7/128 and enhance the best-known attack on ChaCha7.5/256.

Keywords: ARX · Differential-linear attack · ChaCha · Carry-Lock · PNBs

1 Introduction

In the world of data security, encryption is the cornerstone of protecting sensitive information. Among the many encryption methods, stream ciphers stand out for their simplicity and speed. Unlike block ciphers, which encrypt data in fixed chunks, stream ciphers work by blending plaintext one bit or byte at a time with a pseudo-random *keystream*. This makes ARX ciphers particularly attractive for software implementations, offering high throughput with minimal resource requirements. Many modern stream ciphers follow the ARX design philosophy, which relies on three simple yet powerful operations: Addition modulo a power of 2 (denoted as $\boxplus$), bitwise Rotation (e.g., right rotation $\ggg$), and XOR (denoted as $\oplus$). These operations are not only easy to implement in software but also highly resistant to many cryptographic attacks.

The roots of ARX trace back to the 1980s, with the block cipher FEAL [SM87], the first to use this combination. However, ARX truly flourished in stream ciphers, particularly the Salsa and ChaCha families, designed by Daniel J. Bernstein. Introduced in 2007, Salsa was a fast yet secure encryption [Ber08b]. Its successor, ChaCha (2008), improved security by enhancing "diffusion"-a property that scrambles data thoroughly to hide patterns. It relies on a core function that processes a fixed-size block of 512 bits using rounds of transformations involving ARX operations. ChaCha's efficiency and robustness made it a popular replacement for the aging RC4 cipher in protocols like TLS (Transport Layer Security), which secures internet traffic. Today, ChaCha20, a variant using 20 rounds of ARX operations, is widely adopted. Combined with the Poly1305 authentication tool, it forms a secure, lightweight package for encryption in systems like the Linux kernel, Android, and cloud services. ChaCha operates using a series of simple arithmetic and bitwise operations that are highly efficient on modern processors. The key, nonce, and block counter are the inputs that ensure each keystream block is unique and secure.

ARX designs derive their security from the interaction of three word-wise operations:

modular addition, XOR, and rotation. Among these, modular addition is the only non-linear component over $\mathbb{F}_2$. Its non-linearity is not merely an increase in "algebraic complexity": the carry propagation couples bit positions in a data-dependent manner, so that the effect of an input difference or linear mask depends on intermediate carries. Rotations and XOR with constants (or round-dependent constants) provide diffusion and destroy structural symmetries (e.g., rotational relations), but they do not by themselves prevent attacks; rather, they ensure that differences and masks are rapidly spread across many bit positions and words, forcing an adversary to control or predict a large set of carry events.

Our Contribution

The current key-recovery attacks on ChaCha heavily rely on the concept of probabilistic neutral bits (PNBs), which form the basis of a meet-in-the-middle strategy. The attack is done in both forward and backward directions. In the forward direction, a differential-linear distinguisher is searched to trace how specific differences propagate and correlate with certain output bits. This correlation is called the forward correlation. In the backward direction, the attacker guesses the significant key bits and finds out if there is a correlation for the guess, which is called the backward correlation.

In this work, we focus on the backward direction of the attack by enhancing the correlations of the PNBs and reducing the search space for the significant key bits. The key contributions of our work, along with the organization of the paper, are outlined below on a section-by-section basis.

Background Material: Section 2 presents the design of the ChaCha cipher and the PNB-based differential-linear attack. Section 3 explores further advancements in this direction, along with the methodology for calculating data and time complexity. Given the context of this paper, in Subsection 3.1, we specifically review previous attack techniques that attempted to address the impact of carry propagation on PNB-based attacks.

Restricting carry propagation (carry-lock). Section 4 introduces our main tool, the *carry-lock* method, used in the backward evaluation of a PNB-based distinguisher. After guessing the significant key-dependent bits, the attacker completes a full state $\bar{X}$ by assigning random values to the remaining PNB positions. The backward computation starts from the observed keystream and evaluates inverse rounds on $Z \boxminus \bar{X}$, where $\boxminus$ is modular subtraction. A difficulty is that subtraction introduces data-dependent borrows, so the difference pattern in $Z \boxminus \bar{X}$ can contain additional bit differences compared to $Z \oplus \bar{X}$, which reduces the resulting correlation. The carry-lock method imposes simple conditions on selected keystream segments to prevent a borrow from entering or leaving those segments, forcing subtraction to match XOR locally: $(Z \boxminus \bar{X})[\mathcal{I}] = (Z \oplus \bar{X})[\mathcal{I}]$. Thus, on the PNB segment $\mathcal{I}$, we achieve the minimum number of differences (the XOR case), which improves the correlation ε_a.

Improving the correlation of the PNBs: In Section 5, we discuss how to employ the attack technique on the PNB blocks and improve the correlation and count of PNBs. We also draw a comparison between previous approaches in this direction and the *carry-lock* method, explaining that there is no carry propagation beyond the PNB block in our method.

Harmonizing Significant Bits: Section 6 explains how we can execute the idea of the *carry-lock* method on significant key bits to reduce the number of guesses in the recovery process. We remove the last XOR operation in the `quarterround` function of the last

Table 1: Complexities of Key Recovery Attacks on ChaCha and Our Result.

Key Size	Rounds	Data	Time Complexity	Reference
128	6.5	$2^{66.94}$	$2^{123.04}$	[DGSS22]
		$2^{66.29}$	$2^{121.40}$	[DGSS23]
		$2^{37.27}$	$2^{113.08}$	[Dey24]
	7	$2^{91.43}$	$2^{125.90}$	Subsection 7.2
256	6	2^{61}	2^{212}	[CSN21]
		$2^{41.47}$	$2^{99.48}$	[DGM23]
		2^{58}	$2^{77.4}$	[BBC+22]
		$2^{73.7}$	$2^{75.7}$	[WLHL23]
		2^{51}	$2^{61.4}$	[FGT25]
		$2^{55.7}$	$2^{57.4}$	[FGT25]
	7	2^{27}	2^{248}	[AFK+08]
		2^{96}	$2^{238.9}$	[Mai16]
		–	$2^{235.22}$	[DS17]
		$2^{48.83}$	$2^{230.86}$	[BLT20]
		$2^{90.20}$	$2^{221.95}$	[DGSS22]
		$2^{103.30}$	$2^{210.3}$	[WLHL23]
		$2^{93.79}$	$2^{192.89}$	[Dey24]
		$2^{102.63}$	$2^{189.7}$	[XXTQ24]
		$2^{101.09}$	$2^{178.12}$	[SDSM25]
		$2^{102.9}$	$2^{154.2}$	[FGT25]
		$2^{127.7}$	$2^{148.2}$	[FGT25]
	7.5	$2^{32.64}$	$2^{255.24}$	[Dey24]
		$2^{34.47}$	$2^{253.23}$	[SDSM25]
		$2^{127.1}$	$2^{250.2}$	[FGT25]
		$2^{95.46}$	$2^{246.29}$	Subsection 7.1

round of ChaCha, making it a reduced version. By structural analysis of the round function and the state, we observe that several bits of the reduced version are linear combinations of pairs of bits in the original version. We call these bits of each pair to be in harmony with each other. Instead of guessing all the significant bits individually, we can guess the linear combination of these pairs, which leads to a reduction in the number of guesses, resulting in a faster attack.

Application on Key Recovery: Section 7 elaborates on the key-recovery process and cryptanalysis of ChaCha7.5/256 and ChaCha7/128. Subsection 7.1 showcases the cryptanalysis against ChaCha7.5/256, with a detailed explanation of the attack procedure. In

Subsection 7.2, we provide the details of the first-ever cryptanalysis on ChaCha7/128. All the related source codes, including programs for PNB searching, correlation computation, carry-lock validation, and complexity calculations, are available at link to anonymous GitHub repository. The repository contains documented implementations that can be used to reproduce the experimental results presented in this work. Finally, Section 8 summarizes our findings and outlines potential future directions.

Table 2: Table of Notations.

Symbol	Meaning
X	State matrix
X_i	i-th word of X
ChaCha r/n	r round reduced version of ChaCha with n-bit key.
$[n_2 : n_1]$	Random block of length $(n_2 - n_1 + 1)$.
$x[n_2 : n_1]$	Consecutive bits starting from bit $x[n_1]$ to bit $x[n_2]$.
$[i_2 : i_1]$	PNB block of size $(i_2 - i_1 + 1)$.
$\|\|$	Concatenation of bit-strings
$\mathcal{ID}$	Input differential
$\Gamma_m^r[n]$	Round-r mask that selects the n-th bit of the m-th word
$\mathcal{OD}$	Output linear mask
$\boxplus$	Modular addition
$\boxminus$	Modular subtraction
$\oplus$	XOR operation

2 Preliminaries

2.1 Design of ChaCha

The ChaCha [Ber08a] family of stream ciphers uses a keystream generator that takes a 512-bit input and produces a 512-bit output. This input comprises a 128-bit constant c, a 256-bit secret key k, and a 128-bit initialization vector (IV) v. These values are divided into sixteen 32-bit words, with the IV v being the only part an adversary can directly control. Here, the value v (often denoted as the IV/counter field) is used as a *block counter* and/or nonce component: for each output block, the counter is incremented and the permutation is evaluated on a fresh state. This is what turns ChaCha into a stream cipher, since it generates a sequence of keystream blocks that are XORed with the plaintext.

These sixteen words are arranged into a 4×4 matrix X, known as the initial state. This matrix serves as the starting point for the ChaCha round function, which repeatedly applies a series of nonlinear operations to produce the final keystream output.

$$X = \begin{pmatrix} X_0 & X_1 & X_2 & X_3 \\ X_4 & X_5 & X_6 & X_7 \\ X_8 & X_9 & X_{10} & X_{11} \\ X_{12} & X_{13} & X_{14} & X_{15} \end{pmatrix} = \begin{pmatrix} c_0 & c_1 & c_2 & c_3 \\ k_0 & k_1 & k_2 & k_3 \\ k_4 & k_5 & k_6 & k_7 \\ v_0 & v_1 & v_2 & v_3 \end{pmatrix}.$$

ChaCha also has a 128-bit key version, where the 128-bit key occupies the second row and is *copied* into the third row (i.e., the second and third rows are identical).

The initial state goes through alternating odd and even ChaCha rounds, starting with the odd round, until all rounds are covered. A state after r rounds is denoted by X^r and is generated by updating X^{r-1}. Let us now describe the ChaCha round function.

Full Round Function: The ChaCha full round function is made up of four *parallel* applications of the `quarterround` function. The `quarterround` function takes four words (a, b, c, d) and updates them to (a'', b'', c'', d'') using the following equations,

$$\begin{aligned} a' &= a \boxplus b; & d' &= ((d \oplus a') \lll 16); \\ c' &= c \boxplus d'; & b' &= ((b \oplus c') \lll 12); \\ a'' &= a' \boxplus b'; & d'' &= ((d' \oplus a'') \lll 8); \\ c'' &= c' \boxplus d''; & b'' &= ((b' \oplus c'') \lll 7); \end{aligned} \tag{1}$$

In each full round, these four `quarterround` instances act on disjoint word quadruples (columns or diagonals), so they can be viewed as being applied in parallel.

Odd Round: An odd-numbered ChaCha round transforms the state X^{r-1} into X^r by applying updates to the columns of the state, as defined below:

$$\texttt{quarterround}(X_0^{r-1}, X_4^{r-1}, X_8^{r-1}, X_{12}^{r-1}), \texttt{quarterround}(X_1^{r-1}, X_5^{r-1}, X_9^{r-1}, X_{13}^{r-1}),$$
$$\texttt{quarterround}(X_2^{r-1}, X_6^{r-1}, X_{10}^{r-1}, X_{14}^{r-1}), \texttt{quarterround}(X_3^{r-1}, X_7^{r-1}, X_{11}^{r-1}, X_{15}^{r-1}).$$

Even Round: On the other hand, an even ChaCha round updates a state by updating the diagonals of the state as follows:

$$\texttt{quarterround}(X_0^{r-1}, X_5^{r-1}, X_{10}^{r-1}, X_{15}^{r-1}), \texttt{quarterround}(X_1^{r-1}, X_6^{r-1}, X_{11}^{r-1}, X_{12}^{r-1}),$$
$$\texttt{quarterround}(X_2^{r-1}, X_7^{r-1}, X_8^{r-1}, X_{13}^{r-1}), \texttt{quarterround}(X_3^{r-1}, X_4^{r-1}, X_9^{r-1}, X_{14}^{r-1}).$$

After completing all the rounds for an R round ChaCha, we get the state X^R, which is then added to the initial state X word by word. Note that here, addition is modular addition. The resulting state after this addition yields the keystream Z,

$$Z = X \boxplus X^R \tag{2}$$

We denote a R round k-bit ChaCha cipher as ChaCha $R/\ k$.

It is worth mentioning that the equations of the `quarterround` are reversible. We can get back (a, b, c, d) from (a'', b'', c'', d'') using the following equations:

$$\begin{aligned} b' &= (b'' \ggg 7) \oplus c''; & c' &= c'' \boxminus d''; \\ d' &= (d' \ggg 8) \oplus a''; & a' &= a'' \boxminus b'; \\ b &= (b' \ggg 12) \oplus c'; & c &= c' \boxminus d'; \\ d &= (d \ggg 16) \oplus a'; & a &= a' \boxminus b; \end{aligned} \tag{3}$$

Now from Equation 2, we can easily reach out to the state X^{-s} in the reverse direction by calculating

$$(Z \boxminus X)^{-(R-s)},$$

In general, a state after r backward rounds is denoted by X^{-r}.

Half Round Function: The ChaCha half round function is made up of four applications of `half quarterround` function. The `half quarterround` function takes four words (a, b, c, d) and updates them to (a', b', c', d') using the following equations,

$$\begin{aligned} a' &= a \boxplus b; & d' &= ((d \oplus a') \lll 16); \\ c' &= c \boxplus d'; & b' &= ((b \oplus c') \lll 12). \end{aligned} \tag{4}$$

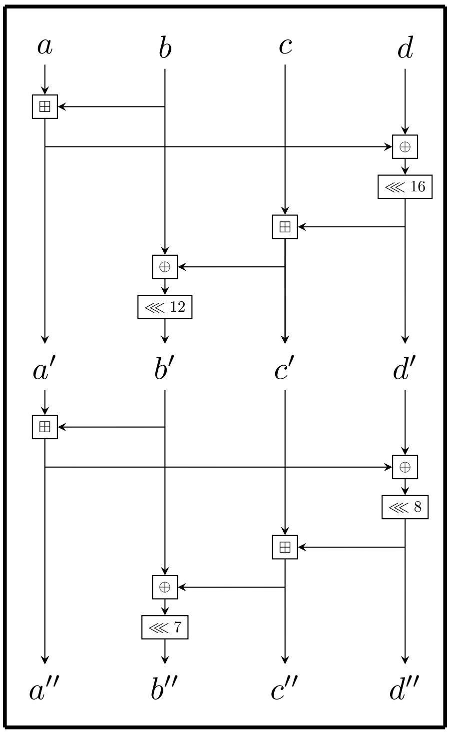


Figure 1: One `quarterround` function of ChaCha round function.

2.2 Existing Attacks

The cryptanalysis of the ChaCha cipher family employs the differential-linear attack framework developed by Langford and Hellman [LH94] in 1994 to analyze DES. The key-recovery attack on the cipher is based on the 'Probabilistic Neutral Bits' (PNB) technique given by Aumasson *et al.* [AFK+08] in FSE 2008. This PNB-based attack is built upon a differential-linear distinguisher.

2.2.1 Differential-Linear Distinguisher:

Constructing an r-round differential-linear distinguisher can be described by decomposing the r-round ChaCha permutation E into three parts

$$E \;=\; E_2 \circ E_m \circ E_1,$$

where E_1, E_m, and E_2 consist of r_1, r_2, and r_3 rounds, respectively, with $r = r_1 + r_2 + r_3$. For any bit-mask $\Gamma \in \{0,1\}^n$ and state $Y \in \{0,1\}^n$, we use the standard inner product

$$\langle \Gamma, Y \rangle \;:=\; \bigoplus_{i=0}^{n-1} \Gamma_i Y_i,$$

which selects the parity of the bits of Y indicated by Γ.

Differential part (E_1). Assume there exists a differential $(\Delta_{\text{in}}, \Delta_m)$ through E_1 with probability p, i.e.,

$$\Pr_X[\, E_1(X) \oplus E_1(X \oplus \Delta_{\text{in}}) = \Delta_m \,] = p.$$

We call a pair $(X, X \oplus \Delta_{\text{in}})$ a *right pair* for E_1 if it satisfies the event inside the probability. In the ChaCha setting, X is sampled by varying the nonce and block counter (equivalently, the IV/counter field); in our attack instantiation we fix one 32-bit word and vary the remaining 96 bits, yielding 2^{96} possible keystream blocks.

Differential-linear part (E_m). Let Γ_m be a nonzero mask. Suppose (Δ_m, Γ_m) forms a differential-linear distinguisher for E_m with correlation ε'_d, meaning

$$2 \Pr_U[\, \langle \Gamma_m,\, E_m(U) \oplus E_m(U \oplus \Delta_m) \rangle = 0 \,] - 1 \;=\; \varepsilon'_d.$$

Assuming that conditioning on the E_1 right-pair event does not significantly change the correlation behavior of this distinguisher for E_m, the composition $E_m \circ E_1$ yields

$$2 \Pr_X[\, \langle \Gamma_m,\, (E_m \circ E_1)(X) \oplus (E_m \circ E_1)(X \oplus \Delta_{\text{in}}) \rangle = 0 \,] - 1 \;=\; p\, \varepsilon'_d.$$

Equivalently, conditioned on right pairs, the correlation through E_m remains ε'_d, while unconditionally it is scaled by the right-pair probability p.

Linear part (E_2). Finally, assume there is a linear approximation $(\Gamma_m, \Gamma_{\text{out}})$ for E_2 with correlation ε_l, i.e.,

$$2 \Pr_V[\, \langle \Gamma_m, V \rangle \oplus \langle \Gamma_{\text{out}}, E_2(V) \rangle = 0 \,] - 1 \;=\; \varepsilon_l.$$

Applying this approximation independently to both branches of the differential pair contributes a factor ε_l^2.

Combined distinguisher. Combining the above components yields an r-round differential-linear distinguisher $(\Delta_{\text{in}}, \Gamma_{\text{out}})$ for ChaCha with correlation

$$2 \Pr_X[\, \langle \Gamma_{\text{out}}, E(X) \oplus E(X \oplus \Delta_{\text{in}}) \rangle = 0 \,] - 1 \;=\; p\, \varepsilon'_d\, \varepsilon_l^2,$$

and we denote the resulting (unconditional) correlation by $\varepsilon_d := p\, \varepsilon'_d\, \varepsilon_l^2$.

2.2.2 PNB Based Key-Recovery:

In this section, we describe the PNB-based key recovery for full R-round ChaCha. First, in the offline phase, the attacker collects the PNBs with good backward correlation. Next, with the help of these PNBs, the attacker in the online phase recovers the key.

- **Offline Phase**
  - ➢ **PNB Filtration**: Suppose we are given an r-round distinguisher $(\Delta_{\text{in}}, \Gamma_{\text{out}})$ with correlation ε_d. We generate input pairs $(X, X' = X \oplus \Delta_{\text{in}})$ and collect the corresponding keystream pairs (Z, Z') after R rounds of ChaCha.

    We define a function f that takes (X, Z, Z') as input and returns

    $$f(X, Z, Z') = \Gamma_{\text{out}} \left((Z \boxminus X)^{-(R-r)} \oplus (Z' \boxminus X')^{-(R-r)} \right).$$

    By construction, this function recovers the distinguisher output:

    $$f(X, Z, Z') = \Gamma_{\text{out}} \left((X)^r \oplus (X')^r \right).$$

    Now we flip the i-th key bit in X, resulting in a new state pair $(\tilde{X}, \tilde{X}')$. Using these modified states, we compute

    $$S = (Z \boxminus \tilde{X})^{-(R-r)}, \quad S' = (Z' \boxminus \tilde{X}')^{-(R-r)}.$$

Over all such input pairs (X, X'), we observe that $\Gamma_{\text{out}}((X)^r \oplus (X')^r)$ can be approximated by $\Gamma_{\text{out}}(S \oplus S')$. The quality of this approximation depends on the i-th key bit and is quantified by the correlation γ_i, defined as:

$$\gamma_i = 2 \Pr_X [\Gamma_{\text{out}}(S \oplus S') = \Gamma_{\text{out}}((X)^r \oplus (X')^r)] - 1.$$

If the correlation γ_i exceeds a fixed threshold γ, we classify the i-th key bit as a *probabilistically neutral bit* (PNB).

➢ **Backward correlation**:

Once we have a list of sufficient PNBs, we start with a sufficient number of initial state pairs $(X,\ X')$ (varying the IV and key) and collect all the corresponding keystream pairs $(Z,\ Z')$. We assign random values to the PNBs and keep the rest of the bits unchanged, as they are in X and X'. Consequently we get another pair of states $(\bar{X},\ \bar{X}')$ and we get the states $Y = (Z \boxminus \bar{X})^{-(R-r)}$, $Y' = (Z' \boxminus \bar{X}')^{-(R-r)}$. Now ε_a is calculated as

$$2 \Pr_X \left[\Gamma_{\text{out}}(Y \oplus Y') = \Gamma_{\text{out}}((X)^r \oplus (X')^r)\right] - 1 = \varepsilon_a.$$

Here we define another function $g(\bar{X}, Z, Z') = \Gamma_r(Y \oplus Y')$, so ε_a is the correlation of g with f. This g is generally mentioned as the PNB approximating function.

- **Online Phase**

➢ **Key Recovery**: Next in the key-recovery phase, we guess the significant key-bits, i.e., the non-PNBs first. First, we select a sufficient number of pairs of IVs, which form the pair of initial states (X, X') in the online mode, along with the unknown key. We then collect the corresponding keystreams Z, Z'. Now, for an initial state X, we guess the non-PNBs, put random values in the PNBs, and calculate

$$\Pr_X \left[\Gamma_r(Y \oplus Y') = 0 | X \oplus X' = \Delta_0\right].$$

If the guess is correct, we have the probability $0.5 \times (\varepsilon_d \varepsilon_a + 1)$; otherwise, the probability is close to 0.5 for a wrong guess. Basically, a threshold (T) is calculated based on the probability, and if the number of $(X,\ X')$ pairs for which $\Gamma_r(Y \oplus Y') = 0$ holds crosses that threshold T, we say that the guess for non-PNBs is correct.

After correctly guessing the non-PNBs, the PNBs are searched exhaustively.

3 Advancements in the Cryptanalysis Techniques

In this section, we list down the major works that influenced the cryptanalysis of the ChaCha family of ciphers and hence turn out to be a stepping stone to introduce novel techniques based on it. We also discuss the computation of data and time complexity values proposed in the recent work by Dey [Dey24] and some modifications done by Sharma *et al.* [SDSM25].

➢ In FSE 2008, Aumasson *et al.* [AFK+08] introduced a 3-round differential distinguisher for ChaCha and introduced the PNB-based attack methodology, setting a precedent for the analysis of the Salsa and ChaCha cipher families [AFK+08]. Building on this, Shi *et al.* leveraged the concept of *Column Chaining Distinguisher (CCD)*, further enhancing the PNB-based cryptanalysis of ChaCha [SZFW13].

- In 2015, Maitra refined the distinguisher for the Salsa and ChaCha cipher families by introducing the innovative *chosen IV* concept, advancing the cryptanalytic capabilities for these ciphers [Mai16]. Choudhuri *et al.* achieved a major milestone in 2016 by presenting the first-ever 5-round distinguisher for ChaCha, alongside a 6-round distinguisher for Salsa [CM17]. Subsequently, Dey *et al.* enhanced the PNB algorithm, identifying a more effective set of PNBs, which significantly improved the attack performance for both Salsa and ChaCha [DS17].

- After nearly a decade of progress, Beierle *et al.* at CRYPTO 2020 improved the distinguisher for ChaCha by half rounds and introduced a 6-round differential-linear distinguisher [BLT20]. They employed the Fast Walsh-Hadamard Transform (FWHT) to recover the key for ChaCha6/256, marking a notable advance in key recovery techniques.

- In EUROCRYPT 2022, Dey *et al.* made a major leap by optimizing the PNB searching algorithm [DGSS22]. They introduced memory and non-memory partition techniques for key bits, significantly improving the key-recovery attack complexity. Furthermore, they demonstrated that using patterned values in PNB positions instead of random bits enhances correlation, thereby reducing the attack complexity.

- At FSE 2023, Dey *et al.* utilized multiple $(\mathcal{ID}, \mathcal{OD})$ pairs to launch a more efficient attack on ChaCha6/256 [DGM23]. In the same year, at CRYPTO, Wang *et al.* introduced the 'syncopation technique', a novel method where conditions were imposed on bits that improved correlation of the PNB-approximating function, reducing attack complexity [WLHL23]. They also analyzed a modified ChaCha7.5/256, where the last two operations in the `quarterround` are omitted, adding further depth to the cryptanalysis of ChaCha. More recently, Sahoo *et al.* (2025) [SCS25] showed how to exploit data that was previously treated as unusable, thereby reducing the overall data complexity even under the imposed conditions, and used this idea to mount improved attacks on ChaCha.

- Bellini *et al.* discovered a new 4-round differential-linear distinguisher for ChaCha, enabling successful attacks on ChaCha7/256 and ChaCha7.25/256 [BGG+23]. This work was further refined in FSE 2024, where Xu *et al.* improved the same 4-round distinguisher by identifying additional intermediate linear masks [XXTQ24].

- In 2024, Dey achieved another significant breakthrough by advancing the attack on ChaCha7/256 through the integration of multi-bit combinations of the differential-linear distinguisher. This approach led to the first-ever attack on ChaCha7.5/256, pushing the boundaries of cryptanalysis for this cipher family [Dey24]. In the work of Dey [Dey24], the author mentioned that the formulation of the data complexity can also be modified by reducing the error probability. The formulation of data complexity is explained in Subsection 3.2. In 2024, Sharma *et al.* [SDSM25] improved the PNB algorithm and slightly modified the computation of time complexity value, hence providing the best-ever attack on ChaCha to date.

- In 2025, Flòrez-Gutiérrez and Todo came up with a new approach called *bit puncturing* which bypassed the PNB-based attack technique [FGT25]. They introduced the first theory-driven key-recovery method that analytically exploits ChaCha's ARX carries instead of relying on empirical Probabilistic Neutral Bits. The new bit-puncturing approach cuts the record complexities for 6, 7, and 7.5-round ChaCha-*e.g.*, the 7-round attack is 2^{40} times faster than the prior state of the art. It delivers the first successful 7.5-round attack with a measurable advantage over exhaustive search, providing an alternative to PNB-based techniques which does not rely on experimentally determined correlations.

3.1 Revisiting the Previous Works on the Backward correlation

In this part, we discuss the ideas of Aumasson *et al.* [AFK+08], Dey *et al.* [DGSS23], and Wang *et al.* [WLHL23] in detail to draw a comparison with our *carry-lock* method introduced in Section 4.

3.1.1 Initial Approach:

In the approach of Aumasson *et al.* [AFK+08], which is discussed thoroughly in Subsubsection 2.2.2, no special initiative was taken to reduce the carry propagation during the subtraction operation ($Z \boxminus X$). The authors assumed that any arbitrary value assigned to those bits would have the same effect, which was reflected in their statement "non-significant key bits being set to a fixed value (e.g., all zero)". Even in 2020, Beierle *et al.* [BLT20] and Coutinho *et al.* [CSN21] assigned zero value to each PNB.

3.1.2 Idea of Assigning Values to PNBs:

In 2022, Dey *et al.* [DGSS23] analyzed the impact of carry propagation theoretically and found that the value assigned to the PNBs affects the probability of difference propagation through carry, during the subtraction operation. They studied three patterns: All zero pattern, Random pattern, and `1` followed by all `0`'s pattern. They concluded that the `1` followed by all `0`'s pattern, i.e., in the PNB block of X_k, after value assignment, $\bar{X}_k[i_2] = \texttt{1}$ and $\bar{X}_k[i_2 - 1 : i_1] = \texttt{000} \dots \texttt{0}$ produces higher backward correlation as compared to the other two patterns.

3.1.3 Syncopation Technique:

In Crypto 2023, Wang *et al.* [WLHL23] proposed a new idea called the syncopation technique, which helps in improving attacks on the ARX ciphers. This technique addresses the challenge of finding a large number of Probabilistic Neutral Bits (PNBs) that are associated with a high correlation, a task that is inherently difficult due to the inverse relationship between the number of PNBs and their correlation strength. Traditional methods of obtaining the PNBs, such as the naive threshold rule and greedy methods, treat the cipher as a black box and do not use the ARX structure's properties. Inspired by the partitioning technique [Leu16], the syncopation technique aims to utilize the ARX structure differently.

3.2 Complexity of the Attack

The complexity analysis of the existing PNB-based differential-linear attack on ChaCha was initially given by [AFK+08]. In their work, the median of experimentally observed correlation values was used as a parameter in the data and time complexity calculations, ensuring that the attack would succeed for at least 50% of the keys. Subsequent works adopted a similar methodology but often used the mean of the correlation values instead. Since the mean and median values are typically close in practice, the resulting complexity estimates can be regarded as representing the average-case scenario. Recently, [Dey24] presented an attack structure involving multi-bit output differences and provided a modified formula for time complexity under that attack model. In our work, we follow the same structural approach as [Dey24]. So, at first, we discuss briefly the attack model and the corresponding complexity formula.

In [Dey24], Dey obtained the PNBs corresponding to the multi-bit output difference as well as the $k(> 1)$ bits of output difference. This is denoted as $\Delta_{\mathcal{OD}} = \bigoplus_{i=1}^{k} \Delta_{\mathcal{OD}_i}$. Here $\Delta_{\mathcal{OD}}$ denotes the multi-bit output difference, which can be written as the linear combination of k output difference bits $\Delta_{\mathcal{OD}_i}$'s. The PNBs are first obtained for the multi-bit output difference. Then, for each output difference bit $\mathcal{OD}_i$, the PNBs are noted after removing the PNBs already obtained for the multi-bit output $\bigoplus_{i=1}^{k} \Delta_{\mathcal{OD}_i}$, because the PNBs for the multi-bit output are already the PNBs for each output difference bit. This relation is explained in detail in [Dey24, Piling Up Lemma, Section IV]. After obtaining the set of probabilistic neutral bits for the linear combination of k output difference bits $\Delta_{\mathcal{OD}_i}$'s, the remaining bits are considered as significant bits, and let S be the set of such bits.

To recover the significant key bits in set S, the attacker assigns arbitrary values to the PNBs, guesses the significant key bits, and obtains two states $\tilde{X}$ and $\tilde{X}'$. After applying the reverse round function on $Z \boxminus \tilde{X}$ and $Z' \boxminus \tilde{X}'$, the matrices $\tilde{Y}$ and $\tilde{Y}'$ are obtained. The Backward correlation value is obtained using the same procedure as explained in Subsubsection 2.2.2. The backward correlation value is denoted by ε_a.

Similarly, for each output difference bit $\mathcal{OD}_i$, the set of significant bits is obtained. For the i-th bit of $\mathcal{OD}_i$, the $\tilde{X}_i$ and $\tilde{X}'_i$ are considered, and applying the reverse round after assigning arbitrary values to the PNBs, guess the significant key bits. The backward correlation value is observed and is denoted by ε_i. There exists a relation between the backward correlation value of the $\Delta_{\mathcal{OD}_i}$'s and each $\mathcal{OD}_i$ bit. As mentioned in the key recovery part, the correlation between the forward correlation ε_d and ε_a exists. After using the key-recovery process for each output difference bit, there exists the correlation value ε as mentioned in Proposition 1 [Dey24], which is given by

$$\varepsilon = \varepsilon_d \cdot \varepsilon_a \cdot \bigoplus_{i=1}^{k} \varepsilon_i$$

Starting from the work of [AFK+08], we cast the key-bit recovery step as a binary hypothesis test on a candidate guess $\widehat{S}$ for the s significant key bits.

$$H_0 : \ \widehat{S} \neq S \quad \text{(the guessed significant key bits are incorrect)},$$
$$H_1 : \ \widehat{S} = S \quad \text{(the guessed significant key bits are correct)}.$$

Since $|S| = s$, there are 2^s possible guesses for $\widehat{S}$, of which exactly one satisfies H_1 and the remaining $2^s - 1$ satisfy H_0. Given a fixed decision rule based on the measured correlation (or test statistic), two error events can occur:

1. **Non-detection :** the correct guess $\widehat{S} = S$ satisfies H_1, but the test decides H_0 (i.e., the correct significant-key value is not detected and the attack fails). The probability of this event is $\Pr_{\text{nd}}$.
2. **False alarm:** an incorrect guess $\widehat{S} \neq S$ satisfies H_0, but the test decides H_1 (i.e., the attack accepts a wrong significant-key value due to an unusually large measured correlation). The probability of this event is $\Pr_{\text{fa}}$.

In our analysis, we require $\Pr_{\text{fa}} \leqslant 2^{-\alpha}$ and denote the non-detection probability by $\Pr_{\text{nd}}$. Using the Neyman–Pearson decision framework, the required number of samples N to achieve these bounds is approximated by

$$N \approx \left(\frac{\sqrt{\alpha \ln 4} - \Phi^{-1}(\Pr_{\text{nd}}) \sqrt{1 - \varepsilon^2}}{\varepsilon} \right)^2 . \tag{5}$$

In [Dey24], a formula for computing the attack complexity was proposed, which was subsequently refined by Sharma *et al.* [SDSM25]. The time complexity is the sum of the complexities of two steps. At first, the attaker produces the lists corresponding to each output difference bits. If for $\mathcal{OD}_i$, we have m_i significant bits, then there are 2^{m_i} possible guesses for those bits. And for each such guess, attacker needs to prepare and store a tuple of length N, and to achieve each term of this sequence, the attacker has to apply the reverse round function by $R-r$ rounds. Therefore, taking that entire set of operation as unit, the complexity to prepare each table requires $2^{m_i} \cdot N$ unit time. In the second step, the attacker makes a guess of entire set of significant bits and finds its projection on each of the k sorted lists, picks the corresponding N-bit tuples and XORs, which leads to $N(k-1)$ XOR operations for each guess. For 2^m guesses, there are $(k-1)\cdot 2^m \cdot N$ XOR operations in total. Authors show that 1 such XOR operation is equivalent to $\frac{1}{2^{11}\cdot(R-r)}$ fraction of our declared unit of complexity. So, the complexity to recover significant key bits is $2^m \cdot N \cdot \frac{k-1}{2^{11}\cdot(R-r)}$. Then, adding the extra computation of $2^{256} \cdot \Pr_{fa}$ performed because of the false alarm error and 2^{256-m} in the final step to recover the PNBs, the resulting expression is as follows:

$$C = \sum_{i=1}^{k} 2^{m_i} \cdot N + 2^m \cdot N \times \frac{k-1}{2^{11} \times (R-r)} + 2^{256-\alpha} + 2^{256-m} \tag{6}$$

Here, m is the dimension of the full non-PNB guess space $\mathcal{G}$, i.e., m denotes the number of non-PNBs for the multi-bit output difference position, and m_i denotes the non-PNBs for the i-th bit of the multi-bit output differential. For a detailed explanation of how the complexity formula is derived, see [Dey24].

4 Introducing the Carry-Lock Method

In ARX-based cryptographic designs, how a difference spreads depends very much on which operation is involved. Some operations are "tame" in the sense that they only reflect the bits that are actually changed, while others can amplify a tiny local change into a wider disturbance. The XOR operation is linear over $\mathbb{F}_2$ and has no extra dependencies. Modular addition/subtraction belongs to the second group because of the carry; a change in a low bit can affect high bits. To better understand this, let us consider two n-bit words z and y and the modular subtraction and XOR operation between the two words,

$$x = z \boxminus y, \quad w = z \oplus y.$$

Suppose we modify a bit-segment $\mathcal{I}$ of y by assigning random values, leading to a new value $\bar{y}$. Then performing the same operations as before with $\bar{y}$ instead of y, we obtain:

$$\bar{x} = z \boxminus \bar{y}, \quad \bar{w} = z \oplus \bar{y}.$$

For our attack, we have to improve the correlation of the PNB approximating function, where we are interested in minimizing the carry effect beyond the PNB segment.

If we compare $(w \oplus \bar{w})$ and $(x \oplus \bar{x})$, it is very obvious that XOR exhibits changes only in segment $\mathcal{I}$, while for the case of subtraction, the changes may propagate beyond $\mathcal{I}$. To control this propagation, we impose some conditions on z. Specifically, we establish sufficient conditions ensuring that any modification within the segment $\mathcal{I}$ remains localized, with no carry propagation beyond its boundaries. This confinement of the carry effect motivates the terminology *carry-lock*. This is formalized in the following result:

Lemma 1. *Let $z, y \in \mathbb{F}_2^n$, and let $\bar{y}$ be derived from y by arbitrarily changing the bit-segment $[n_2 : n_1]$. If the following conditions are satisfied:*

(a) $z[n_1 - 1 : 0] \geqslant y[n_1 - 1 : 0]$.

(b) $z[t] \geqslant y[t]$ *and* $z[t] \geqslant \bar{y}[t]$, $\forall\ t \in \{n_1, n_1 + 1, \ldots, n_2\}$.

Then the following hold:

(1) $(z \boxminus y)[n - 1 : n_2 + 1] = (z \boxminus \bar{y})[n - 1 : n_2 + 1]$.

(2) $(z \boxminus y)[n_2 : n_1] = (z \oplus y)[n_2 : n_1]$.

Proof. For any $t \in \{0, 1, \ldots, n - 1\}$, the t-th bit of $(z \boxminus y)$ is given by

$$(z \boxminus y)[t] = \begin{cases} z[t] \oplus y[t], & \text{if } z[t-1:0] \geqslant y[t-1:0], \\ z[t] \oplus y[t] \oplus 1, & \text{if } z[t-1:0] < y[t-1:0]. \end{cases}$$

Based on conditions (a) and (b), we know that for any $t \in \{n_1, n_1 + 1, \ldots, n_2\}$,

$$z[t - 1 : 0] \geqslant y[t - 1 : 0].$$

Thus, the t-th bit of $(z \boxminus y)[t]$ simplifies to:

$$(z \boxminus y)[t] = z[t] \oplus y[t].$$

For $t > n_2$, the values $(z \boxminus y)[t]$ and $(z \boxminus \bar{y})[t]$ can only differ if one of $y[t - 1 : 0]$ or $\bar{y}[t - 1 : 0]$ is greater than $z[t - 1 : 0]$ while the other is smaller.

However, from our assumptions,

$$y[n_2 : 0] < z[n_2 : 0] \quad \text{and} \quad \bar{y}[n_2 : 0] < z[n_2 : 0].$$

Combining this with the fact that $y[n - 1 : n_2 + 1] = \bar{y}[n - 1 : n_2 + 1]$, we conclude that $z[t - 1 : 0] - y[t - 1 : 0]$ and $z[t - 1 : 0] - \bar{y}[t - 1 : 0]$ have the same parity.

This completes the proof. □

Example 1. Let us take $n = 16$, $i = 4$, and a bit-segment of length 3, i.e., bit segment $[6 : 4]$ of y is arbitrarily modified to generate $\bar{y}$.

Now choose z and y from $\mathbb{F}_2^{16}$ such that

1. $z[3 : 0] \geqslant y[3 : 0]$.
2. $z[t] \geqslant y[t], z[t] \geqslant \bar{y}[t]\ \forall\ t \in \{4, 5, 6\}$.

Then according to Lemma 1, the value $(z \boxminus y)$ and $(z \boxminus \bar{y})$ is same for the block $[15 : 7]$, i.e., $(z \boxminus y)[15 : 7] = (z \boxminus \bar{y})[15 : 7]$. This implies that if we apply these conditions, then there is no carry propagation in the bits $[15 : 7]$ even if we arbitrarily change y. Also from Lemma 1, $(z \boxminus y)[6 : 4] = (z \oplus y)[6 : 4]$.

5 Application of Carry-Lock Method on PNB Blocks

Let us consider two initial states, X and X' with the desired input difference. Consider a PNB block of size $(i_2 - i_1 + 1)$, represented as $[i_2 : i_1]$ in a key word X_w of X. Since the keywords are the same in the two states, if we denote the corresponding keyword of X' as X'_w, we have $X_w = X'_w$. Let the corresponding words of Z and Z' be Z_w and Z'_w, respectively. Now we aim to execute the *carry-lock* method on the PNB block $[i_2 : i_1]$ of both the subtraction operations $Z_w \boxminus X_w$ and $Z'_w \boxminus X'_w$. Next, let us investigate how the attacker can choose the Z, Z', from the available data, based on his guessed value of the key bits, such that for the PNB block $[i_2 : i_1]$, the carry-lock criteria given in Lemma 1 are satisfied.

5.1 Criteria to Choose Data in Order to Execute Carry-Lock Method

The application of the carry-lock method requires that the two conditions established in Lemma 1 be satisfied for both subtraction operations $Z_w \boxminus X_w$ and $Z'_w \boxminus X'_w$. Crucially, we aim to enforce these conditions specifically in the case where the attacker's guess of the significant key bits is correct. But, during the recovery of significant bits, the attacker does not know beforehand the actual values for the block $[i_2 : i_1]$ of X_w. So, the attacker needs to choose the Z, Z' based on his guessed values of keys. We denote the guessed values of X_w, X'_w by $\widehat{X_w}$, $\widehat{X'_w}$. Let us examine what criteria between $Z_w, Z'_w, \widehat{X_w}$ ensure that when the guess of the significant part is correct, it automatically satisfies the *carry-lock* conditions of Lemma 1.

Given that $X_w[i_1 - 1, 0]$ contains significant bits, if the guess is correct, the attacker knows the block $X_w[i_1 - 1 : 0]$. Therefore, in order to satisfy the first condition of Lemma 1 for the PNB block $[i_2 : i_1]$, while selecting the data (output keystream) to be used in the attack, the attacker can use the same condition on $Z_w, \widehat{X}_w$ and $Z'_w, \widehat{X'}_w$.

To satisfy the second condition of Lemma 1, we select Z, Z' in which all the bits in $[i_2 : i_1]$, so that whatever be the actual values of the corresponding bits of X_w, the condition $Z_w[t] \geqslant X_w[t]$ is satisfied. Below, we write the two properties of the criteria formally.

Criteria 1: ***Criteria to Execute Carry-Lock on PNB Blocks***

Consider a keyword X_w with a PNB block $[i_2 : i_1]$. Then, for any guess $\widehat{X_w}$, $\widehat{X'_w}$, the attacker uses the output keystream pairs (Z, Z') which satisfy the following conditions:

1. The block $[i_1 - 1 : 0]$ of the keystream must be greater than or equal to the corresponding block of the guessed value $\widehat{X_w}[i_1 - 1 : 0]$. i.e., $Z_w[i_1 - 1 : 0] \geqslant \widehat{X_w}[i_1 - 1 : 0]$ and $Z'_w[i_1 - 1 : 0] \geqslant \widehat{X'_w}[i_1 - 1 : 0]$.
2. The PNB block itself must be fully set to '1' in the keystream: $Z_w[i_2 : i_1] = \texttt{0b11}\ldots\texttt{1}$ and $Z'_w[i_2 : i_1] = \texttt{0b11}\ldots\texttt{1}$.

5.2 Comparison with Previous Attack Approaches

The *carry-lock* condition provided a structured way to confine modifications within a specific segment. Here, we provide a comparison of our method with the previous works in this direction to reduce the probability of difference propagation ([WLHL23], [DGSS23]), which we have discussed in Subsection 3.1.

Consider two randomly chosen n-bit numbers z and y. Suppose $\bar{y}$ is obtained by assigning arbitrary values to y of a block at position $[n_2 : n_1]$. Now, we observe the differences between $z - y$ and $z - \bar{y}$. The approach of [DGSS23] assigns specific values (10..00) to the block $[n_2 : n_1]$. In comparison to the method proposed in [AFK+08], this approach yields a lower probability of difference propagation beyond the n_2-th bit position. The syncopation technique is more effective, which, by choosing z with specific values on the $(n_2 + 1)$-th bit, ensures that the difference does not propagate beyond the $(n_2 + 1)$-th bit, i.e., the propagation is at most one bit.

In Table 3, we provide the comparison of the approach of [AFK+08], [DGSS23], and [WLHL23] with our approach.

We define a random variable X as the number of bits beyond the $[n_2 : n_1]$ block where the difference between $z - y$ and $z - \bar{y}$ propagates, *i.e.*, where these two quantities differ. For each approach, we empirically estimate the expected value $\mathbb{E}[X]$ by computing the average

Table 3: Experimentally observed average propagation distance (in bits) for prior work versus *carry-lock* method.

Technique	**Expected Propagation**
Classical PNBs (Aumasson *et al.* [AFK+08])	0.33
Pattern technique (Dey *et al.* [Dey24])	0.25
Syncopation technique (Wang *et al.* [WLHL23])	0.17
This Work	**0.00**

propagation distance across 2^{20} randomly chosen samples:

$$\mathbb{E}[X] \approx \frac{1}{2^{20}} \sum_{i=1}^{2^{20}} X_i,$$

where X_i denotes the number of post-n_2 differing bits in the i-th sample. This provides a practical approximation of the expected propagation distance for each method under comparison. The implementations for all four approaches are available in the `table3/` directory of the supplementary code repository: `aumasson.py`, `pattern.py`, `syncopation.py`, and `carrylock.py`.

Specifically, syncopation conditions can not control the propagation of differences in the $(n_2 + 1)$-th bit, i.e., the bit immediately following the block $[n_2 : n_1]$. The following lemma formalizes the specific scenario when the difference propagates to the (n_2+1)-th bit.

Lemma 2. *Consider two elements $z, y \in \mathbb{F}_2^n$, and let $[n_2 : n_1]$ represent a block of bits that can be modified arbitrarily to obtain $\bar{y}$. Suppose that the bit at position $n_2 + 1$ differs between z and y , i.e., $z[n_2 + 1] \neq y[n_2 + 1]$. If either of the following conditions holds:*

a) $z[n_2 : n_1] \geqslant y[n_2 : n_1]$ and $z[n_2 : n_1] < \bar{y}[n_2 : n_1]$, or

b) $z[n_2 : n_1] < y[n_2 : n_1]$ and $z[n_2 : n_1] \geqslant \bar{y}[n_1 : n_1]$,

then the difference $(z \boxminus y) \oplus (z \boxminus \bar{y})$ has a nonzero bit at position $n_2 + 1$.

Proof. Without loss of generality, assume that

$$z[n_2 : n_1] \geqslant y[n_2 : n_1] \quad \text{and} \quad z[n_2 : n_1] < \bar{y}[n_2 : n_1].$$

From the subtraction operation, the bit at position $n_2 + 1$ in $(z \boxminus y)$ is given by:

$$(z \boxminus y)[n_2 + 1] = z[n_2 + 1] \oplus y[n_2 + 1] \oplus B,$$

where B is the carry due to the bit segment $[n_1 - 1 : 0]$.

Given that $z[n_2 + 1] \neq y[n_2 + 1]$, i.e., $z[n_2 + 1] \oplus y[n_2 + 1] = 1$, it follows that:

$$(z \boxminus y)[n_2 + 1] = 1 \oplus B.$$

Similarly, for $(z \boxminus \bar{y})$, we have:

$$(z \boxminus \bar{y})[n_2 + 1] = z[n_2 + 1] \oplus \bar{y}[n_2 + 1] \oplus B \oplus 1.$$

Since we have $y[n_2+1] = \bar{y}[n_2+1]$, we substitute $\bar{y}[n_2+1] = y[n_2+1]$:

$$(z \boxminus \bar{y})[n_2+1] = z[n_2+1] \oplus y[n_2+1] \oplus B \oplus 1.$$

Given that $z[n_2+1] \oplus y[n_2+1] = 1$, we conclude:

$$(z \boxminus \bar{y})[n_2+1] = B.$$

Taking the XOR of the two differences, we obtain:

$$(z \boxminus y)[n_2+1] \oplus (z \boxminus \bar{y})[n_2+1] = 1 \oplus B \oplus B = 1.$$

This confirms that the difference $(z \boxminus y) \oplus (z \boxminus \bar{y})$ has a nonzero bit at position n_2+1.

The argument symmetrically holds for the second case where $z[n_2 : n_1] \leqslant y[n_2 : n_1]$ and $z[n_2 : n_1] > \bar{y}[n_2 : n_1]$, completing the proof. □

Comparison by an Illustration:

To illustrate the scenario mentioned above, and to compare it with the carry lock method, let us take an example.

Example 2. If we take a 32-bit word, y as

$$y = \texttt{0b00000110 10110100 00100111 00101111},$$

and randomly change [20:16] block (red in color) of y, we get

$$\bar{y} = \texttt{0b00000110 10111110 00100111 00101111}.$$

Next, we want to observe the difference between $z - y$ and $z - \bar{y}$ for the syncopation technique and the *carry-lock* method.

Syncopation Technique: In order to apply the syncopation technique, we consider the following z

$$z = \texttt{0b11100100 11010110 00111000 00000011}$$

It is worth noting that the condition of the syncopation technique is satisfied here. Also we have $z[20:16] > y[20:16]$ but after modification $z[20:16] < \bar{y}[20:16]$. Now, focusing on the block and its adjacent bits, we have,

$$\begin{aligned} z - y &= \texttt{0b}\ldots\texttt{00100010}\ldots\texttt{11010100} \\ z - \bar{y} &= \texttt{0b}\ldots\texttt{00011000}\ldots\texttt{11010100} \\ (z - y) \oplus (z - \bar{y}) &= \texttt{0b}\ldots\texttt{00111010}\ldots\texttt{00000000} \end{aligned}$$

We can see that there is an extra difference in the 21st bit (in blue) of $(z - y) \oplus (z - \bar{y})$ beyond the modified block.

The example demonstrates a scenario when, despite syncopation's conditions being satisfied, modifying a block (here, bits [20:16]) introduces a difference in the adjacent bit 21.

Applying *Carry-Lock* Method: We show that in the same example, the *carry-lock* method eliminates the possibility of difference propagation till the 21st bit. The conditions from Lemma 1 are as follows:

a) Ensure $z[15:0] \geqslant y[15:0]$ (prevents carries before the PNB block)

b) Set $z[20:16] = \texttt{0b11111}$ (guarantees carries generated within the PNB block resolve locally).

If we revisit the example with our conditions, we see that the first condition is already met, and if we put $z[20:16] = \texttt{0b11111}$, we have

$$z = \texttt{0b}\ldots\texttt{11011111}\ldots\texttt{00000011}.$$

Now we have

$$z - y = \texttt{0b}\ldots\texttt{00101011}\ldots\texttt{11010100}$$

$$z - \bar{y} = \texttt{0b}\ldots\texttt{00100001}\ldots\texttt{11010100}$$

$$(z - y) \oplus (z - \bar{y}) = \texttt{0b}\ldots\texttt{00001010}\ldots\texttt{00000000},$$

which shows that differences are strictly confined to the modified block [20:16]. Here, we say z is aligned with y.

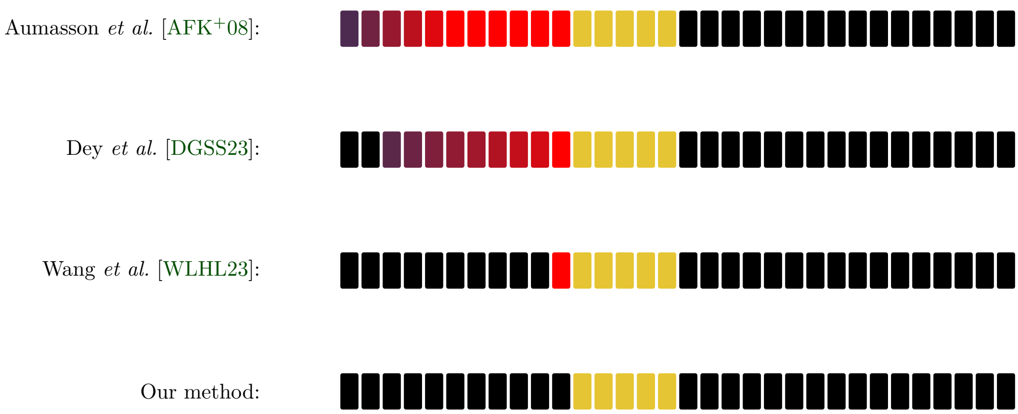


Figure 2: Comparison of the Difference Propagation.

Figure 2 represents the probability of difference propagation for the four approaches, for the PNB block $[20:16]$. The yellow color represents the block. The intensity of the red color represents the probability of the difference. Extreme black represents no difference. The transition from red to black represents the reduction in the probability of difference.

5.3 Improvement in the correlation of PNBs

In a PNB-based key recovery attack, the identification of PNBs with strong backward correlation is crucial. However, as the number of rounds in a cipher E increases, obtaining a sufficient number of PNBs becomes increasingly challenging. This is due to the diffusion introduced by additional rounds, which disperses the differences induced by key-bit modifications.

Consider a differential spanning r rounds, where the ChaCha cipher E operates over R rounds with $R > r$. If we maintain a fixed threshold for PNB selection, the number of PNBs available for an extended cipher E covering Q rounds ($Q > R$) will be lower than that for R rounds. This reduction arises because the additional $Q - R$ rounds in the backward direction further diffuse the difference induced by the key-bit flip, reducing the correlation needed for PNB selection.

Our *carry-lock* method plays a crucial role in mitigating this diffusion effect. Specifically, in the PNB-searching algorithm, if the keyword containing the key-bit to be flipped is aligned with the corresponding keystream word, the difference propagation is constrained within the bit. Essentially, under this alignment, the propagation behaves like an XOR operation rather than undergoing complex modular addition diffusion. This method significantly enhances the correlation associated with the key-bit, thereby increasing the likelihood of forming a PNB.

Finding extra PNBs: Building upon this observation, for the cipher E, we define another cipher $E^{\oplus}$ where

$$Z_w = X_w \boxplus X_w^R, \quad \text{for } w = 0, 1, \ldots, 15,$$

is replaced with

$$Z_w = X_w \oplus X_w^R, \quad \text{for } w = 0, 1, \ldots, 15.$$

Suppose we get a set of PNBs U in E. In the cipher $E^{\oplus}$ we get another set $U^{\oplus}$ of PNBs which contains U.

However, due to the change in operation, the additional elements would not behave as neutrally as the preliminary PNBs. To enhance the characteristics of these elements, we use the idea mentioned in Lemma 1. For these elements to be treated as PNBs, we have to impose some conditions on Z, Z' mentioned in Criteria 1.

Note that the second condition on Z and Z' ensures that even if we do not know the values of the bits in the PNB segment from X_w and X'_w, the second condition of the lemma will be satisfied. This results in an increase in the data complexity values, as explained thoroughly below:

Analysis of Data: Let us discuss the constraints in a step-wise manner as mentioned in Criteria 1, in order to analyze the required quantity of data. Note that, following the prior works as mentioned in Section 3.2, we will analyze the average-case scenario. The PNB block $[i_2 : i_1]$ is of size $(i_2 - i_1 + 1)$.

1. In Criteria 1, the first condition includes two sub-conditions, each of which is satisfied with probability $\frac{1}{2}$. So, on average, out of $2^2 = 4$ samples, 1 satisfies this condition.
2. For the condition on bit segment $[i_2 : i_1]$ of Z, Z', i.e., $Z[i_2 : i_1] = Z'[i_2 : i_1] =$ `0b11...1`, on average out of $2^{2(i_2 - i_1 + 1)}$ pairs, one will satisfy the condition.

Therefore, the number of samples required to obtain one suitable Z, Z' for a PNB block of size $(i_2 - i_1 + 1)$ is denoted by E_{D_1} and is given as

$$E_{D_1} = 2^{2 \cdot (i_2 - i_1 + 2)}.$$

6 Harmonizing Significant Bits Using the Carry-Lock Method

In Section 4 and 5, we applied the carry-lock method to PNB blocks in order to (i) prevent carry propagation across the target block and (ii) increase the backward correlation by making the subtraction behave like XOR on that block. In this section, we extend the same principle to *significant* (non-PNB) key bits. The goal is different: rather than turning additional bits into PNBs, we use carry-lock to expose a structural redundancy that reduces the number of independent guesses.

6.1 A structural identity behind harmonization

During the attack, The attacker guesses a state $\widehat{X}$, and computes $Z \boxminus \widehat{X}$ and then applies reverse rounds. Specifically, if we focus on the values of b' and c' obtained by the attacker by the $Z \boxminus \widehat{X}$ operation, are

$$b' = Z_b \boxminus \widehat{X}_b, \qquad c' = Z_c \boxminus \widehat{X}_c.$$

In the next step, the attacker applies the reverse round operation. Note that, during this reverse round, the first operation is

$$b = (b' \ggg l) \ \oplus \ c'$$

Expressing it using the words of Z and $\widehat{X}$, we have

$$b = \big[(Z_b \boxminus \widehat{X}_b) \ggg l\big] \ \oplus \ (Z_c \boxminus \widehat{X}_c). \tag{7}$$

Consider a segment $\mathcal{I} = [n_2 : n_1]$ of the word b. For ChaCha each word consists of 32 bits, hence b is the concatenation of 3 segments, left ($b[31 : n_2 + 1]$), middle ($b[\mathcal{I}]$), right ($b[n_1 - 1 : 0]$) (check b in Figure 3).

$$b[31 : 0] = b[31 : n_2 + 1] \parallel b[\mathcal{I}] \parallel b[n_1 - 1 : 0].$$

The middle segment $b[\mathcal{I}]$ can be expressed as the XOR of $b'[\mathcal{J}], c'[\mathcal{I}]$, during reverse round, where $\mathcal{J}$ represent the segment $[n_2 + l : n_1 + l]$ (check figure 3).

$$b[\mathcal{I}] = b'[\mathcal{J}] \oplus c'[\mathcal{I}] \tag{8}$$

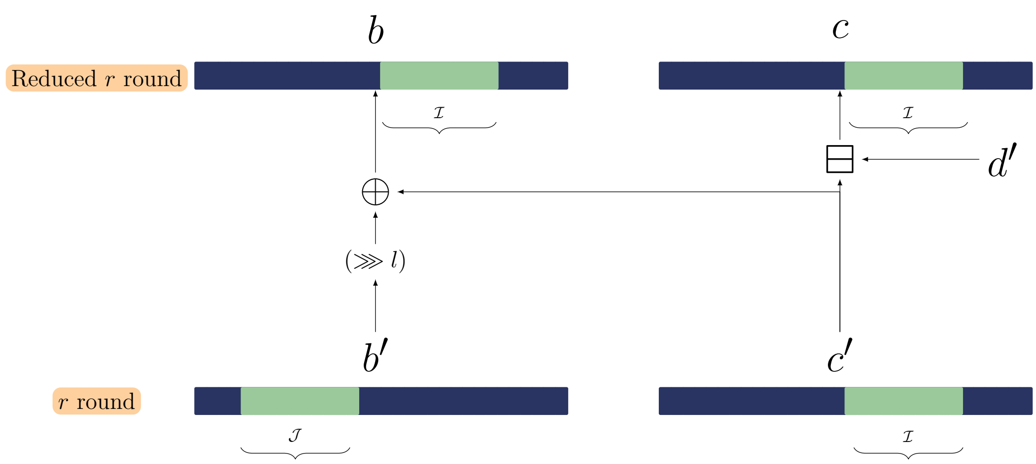


Figure 3: Selection of the *harmonic pairs* in a `quarterround` of ChaCha.

Harmonization identity. In our idea, we apply the carry-lock technique during both the operations $b' = Z_b \boxminus \widehat{X}_b$, $c' = Z_c \boxminus \widehat{X}_c$ to stop the carry propagation within the middle and last segment. Therefore, those segments can be expressed as follows:

$$b'[\mathcal{J}] = Z_b[\mathcal{J}] \boxminus \widehat{X}_b[\mathcal{J}], c'[\mathcal{I}] = Z_c[\mathcal{I}] \boxminus \widehat{X}_c[\mathcal{I}] \tag{9}$$

Therefore,

$$b[\mathcal{I}] = (Z_b[\mathcal{J}] \boxminus \widehat{X}_b[\mathcal{J}]) \oplus (Z_c[\mathcal{I}] \boxminus \widehat{X}_c[\mathcal{I}])$$

Algorithm 1: Choosing a harmonic pair

```
Input: A set P of PNB positions in the word c for r-round ChaCha (full cipher)
Output: Harmonic pair (b'[J], c'[I])
Compute the PNB set P' for the reduced cipher (with the last XOR-rotation
  update removed);
Remove positions common to P and P' to obtain the candidate set P'';
Choose a block I ⊆ P'';
Let J be the index image of I under the XOR-rotation update b' = (b ⊕ c') <<< l;
return (b'[J], c'[I]);
```

Further, if the attacker chooses Z satisfying the carry-lock conditions given in (Lemma 1), for both segments $Z_b[\mathcal{J}]$ and $Z_c[\mathcal{I}]$, according to the lemma, the subtraction operation gives the same output as the XOR operation, i.e.,

$$Z_b[\mathcal{J}] - \widehat{X}_b[\mathcal{J}] = Z_b[\mathcal{J}] \oplus \widehat{X}_b[\mathcal{J}], \qquad Z_c[\mathcal{I}] - \widehat{X}_c[\mathcal{I}] = Z_c[\mathcal{I}] \oplus \widehat{X}_c[\mathcal{I}].$$

Substituting into (9) gives the identity

$$b[\mathcal{I}] = (Z_b[\mathcal{J}] \oplus Z_c[\mathcal{I}]) \;\oplus\; (\widehat{X}_b[\mathcal{J}] \oplus \widehat{X}_c[\mathcal{I}]). \tag{10}$$

Therefore, under the carry-lock condition, in order to achieve the correct value of $b[\mathcal{I}]$, the attacker need to guess the linear combination $\widehat{X}_b[\mathcal{J}] \oplus \widehat{X}_c[\mathcal{I}]$ correctly, not the individual values of $\widehat{X}_b[\mathcal{J}]$ and $\widehat{X}_c[\mathcal{I}]$.

Without carry-lock, in the existing approach, computing $b[\mathcal{I}]$ from (8) require correct guess of both $\widehat{X}_b[\mathcal{J}]$ and $\widehat{X}_c[\mathcal{I}]$, which has $2^{2\mathcal{I}}$ possible guesses. With carry-lock the number of possible guesses come down to $2^{\mathcal{I}}$.

We call the pair of segments $(b'[\mathcal{J}], c'[\mathcal{I}])$ a *harmonic pair*, and we call $b'[\mathcal{J}]$ the *harmonic counterpart* of $c'[\mathcal{I}]$.

6.2 Identifying harmonic-pair blocks

We define a reduced version of the cipher obtained by removing the final operation $b' = (b \oplus c') \lll l$ in the last round. We need to identify key bit blocks at c' which behave as PNBs in the reduced version, but not PNBs in the full version.

Once we find such a block $\mathcal{I} = [n_2 : n_1]$, the *harmonic pair* is $c[\mathcal{I}$ and $b[\mathcal{J}]$, where $\mathcal{J}$ represents the block l-bits towards the left, i.e., $[n_2 + l : n_1 + l]$. When we restore the removed operation, the bits in $c'[\mathcal{I}]$ and $b'[\mathcal{J}]$ become non-PNB in the full cipher. The explanation is simple, in the reverse-round, the blocks that are influenced by $c'[\mathcal{I}]$ and $b'[\mathcal{J}]$ are $b[\mathcal{I}]$ and $c[\mathcal{I}]$ (see Figure 3). According to our process, if the linear combination of $X_b[\mathcal{J}] \oplus X_c[\mathcal{I}]$ are guessed correctly, even if the individual values are incorrect, that will still lead to a correct value of $b[\mathcal{I}]$. And, a possible incorrect value of $c[\mathcal{I}]$ will not affect the attack because according to our choice it is at PNB position of reduced round. The procedure is given in algorithm 1.

6.3 Carry-lock constraints for significant bits and data cost

To exploit Equation 10 in the key-recovery procedure, carry-lock must hold on the significant segments in both words of a harmonic pair. Unlike the PNB case, these segments are not freely assignable: they contain significant key bits that are guessed. Therefore, the attacker must filter keystream pairs (Z, Z') in a way that guarantees that, *if the guess is correct*, the carry-lock conditions hold on the targeted segments in both $(Z_b \boxminus \widehat{X}_b)$ and $(Z_c \boxminus \widehat{X}_c)$.

Criteria 2: ***Criteria to Execute Carry-Lock on significant (non-PNB) blocks***

Let $X_b[j_2 : j_1]$ be a significant segment and let $X_c[j_4 : j_3]$ be its harmonic counterpart. For a guess $(\widehat{X}, \widehat{X}')$, the attacker keeps only keystream pairs (Z, Z') satisfying:

1.
$$\begin{aligned} Z_b[j_1-1:0] &\geqslant \widehat{X}_b[j_1-1:0], \quad & Z'_b[j_1-1:0] &\geqslant \widehat{X}'_b[j_1-1:0], \\ Z_c[j_3-1:0] &\geqslant \widehat{X}_c[j_3-1:0], \quad & Z'_c[j_3-1:0] &\geqslant \widehat{X}'_c[j_3-1:0]. \end{aligned} \tag{11}$$

2. For every $p \in [j_2 : j_1]$, if $\widehat{X}_b[p] = 1$ then $Z_b[m] = Z'_b[p] = 1$. Similarly, for every $q \in [j_4 : j_3]$, if $\widehat{X}_c[q] = 1$ then $Z_c[q] = Z'_c[q] = 1$.

The first item prevents a carry from propagating into the target segment from lower bits. The second item ensures that, on each locked bit, the condition $Z[\cdot] \geqslant \widehat{X}[\cdot]$ holds bitwise, so the subtraction on that bit does not generate a carry and thus matches XOR on the segment, as required by Lemma 1.

Expected data filtering cost.

- The first condition consists of four independent $\geqslant$ constraints. Therefore on average, out of $2^4 = 16$ random pairs of output keystreams and a random guess of X, one satisfies the condition.
- The bit segment $\mathcal{I} := [j_2 : j_1]$ has length $|\mathcal{I}| = j_2 - j_1 + 1$. For a random guess, each bit of $\widehat{X}_b[j_2 : j_1]$ equals 1 with probability 1/2, so the second condition enforces, on average, $|\mathcal{I}|/2$ constraints on $Z_b[j_2 : j_1]$. The same reasoning applies to $Z'_b[j_2 : j_1]$, $Z_c[j_4 : j_3]$, and $Z'_c[j_4 : j_3]$, giving an average of
$$4 \cdot \frac{|\mathcal{I}|}{2} \;=\; 2|\mathcal{I}| \;=\; 2\,(j_2 - j_1 + 1)$$
bit-constraints in total. Each such constraint is satisfied with probability 0.5.

Combining both parts, the expected number of keystream pairs needed to obtain one suitable (Z, Z') pair is
$$E_{D_2} = 2^{\,2\cdot(j_2 - j_1 + 3)}. \tag{12}$$

6.4 Modification in the Complexity calculation

As explained thoroughly in Subsection 5.3 and Section 6, to find one suitable pair of Z and Z' among the number of samples N (the formulation given in Equation 5) which satisfy the condition mentioned in Equation 11, the total number of samples required is given as

$$N^T = E_{D_1} \cdot E_{D_2} \cdot N. \tag{13}$$

In Subsubsection 2.2.1, we mentioned that the maximum limit for the number of samples is 2^{96}. Hence, in the key recovery attack, we try to choose the number of samples E_{D_1} and E_{D_2} in such a way that $N^T \leqslant 2^{96}$. Similarly, the idea of harmonizing the significant bit will help us reduce the number of guesses. Hence, there will be modifications in the formulation of time complexity. As explained in Section 6, if we consider a significant bit segment $\mathcal{I} := [n_2 : n_1]$ which spans $|\mathcal{I}| = (n_2 - n_1 + 1)$ bits. Then the dimension of non-PNB guess space $\mathcal{G}$ reduces by a factor, and hence the dimension of the new non-PNB guess space

$\mathcal{G}_{\text{new}}$ is given by $\dim(\mathcal{G}_{\text{new}}) = m - |\mathcal{I}| = m - (n_2 - n_1 + 1)$. Therefore, if we select l such bit-segments, then accordingly the formula of time complexity (given in Equation 6) is to be modified. Previously, the number of guesses to recover significant bits were 2^m, where m is the number of significant bits. In the new approach, the number of guesses would be $2^{\dim(\mathcal{G}_{\text{new}})}$. So, m will be replaced by $\dim(\mathcal{G}_{\text{new}})$ in the equation. Similar argument goes for the recovery of PNBs, where $2^{|K|-m}$ would be replaced by $|K| - \dim(\mathcal{G}_{\text{new}})$ in the formula. So, the modified formula is

$$C = \sum_{i=1}^{k} 2^{m_i} \cdot N + 2^{\dim(\mathcal{G}_{\text{new}})} \cdot N \times \frac{k-1}{2^{11} \times (R-r)} + 2^{|K|-\alpha} + 2^{|K|-\dim(\mathcal{G}_{\text{new}})}, \tag{14}$$

where $\dim(\mathcal{G}_{\text{new}}) = m - \sum_{i=1}^{l} |\mathcal{I}_i|$ and $|K|$ is the key size.

Also, m is the dimension of the full non-PNB guess space $\mathcal{G}$ (See Equation 6) obtained after eliminating the PNBs from the total number of keys, l is the number of significant bit-segments selected that are harmonized, and $|\mathcal{I}_i|$ denotes the dimension of the i-th block of significant key bits as explained in Section 6.

7 Key Recovery Process and Application on ChaCha

In this section, we analyze the 128-bit and 256-bit key versions of ChaCha, focusing on attacks against ChaCha7.5/256 and the first-ever attack on ChaCha7/128. These attacks are based on the *carry-lock* method, detailed in Section 4. The improvements in data and time complexity stem from applying the *carry-lock* method to PNB blocks and harmonizing significant bits, as thoroughly discussed in Section 5 and Section 6, respectively. These refinements lead to modifications in the data and time complexity formulas, which are presented in Subsection 6.4. We start with the key recovery process.

The whole key recovery in the online phase starts with the data collection phase.

Data Collection:

An attacker selects an IV v and records the corresponding keystream Z. Let the pair (v, Z) be collectively denoted as D. Similarly, the attacker obtains the differenced version D' and collects paired observations (D, D'). Suppose the attacker gathers a total of N such pairs.

Among these, a subset of n positions corresponds to PNBs, some of which appear as structured blocks. To refine the dataset, the attacker filters the N pairs based on the unity condition imposed on the keystream at PNB positions, retaining only N^T pairs that satisfy this condition.

Given that the attacker can make informed guesses about the significant (non-PNB) bits, they selectively choose data such that the other *carry-lock* condition is satisfied.

Significant Part Guess:

Now the attacker has to make guesses about $m = 256 - n$ bits and analyze the guess. Among m significant bits, there are n_1 bits that the attacker will guess, combined so that they will guess $m - n_1$ bits. According to the guess, the attacker has to choose the IV and keystream blocks from N' where the first condition of the *carry-lock* method is true for the keystreams. Once the attacker makes a potential correct guess, they will brute force the PNBs along with the combined significant bits.

Attack on ChaCha

To mount the key recovery attack, we use the 3.5-round differential-linear distinguisher $(\Delta X_{13}^{(0)}[6], \Gamma_2^{(3.5)}[0])$ with correlation $\varepsilon_d = 0.00317$ from [BLT20]. We linearly extend the single bit distinguisher to one half round more to a multi-bit distinguisher $\Gamma_2^{(4)}[0] \oplus \Gamma_8^{(4)}[0] \oplus \Gamma_7^{(4)}[7]$ with $\varepsilon_l = 1$. Working with a multi-bit output differential allows us to lower the overall time complexity when we apply the divide-and-conquer strategy described by Dey [Dey24]. In all of our attacks we have used $\Phi^{-1}[\Pr_{nd}] = 0.8$. The experimental results presented in this section can be reproduced using the source codes from the GitHub repository.

7.1 Experimental Results for the attack on ChaCha7.5/256

To improve the attack on the ChaCha7.5/256, we first collect 15 PNBs by keeping the threshold value $\gamma = 0.4$. The PNB search was conducted over 2^{20} random state pairs for each key bit position. As discussed in Subsection 5.3, by replacing $\boxplus$ operation with $\oplus$ in Equation 2 we increase the number of PNBs to 25. We further carried out the search under the exact carry-lock conditions (`pnb_search_carry_lock_condition.cpp`) and obtained the same PNB set, which verifies our claim that the carry-lock constraints make the relevant subtraction behave like XOR on the targeted bit for the purpose of PNB identification. The identified PNBs and their individual correlation values are stored in the `chacha7.5_pnbs` directory of the repository.

Table 4: Experimentally observed correlation values of PNB block for ChaCha7.5/256 and comparison with previous approaches.

Keyword	Bit-Segment (seg_i)	Correlation (per technique)			
		Aumasson *et al.* [AFK+08]	Dey *et al.* [DGSS23]	Wang *et al.* [WLHL23]	This Work
k_2	$[8:6]$	0.46	0.51	0.54	0.68
	$[25:22]$	0.22	0.23	0.24	0.72
k_3	$[11:7]$	0.33	0.39	0.31	0.78
	$[27:24]$	0.45	0.5	0.52	0.63
k_4	$[30:27]$	0.60	0.62	0.60	0.62

I. Out of the 25 PNBs, we first filter out the five PNB segments comprising a total of 20 PNBs. The blocks, along with their correlation, are mentioned Table 4. Here, for each block of PNBs, we compare the experimentally observed correlation value obtained using the *carry-lock* method introduced in our work with the experimentally observed correlation for previous three major ideas explained in Section 3. The correlation values reported here were obtained by averaging over 2^{30} random samples for each configuration. The correlation computation was performed using `correlation_check.cpp`.

   Firstly, this shows that our idea produces higher correlation than previous approaches. Secondly, We compared this XOR-conditioned correlation with the theoretical correlation i.e. with carry-lock condition using about 2^{15} random trials, and observed close agreement in all tested cases (`correlation_check_carry_lock_condition.cpp`). For example, for the segment $\mathcal{I} = [8:6]$, we impose the carry-lock conditions and with these conditions, the resulting correlation is 0.68287, whereas the XOR-conditioned evaluation gives 0.68227, which is consistent with the theoretical prediction.

   Since there are five bit-segments (seg_i, $1 \leqslant i \leqslant 5$) of lengths 3, 4, 5, 4 and 4 respectively, the total number of samples required to apply the *carry-lock* method is $2^{2\times(3+1)} \times 2^{2\times(4+1)} \times 2^{2\times(5+1)} \times 2^{2\times(4+1)} \times 2^{2\times(4+1)} = 2^{50}$.

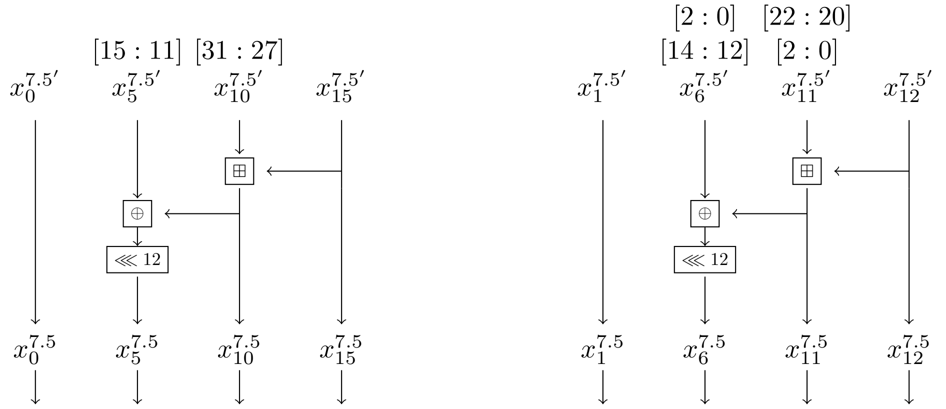
II. The remaining five PNBs with their correlation are mentioned in Table 5. Since we have high correlation values for the single PNBs, we do not apply the *carry-lock* method. Consequently, for all the 25 PNBs, the backward correlation (ε_a) is calculated by multiplying all the correlation values of the bit-segments (in *carry-lock* method) and the single PNBs, which is 0.03288.

Table 5: List of 5 PNBs for ChaCha7.5/256 and their experimentally observed correlation

Keyword	k_0	k_1	k_2			k_3	k_4	k_5	k_6	k_7
Bit	-	-	11	14	31	31	-	-	-	31
Correlation	-	-	0.62	0.67	0.84	0.81	-	-	-	0.78

The attack can be further improved by *harmonizing* some of the significant bits (non-PNBs).

III. As described in Subsection 6.2, we apply the *carry-lock* method in significant bits. From the reduced cipher version, we select the bit-segment $[31:27]$ from the keyword $k_6(X_{10})$. The corresponding bit segment from $k_1(X_5)$ is $[15:11]$. Now, instead of guessing these two significant parts, we can guess the combined $(X_5[15:11] \oplus X_{10}[31:27])$. As a result, the significant search space complexity reduces by a factor of 2^5, but to apply this technique we need on average $E_{D_2} = 2^{2\times 7} = 2^{14}$ samples.



(a) A part of `quarterround` $(X_0, X_5, X_{10}, X_{15})$.

(b) A part of `quarterround` $(X_1, X_6, X_{11}, X_{12})$.

A similar combination of significant bits are $(X_6[14:12] \oplus X_{11}[2:0])$ and $(X_6[2:0] \oplus X_{11}[22:20])$ as shown in Figure 4a. Here, applying the harmonizing trick will give us a significant space reduction by a factor of 2^6; however, the resulting samples requirement for this setting is $E_{D_2} = 2^{2\times 8} = 2^{16}$.

IV. Combining all the factors, now we apply the attack technique of [Dey24]. The initial set of PNBs consists of the same 25 PNBs. Now the output mask of the DL distinguisher involves three bits $\Delta X_2^{(4)}[0]$, $\Delta X_8^{(4)}[0]$, and $\Delta X_7^{(4)}[7]$. The PNBs corresponding to these three bits are given in Table 6.

Complexity: We first obtain 25 PNBs utilizing the *carry-lock* technique. Because of these settings, we need an extra 2^{50} data. We apply the *harmonizing* technique only once, to jointly guess $(X_5[15:11] \oplus X_{10}[31:27])$. Consequently, the effective significant (non-PNB)

Table 6: PNBs for each bit of the mask in ChaCha7.5/256.

Keywords	Bits		
	$\Delta X_2^{(4)}[0]$	$\Delta X_8^{(4)}[0]$	$\Delta X_7^{(4)}[7]$
k_0	-	$[2:0]$, 31	-
k_1	$[8:3]$	-	-
k_2	$[14:12]$, $[20:19]$	$[6:1]$, 13, $[30:29]$	$[10:9]$
k_3	-	$[15:12]$, $[26:24]$	$[7:2]$, 20, $[24:22]$, $[30:29]$
k_4	-	-	8
k_5	30	-	-
k_6	26	-	-
k_7	0, 20	-	-
Count	14	19	15
Correlation (ε_i)	0.72	0.97	0.85

search space is reduced to dimension $m = 256 - 25 - 5 = 226$, at the cost of increasing the data requirement by a factor of 2^{14}. At this point, the total data inflation is 2^{64}.

From Table 6, following the work from [Dey24] the correlation value $\varepsilon = \varepsilon_d \times \varepsilon_a \times \prod_{i=1}^{3} \varepsilon_i = 0.00317 \times 0.03288 \times 0.72 \times 0.97 \times 0.85 = 2^{-13.98}$. For $\alpha = 12.5$ we obtain $N^T = 2^{31.46}$ using Equation 5. With $R = 7.5$, $r = 4$, $k = 3$, $\dim(\mathcal{G}_{\text{new}}) = 226$, $m_1 = 212$, $m_2 = 207$ and $m_3 = 211$ the total time complexity $C = 2^{246.29}$ from Equation 14. Finally multiplying N^T by 2^{64} gives us the total data requirement of $N = 2^{95.46}$. The complexity values were computed using `complexity_256.py`, which implements Equation 14 with the experimentally obtained correlation values.

7.2 Experimental result for attack on ChaCha7/128

The most recent attack on the 128-bit key version of ChaCha, mentioned in [Dey24] mounted an attack on ChaCha6.5/128. Following that work, we carry out the first attack against ChaCha7/128.

Using the threshold $\gamma = 0.15$, we initially identify 17 PNBs experimentally. Applying the carry-lock criteria, the same threshold produces 11 additional candidate PNBs. After screening for attack relevance, we retain 7 of these candidates and discard the rest, yielding a total of 24 PNBs used in the final attack.

Table 7: List PNBs for ChaCha7/128.

Keyword	k_0	k_1	k_2					k_3				
Bit	-	-	$[3:2]$	$[9:7]$	$[21:15]$	$[29:27]$	31	0	$[9:8]$	$[21:20]$	$[24:23]$	31

From Table 7, we obtain seven bit-segments; we apply the *carry-lock* condition to these segments and to the remaining single-bit positions, excluding the 31st bits since imposing conditions on them is ineffective. In order to further improve the correlation, we applied the pattern technique from [DGSS23] to the bit-segments. The correlation value we obtain

Table 8: PNBs for each bit of the mask in ChaCha7/128.

Keywords	Bits		
	$\Delta X_2^{(4)}[0]$	$\Delta X_8^{(4)}[0]$	$\Delta X_7^{(4)}[7]$
k_0	31	31	-
k_1	$[31:0]$	-	-
k_2	-	-	$[14:10]$, $[26:22]$, 30
k_3	-	$[19:10]$, 22, $[30:25]$	-
Count	33	18	11
correlation (ε_i)	1	1	1

for these 24 PNBs together is 0.00813. The correlation was computed experimentally using `correlation_check.cpp`.

The significant search space here is of dimension $m = 128 - 24 = 104$. Since we applied our technique of *carry-lock* to bit-segments of lengths 2, 3, 7, 3, 1, 2, 2 and 2, $E_{D_1} = 2^{2\times(2+1)} \times 2^{2\times(3+1)} \times 2^{2\times(7+1)} \times 2^{2\times(3+1)} \times 2^{2\times(1+1)} \times 2^{2\times(2+1)} \times 2^{2\times(2+1)} \times 2^{2\times(2+1)} = 2^{60}$. In this case, we did not have any advantage using the *harmonizing* technique, hence $E_{D_2} = 2^0 = 1$. As a result $\dim(\mathcal{G}_{\text{new}}) = m = 104$.

Following a similar approach as of ChaCha7.5/256, here we find out the PNBs for each of the bits of the mask, as noted in Table 8.

Complexity: The correlation value is $\varepsilon = \varepsilon_d \times \varepsilon_a \times \prod_{i=1}^{3} \varepsilon_i = 0.00317 \times 0.00813 \times 1 \times 1 \times 1 = 2^{-15.24}$. Using $\alpha = 3.45$, we have $N^T = 2^{31.43}$. In this case, $|K| = 128$, $R = 7$, $r = 4$, $k = 3$, $\dim(\mathcal{G}_{\text{new}}) = 104$, $m_1 = 71$, $m_2 = 86$ and $m_3 = 93$, hence the time complexity C becomes $2^{125.90}$. Since we use the *carry-lock* technique the data complexity (N) of this attack is $2^{31.43} \times 2^{60} = 2^{91.43}$. The complexity computation for ChaCha7/128 was performed using `complexity_128_24.py`.

8 Conclusion

This work tackles the carry propagation issue by effectively confining it using the *carry-lock* method, which enhances the existing PNB-based differential-linear cryptanalysis of ChaCha. Our approach not only increases the number of identifiable PNBs but also strengthens the backward correlation, leading to a more effective attack. Specifically, we have improved the attack on ChaCha7.5/256 and, for the first time, successfully mounted an attack on ChaCha7/128 with a complexity lower than brute force. This advancement opens new directions for applying similar techniques to higher-round variants of ChaCha, as well as other ARX-based designs.

References

[AFK+08] Jean-Philippe Aumasson, Simon Fischer, Shahram Khazaei, Willi Meier, and Christian Rechberger. New Features of Latin Dances: Analysis of Salsa, ChaCha, and Rumba. In Kaisa Nyberg, editor, *Fast Software Encryption*, volume 5086, pages 470–488, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[BBC+22] Christof Beierle, Marek Broll, Federico Canale, Nicolas David, Antonio Flórez-Gutiérrez, Gregor Leander, María Naya-Plasencia, and Yosuke Todo. Improved Differential-Linear Attacks with Applications to ARX Ciphers. *J. Cryptol.*, 35(4), oct 2022.

[Ber08a] Daniel Bernstein. ChaCha, a variant of Salsa20. *In Workshop Record of SASC*, pages vol. 8, pp. 3–5, 2008.

[Ber08b] Daniel J. Bernstein. The Salsa20 Family of Stream Ciphers. In Matthew Robshaw and Olivier Billet, editors, *New Stream Cipher Designs: The eSTREAM Finalists*, volume 4986, pages 84–97. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[BGG+23] Emanuele Bellini, David Gerault, Juan Grados, Rusydi H. Makarim, and Thomas Peyrin. Boosting Differential-Linear Cryptanalysis of ChaCha7 with MILP. *IACR Transactions on Symmetric Cryptology*, 2023(2):189–223, 2023.

[BLT20] Christof Beierle, Gregor Leander, and Yosuke Todo. Improved Differential-Linear Attacks with Applications to ARX Ciphers. In *CRYPTO(3)*, volume 12172 of *Lecture Notes in Computer Science*, pages 329–358. Springer, 2020.

[CM17] Arka Rai Choudhuri and Subhamoy Maitra. Significantly Improved Multi-bit Differentials for Reduced Round Salsa and ChaCha. *IACR Transactions on Symmetric Cryptology*, 2016(2):261–287, 2017.

[CSN21] Murilo Coutinho and Tertuliano C. Souza Neto. Improved Linear Approximations to ARX Ciphers and Attacks Against ChaCha. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021*, volume 12696, pages 711–740, Cham, 2021. Springer International Publishing.

[Dey24] Sabyasachi Dey. Advancing the idea of probabilistic neutral bits: first key recovery attack on 7.5 round ChaCha. *IEEE Transactions on Information Theory*, 2024.

[DGM23] Sabyasachi Dey, Hirendra Kumar Garai, and Subhamoy Maitra. Cryptanalysis of Reduced Round ChaCha- New Attack and Deeper Analysis. *IACR Transactions on Symmetric Cryptology*, page 89–110, Mar. 2023.

[DGSS22] Sabyasachi Dey, Hirendra Kumar Garai, Santanu Sarkar, and Nitin Kumar Sharma. Revamped Differential-Linear Cryptanalysis on Reduced Round ChaCha. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology – EUROCRYPT 2022*, pages 86–114, Cham, 2022. Springer International Publishing.

[DGSS23] Sabyasachi Dey, Hirendra Kumar Garai, Santanu Sarkar, and Nitin Kumar Sharma. Enhanced Differential-Linear Attacks on Reduced Round ChaCha. *IEEE Transactions on Information Theory*, 69(8):5318–5336, 2023.

[DS17] Sabyasachi Dey and Santanu Sarkar. Improved analysis for reduced round Salsa and Chacha. *Discrete Applied Mathematics*, 227:58–69, 2017.

[FGT25] Antonio Flórez-Gutiérrez and Yosuke Todo. Improved cryptanalysis of chacha: Beating pnbs with bit puncturing. In Serge Fehr and Pierre-Alain Fouque, editors, *Advances in Cryptology – EUROCRYPT 2025*, pages 427–457, Cham, 2025. Springer Nature Switzerland.

[Leu16] Gaëtan Leurent. Improved Differential-Linear Cryptanalysis of 7-Round Chaskey with Partitioning. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part I*, volume 9665 of *Lecture Notes in Computer Science*, pages 344–371. Springer, 2016.

[LH94] Susan K. Langford and Martin E. Hellman. Differential-Linear Cryptanalysis. In Yvo G. Desmedt, editor, *Advances in Cryptology — CRYPTO '94*, pages 17–25, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.

[Mai16] Subhamoy Maitra. Chosen IV cryptanalysis on reduced round ChaCha and Salsa. *Discrete Applied Mathematics*, 208:88–97, 2016.

[SCS25] Soumya Sahoo, Debasmita Chakraborty, and Santanu Sarkar. Chasing shadows: Advancements in Differential-Linear Cryptanalysis for ChaCha. *IEEE Transactions on Information Theory*, 71(8):6451–6469, 2025.

[SDSM25] Nitin Kumar Sharma, Sabyasachi Dey, Santanu Sarkar, and Subhamoy Maitra. On Improved Cryptanalytic Results Against ChaCha for Reduced Rounds ≥ 7. In Sourav Mukhopadhyay and Pantelimon Stănică, editors, *Progress in Cryptology – INDOCRYPT 2024*, pages 29–52, Cham, 2025. Springer Nature Switzerland.

[SM87] Akihiro Shimizu and Shoji Miyaguchi. Fast data encipherment algorithm FEAL. In David Chaum and Wyn L. Price, editors, *Advances in Cryptology - EUROCRYPT '87, Workshop on the Theory and Application of of Cryptographic Techniques, Amsterdam, The Netherlands, April 13-15, 1987, Proceedings*, volume 304 of *Lecture Notes in Computer Science*, pages 267–278. Springer, 1987.

[SZFW13] Zhenqing Shi, Bin Zhang, Dengguo Feng, and Wenling Wu. Improved Key Recovery Attacks on Reduced-Round Salsa20 and ChaCha. In Taekyoung Kwon, Mun-Kyu Lee, and Daesung Kwon, editors, *Information Security and Cryptology – ICISC 2012*, pages 337–351, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[WLHL23] Shichang Wang, Meicheng Liu, Shiqi Hou, and Dongdai Lin. Moving a Step of ChaCha in Syncopated Rhythm. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023*, pages 273–304, Cham, 2023. Springer Nature Switzerland.

[XXTQ24] Zhichao Xu, Hong Xu, Lin Tan, and Wenfeng Qi. Differential-Linear Cryptanalysis of Reduced Round ChaCha. *IACR Transactions on Symmetric Cryptology*, 2024:166–189, 06 2024.