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 ~~probabilistically~~ probabilistic neutral bit-based differential attacks on ChaCha. Existing attacks rely on ~~probabilistically~~ probabilistic neutral bits (PNBs) by partitioning key bits into significant bits and PNBs and recovering them in two stages. We observe that the ~~bias~~ 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 ~~bias~~ 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”~~*keystream*. This makes ~~them ideal for real-time applications like secure messaging, where speed and low computational overhead are critical. A key framework for designing~~ ARX ciphers particularly attractive for software implementations, offering high throughput with minimal resource requirements. Many modern stream ciphers ~~is the ARX paradigm~~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 ~~pioneered the use of ARX for fast ,~~ was a fast yet secure encryption [Ber08b]. Its successor, ChaCha (2008), improved security by enhancing “diffusion”~~—a~~ -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. ~~They rely on a core function that processes a fixed-size block of 512 bits using rounds of transformations involving ARX operations.~~ The key, nonce, and block counter are the inputs that ensure each keystream block is unique and secure.

~~There are several observations regarding the security of ARX ciphers. First, the addition operation introduces~~ 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 ~~, raising the~~ is not merely an increase in "algebraic complexity"~~of the cipher, which hinders equation-solving attacks. Second, rotations and XORs with constants break symmetry, making it harder for attackers to exploit predictable bit patterns (~~: 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 ~~attacks). Together, these operations rapidly mix input bits, preventing adversaries from isolating useful structures. For example, integral attacksthat rely on tracking specific bit groups fail because ARX operations spread data chaotically in just a few steps~~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 ~~probabilistically~~ 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 ~~bias~~correlation. In the backward direction, the attacker guesses the significant key bits and finds out if there is a ~~bias~~ correlation for the guess, which is called the backward ~~bias~~correlation.

In this work, we focus on the backward direction of the attack by enhancing the correlations of the ~~probabilistically neutral bits (PNBs)~~ 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 ~~the Carry Propagation:~~carry propagation (carry-lock). Section 4 ~~presents our novel idea~~introduces our main tool, the *carry-lock* method~~. In the~~, used in the backward evaluation of a PNB-based ~~attack method, after~~ distinguisher. After guessing the significant ~~key bitsand putting random values in the PNBs~~key-dependent bits, the attacker ~~generates a~~ completes a full state $\bar{X}$ ~~, subtracted from the output keystream , i.e., $Z - \bar{X}$. It is noticed that the backward bias significantly depends on the difference between $Z - X$ and $Z - \bar{X}$, i.e., a reduction in the number of differences increases the bias. The minimum possible difference is the one that we achieve when subtraction is substituted by XOR. We analyze the subtraction operation and identify certain conditions on some parts of the Z keystream such that the possibility of a carry is locked in some specific segments~~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, ~~we can achieve the same~~ on the PNB segment $\mathcal{I}$, we achieve the minimum number of differences ~~as XOR~~ (the XOR case), which improves the correlation ε_a.

Improving the ~~Bias~~ 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 ~~PNB~~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 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~~Key-Recovery: Section 7 elaborates ~~the key recovery~~ 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.

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

Table 1: Complexities of ~~Key Recovery Attacks~~ key-recovery attacks on ChaCha and ~~Our Result~~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

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 ~~key is repeated~~ 128-bit key occupies the second row and is copied into the third row ~~of the matrix~~(i.e., the second and third rows are identical).

Table 2: Table of ~~Notations~~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
$\Gamma^r_m[n]$	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

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 ~~applications of~~ 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 ~~modulo~~ 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:

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

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,

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

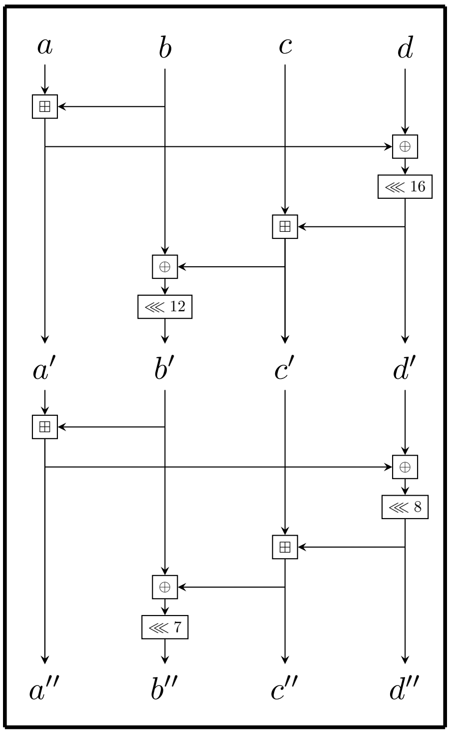


Figure 1: ~~One Quarterround Function in~~ `quarterround` function of ChaCha.

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 ~~takes advantage of the~~ is built upon a differential-linear distinguisher~~found earlier~~.

2.2.1 Differential-Linear Distinguisher:

Constructing an r-round differential-linear distinguisher ~~involves~~ can be described by decomposing the r-round ChaCha ~~cipher~~ permutation E into three ~~distinct sub-ciphers: E_1, E_m, and E_2. The cipher can be expressed as a composition of these parts , $E = E_2 \circ E_m \circ E_1$,~~ 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, ~~satisfying $r = r_1 + r_2 + r_3$~~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.

~~Suppose we have a differential (Δ_{in}, Δ_m) for the sub-cipher E_1 with probability p. We start with a state X and inject the input difference Δ_{in}, producing $X' = X \oplus \Delta_{in}$ and process both states through r_1 rounds of ChaCha. We vary the state X (the IVs and the keys), repeat the process, and in the output states, we observe the differential Δ_m~~ **Differential part (E_1).** Assume there exists a differential (Δ_{in}, Δ_m) through E_1 with probability p, i.e.,

$$\Pr_X\left[E_1(X)^{r_1} \oplus E_1(X' \oplus \Delta_{in})^{r_1} = \Delta_m\right] = p.$$

~~Using this equation, we filter out such states (X, X') that give Δ_m difference after r_1 rounds. The Key and IV form a pair and are called the right pair~~We call a pair $(X, X \oplus \Delta_{in})$ a right pair for E_1 if it satisfies the event inside the probability. In the ~~attack procedure, an IV is fixed to one column, and the remaining IVs can be changed to generate the keystream. Hence, the possible keystreams that can be generated are restricted to 2^{96}.~~ 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.

~~Next if we have a~~ **Differential-linear part (E_m).** Let Γ_m be a nonzero mask. Suppose (Δ_m, Γ_m) forms a differential-linear distinguisher ~~(Δ_m, Γ_m)~~ for E_m with correlation ε'_d, ~~then we have~~ meaning

$$2\Pr{}_{\cancel{X[\Gamma_m((X)^{r_2}\oplus(X\oplus\Delta_m)^{r_2})=0]-1}\,U}\left[\langle \Gamma_m, E_m(U) \oplus E_m(U \oplus \Delta_m)\rangle = 0\right] - 1 = \varepsilon'_d.$$

~~Assuming independency between~~ Assuming that conditioning on the E_1 ~~and~~ right-pair event does not significantly change the correlation behavior of this distinguisher for E_m ~~we have for~~, the composition $E_m \circ E_1$ ~~,~~

$$\cancel{2\Pr_X\left[\Gamma_m\left((X)^{r_1+r_2} \oplus (X')^{r_1+r_2}\right) = 0\right] - 1 = p\varepsilon'_d.}$$

yields

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

~~Apparently we have a differential-linear distinguisher for the cipher $E_m \circ E_1$ with correlation $p\varepsilon'_d$. This correlation~~ Equivalently, conditioned on right pairs, the correlation through E_m remains ε'_d ~~is called the *forward bias* in the literature~~, while unconditionally it is scaled by the right-pair probability p.

~~Lastly if there exists~~ **Linear part (E_2).** Finally, assume there is a linear approximation ~~$(\Gamma_m, \Gamma_{\text{out}})$~~ $(\Gamma_m, \Gamma_{\text{out}})$ for E_2 with correlation ε_l ~~for E_3, we have~~

$$2 \Pr_X \left[\Gamma_m(X) \oplus \Gamma_{\text{out}}((X)^{r_3}) = 0\right] - 1 = \varepsilon_l^2.$$

~~Combining all three distinguishers we have,~~ , i.e.,

$$2 \Pr{}_X \left[\Gamma_{\text{out}}\left((X)^r \oplus (X')^r\right) = 0\right] - 1_V \left[\langle \Gamma_m, V \rangle \oplus \langle \Gamma_{\text{out}}, E_2(V) \rangle = p0 \right] -1 \;= \varepsilon'_d \varepsilon^2{}_l,.$$

~~and we have a~~ 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 cipher with bias $\varepsilon_d = \varepsilon'_d \varepsilon_l^2$~~ $(\Delta_{\text{in}}, \Gamma_{\text{out}})$ 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~~ PNB-Based Key-Recovery:

~~Here~~ In this section, we describe the PNB-based ~~key recovery~~ key-recovery for full R-round ChaCha. First, in the offline phase, the attacker collects the PNBs with good backward ~~bias~~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 ~~bias~~ 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 ~~Bias~~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 ~~put~~ assign random values to the PNBs and keep the rest of the bits ~~the same~~ 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 ~~the backward bias~~ 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~~ PNB-approximating function.

- **Online Phase**

➢ **~~Key Recovery~~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 as~~ 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~~ 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 ~~bias~~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 ~~that strengthened the~~ 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 —~~ChaCha-*e.g.*, the 7-round attack is 2^{40} times faster than the prior state of the art~~—and~~. It delivers the first successful 7.5-round attack with a measurable advantage over exhaustive

search, providing ~~a principiled~~ an alternative to PNB-based techniques which does not rely on experimentally determined correlations.

3.1 Revisiting the Previous Works on the Backward ~~Bias~~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 ($\cancel{Z - X}$$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 ~~bias~~ 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 ~~bias~~ 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 ~~bias~~ 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 ~~that~~ 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 ~~$Z = \tilde{X}$ and $Z' = \tilde{X}'$~~ $Z \boxminus \tilde{X}$ and $Z' \boxminus \tilde{X}'$, the matrices $\tilde{Y}$ and $\tilde{Y}'$ are obtained. The Backward ~~bias~~ correlation value is obtained using the same procedure as explained in Subsubsection 2.2.2. The backward ~~bias~~ 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 ~~bias~~ correlation value is observed and is denoted by ε_i. There exists a relation between the backward ~~bias~~ correlation value of the $\Delta_{\mathcal{OD}_i}$'s and each $\mathcal{OD}_i$ bit. As mentioned in the ~~key recovery~~ key-recovery part, the correlation between the forward ~~bias~~ 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

$$\varepsilon = \varepsilon_d \cancel{\cdot} \varepsilon_a \cancel{\cdot \oplus} \prod_{i=1}^{k} \varepsilon_i$$

Starting from the work of ~~[AFK+08], to perform the attack, hypothesis testing is used, where the null hypothesis and alternative hypothesis are as follows:~~ [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 : The guess of significant bits is not correct. H_1 : The guess is correct.~~

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

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

~~If s is the size of the significant bit set S, then out of 2^s possible guesses , only one guess is correct. Hence,~~ Since $|S| = s$, there are 2^s possible guesses for $\widehat{S}$, of which exactly one satisfies H_1 and the remaining $2^s - 1$ ~~guesses satisfy the null hypothesis~~ satisfy H_0~~, and only one guess satisfies the alternative hypothesis H_1. In the hypothesis testing, when guessing the value, there are two possible errors as given~~. Given a fixed decision rule based on the measured correlation (or test statistic), two error events can occur:

1. ~~Error of Non-detection: The selected variable is correct but not detected~~**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 ~~$\Pr_{nd}$~~ $\Pr_{\mathrm{nd}}$.

2. ~~Error of False Alarm: An incorrect variable is chosen because it gives significant bias~~**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 ~~the event is $\Pr_{fa}$~~this event is $\Pr_{\mathrm{fa}}$.

In our analysis, we ~~consider the probability of a false alarm as $\Pr_{fa} \leqslant 2^{-\alpha}$ and the probability of~~require $\Pr_{\mathrm{fa}} \lesssim 2^{-\alpha}$ and denote the non-detection ~~is given as $\Phi^{-1}[\Pr_{nd}] = 0.8.$~~

~~As mentioned in [Dey24], using the Neyman-Pearson Decision theory~~probability by $\Pr_{\mathrm{nd}}$. Using the Neyman–Pearson decision framework, the required number of samples N to achieve ~~a bound on these probabilities is~~

these bounds is approximated by

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

~~To compute time complexity, [Dey24]proposed a formula~~In [Dey24], a formula for computing the attack complexity was proposed, which was ~~later modified~~ subsequently refined by Sharma *et al.* [SDSM25]~~is given as~~. 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, ~~even a single-bit modification can ripple unpredictably through an entire computation. This chaotic spread of differences—often due to carry propagation strengthens security by increasing diffusion. To understand the scenario~~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 ~~the following operations:~~ 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:

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

~~Now, let's analyze the difference between $w, \bar{w}$, i.e., $(w \oplus \bar{w})$ and compare it with the difference between $x, \bar{x}$, i.e., $(x \oplus \bar{x})$.~~ 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 $\mathcal{I}$, ~~i.e., $w \oplus \bar{w}$ can have non-zero values only within segment $\mathcal{I}$. But subtractionbehaves differently:~~ 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 ~~seek constraints ensuring that modifications in~~ establish sufficient conditions ensuring that any modification within the segment $\mathcal{I}$ ~~remain confined within $\mathcal{I}$, leading to~~ 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 $[n_2 : n_1]$ ~~of y are arbitrarily changed to produce $\bar{y}$~~. If the following conditions are satisfied:*

~~*a)*~~

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

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

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

(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. $z[t] \geqslant y[t], z[t] \geqslant \bar{y}[t] \; \forall \; t \in \{4, 5, 6\}$.

~~Also $z[6] \geq y[6]$, $z[5] \geq y[5]$, $z[4] \geq y[4]$, $z[6] \geq \bar{y}[6]$, $z[5] \geq \bar{y}[5]$ and $z[4] \geq \bar{y}[4]$. That is $z[t] \geq y[t], z[t] \geq \bar{y}[t] \; \forall \; t \in \{4,5,6\}$.~~

Then according to~~the Lemma 1~~ 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 , if we apply the two conditions fromLemma 1 for the bit segment $[6:4]$, the value $(z \boxminus y) = (z \oplus y)$~~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

~~To execute the *carry-lock*,~~ The application of the carry-lock method requires that the two conditions ~~given~~ established in Lemma 1 ~~are to~~ be satisfied for ~~$Z_w \boxminus X_w$~~ both subtraction operations $Z_w \boxminus X_w$ and $Z'_w \boxminus X'_w$. ~~Note that we are specifically interested in executing the *carry-lock* when the~~ Crucially, we aim to enforce these conditions specifically in the case where the attacker's guess of the ~~key is correct (significant bits)~~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 ~~$\hat{X_w}$, $\hat{X'_w}$~~ $\widehat{X}_w$, $\widehat{X'}_w$. Let us examine what criteria between ~~$Z_w, Z'_w, \hat{X_w}$~~ $Z_w, Z'_w, \widehat{X}_w$ 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 ~~$Z_w, \hat{X}_w$ and $Z'_w, \hat{X'}_w$~~ $Z_w, \widehat{X}_w$ 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 ~~$\hat{X}_w$, $\hat{X'}_w$~~ $\widehat{X}_w$, $\widehat{X'}_w$, 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 ~~$\hat{X}_w[i_1 - 1 : 0]$~~ $\widehat{X}_w[i_1 - 1 : 0]$. i.e., ~~$Z_w[i_1 - 1 : 0] \geqslant \hat{X}_w[i_1 - 1 : 0]$ and $Z'_w[i_1 - 1 : 0] \geqslant \hat{X'}_w[i_1 - 1 : 0]$.~~ $Z_w[i_1 - 1 : 0] \geqslant \widehat{X}_w[i_1 - 1 : 0]$ 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 $[n_2 : n_1]$. ~~It reduces the probability of a difference between propagation beyond~~ In comparison to the method proposed in [AFK+08], this approach yields a lower probability of difference propagation beyond the n_2-th bit ~~, compared to the approach of [AFK+08]~~ 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.

Table 3: ~~Expected Propagation Distance~~ Experimentally observed average propagation distance (in bits) for ~~Prior Work~~ prior work versus ~~Our~~ *carry-lock* ~~Method~~method.

Technique	**Expected ~~Propagation~~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**

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

We define a random variable ~~denoting the expected number of differences between $z-y$ and $z-\bar{y}$~~ 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 ~~estimate $E(X)$, i.e., the expected number of bits up to which the difference (between $z-y$ and $z-\bar{y}$) propagates beyond the~~ empirically estimate the expected value $\mathbb{E}[X]$ by computing the average 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 ~~bit.~~ 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) $\cancel{z[n_2 : n_1] \geq y[n_2 : n_1]}$ $z[n_2 : n_1] \geqslant y[n_2 : n_1]$ *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* $\cancel{z[n_2 : n_1] \geq \bar{y}[n_1 : n_1]}$ $z[n_2 : n_1] \gtrsim \bar{y}[n_1 : n_1]$*,*

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

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

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 $\cancel{z[n_2 : n_1] \leq y[n_2 : n_1]}$ $z[n_2 : n_1] \leqslant y[n_2 : n_1]$ 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.

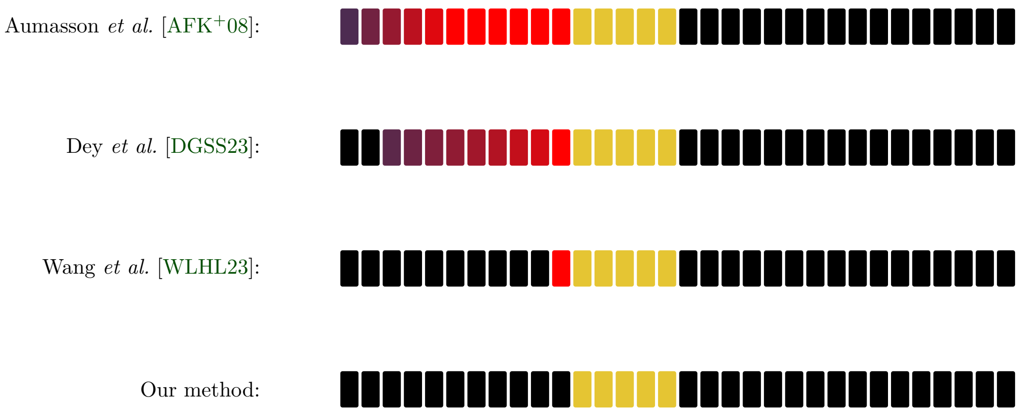


Figure 2: Comparison of the Difference Propagation.

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 ~~Bias~~ correlation of PNBs

In a PNB-based ~~key recovery~~ key-recovery attack, the identification of PNBs with strong backward ~~bias~~ 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 ~~for the average-case scenario~~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~~ Using the Carry-Lock Method

~~As discussed, due to the introduction of *carry-lock* in a specific segment, the modular subtraction behaves like bitwise XOR in that segment. Because of this fact, we further observe that if the attacker can introduce the carry lock in a segment of significant bits, then instead of guessing all the key bits, the attacker can guess some linear combinations of bits directly, reducing~~

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 ~~possible guesses. Let us discuss it in detail.~~ independent guesses.

~~The full key space of the cipher is given by $\mathcal{K} = \mathbb{F}_2^{|K|}$, where K is the secret key, and $|K|$ represents its length in bits. We assume to have m significant key bits. Then these m key bits generate the space $\mathcal{G} \subset \mathcal{K}$.~~

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 ~~online phase, an attacker with an initialization vector v generates a keystream Z. In the offline phase, using the initialization vector v and a guess element $g \in \mathcal{G}$~~next step, the attacker ~~constructs the guessed state $\hat{X}$ by putting random values into the $(|K| - m)$ PNBs. The correctness of g is verified by analyzing the state after reaching $\mathcal{OD}$ from backward by applying reverse rounds on $Z \boxminus \hat{X}$.~~ 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

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

~~In a `quarterround` of ChaCha~~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 ($b[31 : n_2 + 1]$), ~~a vector (a, b, c, d) is updated using the ARX operations to (a', b', c', d'). Each step updates one word by adding, rotating, or XOR-ing it with another. For example, we have the following two equations, which update~~ middle ($b[\mathcal{I}]$), right ($b[n_1 - 1 : 0]$) (check b ~~and c to b' and c' respectively,~~

$$\cancel{c' = c \boxplus d; \qquad b' = (b + c') \lll l}$$

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).

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

~~For instance, Figure 3captures a fragment of ChaCha's `quarterround` which updates b and c.~~

~~An attacker with access to the keystream Z and by guessing state $\hat{X}$ can create the state $Z \boxminus \hat{X}$. Now this state will run through reverse rounds. We denote the vectors of the state $Z \boxminus \hat{X}$ for any `quarterround` as $(Z_a \boxminus \hat{X}_a, Z_b \boxminus \hat{X}_b, Z_c \boxminus \hat{X}_c, Z_d \boxminus \hat{X}_d)$. Following~~

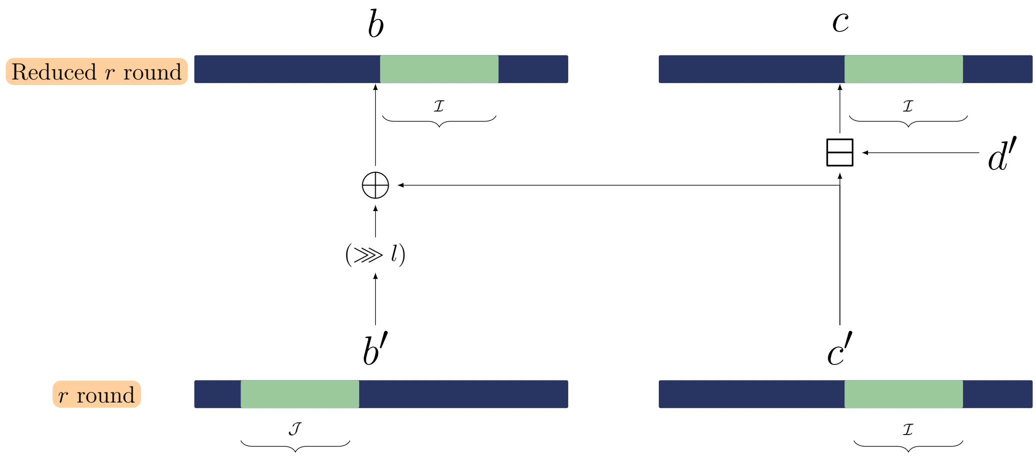


Figure 3: Selection of the ~~Bit-Segment~~ *harmonic pairs* in a `quarterround` of ChaCha.

~~the notation, $b' = Z_b \boxminus \hat{X}_b$ and $c' = Z_c \boxminus \hat{X}_c$. Words b' and c' updates back to b and c upon reverse round application via:~~

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

$$= [(Z_b \boxminus \hat{X}_b) \ggg l] \oplus (Z_c \boxminus \hat{X}_c)$$

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:

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

~~Since each word is 32-bit, we can write,~~ Therefore,

$$b[31:0 = b31 : n_2+1 \| b\mathcal{I}] \| bn_1 - 1 : 0. = (Z_b[\mathcal{J}] \boxminus \widehat{X}_b[\mathcal{J}]) \oplus (Z_c[\mathcal{I}] \boxminus \widehat{X}_c[\mathcal{I}])$$

~~where $\mathcal{I} = [n_2 : n_1]$ is a bit-segment, $\mathcal{J}$ be the rotated version of the bit-segment $\mathcal{I}$.~~

~~From , we have $b[\mathcal{I}] = (Z_b \boxminus \hat{X}_b)[\mathcal{J}] \oplus (Z_c \boxminus \hat{X}_c)[\mathcal{I}]$. Now if we apply carry lock condition on the blocks $\mathcal{I}$ and $\mathcal{J}$ we obtain,~~

$$b[\mathcal{I}] = Z_b[\mathcal{J}] \oplus \hat{X}_b[\mathcal{J}] \oplus Z_c[\mathcal{I}] \oplus \hat{X}_c[\mathcal{I}]$$

$$= (Z_b[\mathcal{J}] \oplus Z_c[\mathcal{I}]) \oplus (\hat{X}_b[\mathcal{J}] \oplus \hat{X}_c[\mathcal{I}]).$$

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.,

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

~~As $Z_b[\mathcal{J}] \oplus Z_c[\mathcal{I}]$ is a public value and known, reveals that generation of $b[\mathcal{I}]$ naively requires guessing both $\hat{X}_b[\mathcal{J}]$ and $\hat{X}_c[\mathcal{I}]$ from $\mathbb{F}_2^{|\mathcal{I}|}$, inducing a guess space of dimension:~~

$$\cancel{\dim(\hat{X}_b[\mathcal{J}] \times \hat{X}_c[\mathcal{I}]) = 2|\mathcal{I}|.}$$

Substituting into (9) gives the identity

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

~~However, from , we can have a critical optimization: instead of independent guesses, we can only guess the combined value:~~

$$\cancel{\hat{X}_b[\mathcal{J}] \oplus \hat{X}_c[\mathcal{I}] \in \mathbb{F}_2^{|\mathcal{I}|},}$$

Therefore, under the carry-lock condition, in order to ~~generate~~ achieve the correct value of $b[\mathcal{I}]$~~. Let us denote $(b'[\mathcal{J}], c'[\mathcal{I}])$ as an *harmonic-pair*. Here, $b'[\mathcal{J}]$ is called the *harmonic counterpart* of $c'[\mathcal{I}]$~~, 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 $\widehat{X}_c[\mathcal{I}]$.

~~It is to be noted that the space $\mathbb{F}_2^{|\mathcal{I}|}$ has dimension $|\mathcal{I}|$, hence the guess space for the generation of~~ Without carry-lock, in the existing approach, computing $b[\mathcal{I}]$ ~~is reduced from $2^{2|\mathcal{I}|}$ to $2^{|\mathcal{I}|}$. The full non-PNB guess space $\mathcal{G}$ has dimension m and this combined guess space has dimension $|\mathcal{I}|$ which yields the new search space $\mathcal{G}_{\text{new}}$ with dimension~~

$$\cancel{\dim(\mathcal{G}_{\text{new}}) = m - |\mathcal{I}| = m - (n_2 - n_1 + 1).}$$

~~This suggests a time complexity improvement by a factor of $2^{n_2-n_1+1}$. We now discuss how we select the harmonic pairs.~~ 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~~

6.2 Identifying harmonic-pair blocks

~~The selection of harmonic pairs is dependent upon the selection of PNBS. To better understand the structure of PNBs, we~~

We define a reduced version of the cipher ~~where we omit the second operation from . In these reduced settings, certain bits of the word c~~ 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.

~~We select a specific block of bits $\mathcal{I}$ from the set of PNBs in the word c in the reduced versionof the cipher. The key observation is that when we transition to the full cipher, these bits undergo the transformation:~~

$$\cancel{b' = (b \oplus c') \lll l.}$$

~~This implies that the bitwise structure of c' interacts with b before rotation. Specifically,~~
Once we find such a block $\mathcal{I} = [n_2 : n_1]$, the ~~block $\mathcal{I}$ from c' is XORed with the corresponding~~

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 $(b'[\mathcal{J}], c'[\mathcal{I}])$

~~Find a~~ Compute the PNB set P' ~~of PNBs from c' from reduced r-round ChaCha~~for the reduced cipher (with the last XOR-rotation update removed);
~~Filter out the common PNBs from P' and form~~ Remove positions common to P and P' to obtain the candidate set P'';
~~Select a PNB block $\mathcal{I}$ from P''~~Choose a block $\mathcal{I} \subseteq P''$;
~~Rotate $\mathcal{I}$ by l-bits to obtain~~ Let $\mathcal{J}$ be the index image of $\mathcal{I}$ under the XOR-rotation update $b' = (b \oplus c') \lll l$;
~~Output $(b'[\mathcal{J}, c'[\mathcal{I}])$ Choosing the Harmonic pair~~ **return**$(b'[\mathcal{J}], c'[\mathcal{I}])$;

~~block from b, and after rotation by l bits, it results in a transformed block, which is~~ *harmonic pair* is $c[\mathcal{I}$ and $b[\mathcal{J}]$, where $\mathcal{J}$ ~~, inside b'. Once the transformation is applied, the block $\mathcal{I}$ becomes a~~ 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 ~~block from c~~ in the full cipher.

~~Apparently, the block $\mathcal{I}$ of b is also non-PNB. Thus, we have two non-PNB blocks , $\mathcal{J}$ and $\mathcal{I}$, originating from b' and c', which form the harmonic pair. describes the way to select $\mathcal{I}$ and $\mathcal{J}$ for r roundChaCha. describes the selection process~~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.

~~Application of Carry-Lock Method in Reducing the Significant Key Space~~

6.3 Carry-lock constraints for significant bits and data cost

~~Consider a significant block $[j_2 : j_1]$ of the key word X_b, has a harmonic counterpart $X_c[j_4 : j_3]$. In order the reduce the significant key space, the criteria of *carry-lock* has to be satisfied for $X_b[j_2 : j_1]$~~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, ~~$X_b'[j_2 : j_1]$, $X_c[j_4 : j_3]$, $X_c'[j_4 : j_3]$. Note that, although the attacker does not know the value of these blocks, they need the conditions of to be satisfied only when the guess of the significant bits is correct. So,~~ the attacker ~~can choose the keystreams based on the values of his guessed value of the keys, as follows:~~

~~Criteria 2Consider a harmonic pair of blocks $X_b[j_2 : j_1]$, $X_c[j_4 : j_3]$, then in order to execute the conditions of *carry-lock* on both the blocks, for any guess $\hat{X}, \hat{X}'$~~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 ~~uses the of output keystream pairs (Z,Z') which satisfy the following conditions:~~ keeps only keystream pairs (Z,Z') satisfying:

1. ~~$Z_b[j_1-1:0] \geqslant \hat{X}_b[j_1-1:0], Z'_b[j_1-1:0] \geqslant \hat{X}'_b[j_1-1:0]$, and $Z_c[j_3-1:0] \geqslant \hat{X}_c[j_3-1:0], Z'_c[j_3-1:0] \geqslant \hat{X}'_c[j_3-1:0]$.~~

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

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

~~In this process, there is an increase in the number of samples required to perform the computation. But as explained thoroughly, the behavior of the bits mentioned above differs from that in . Hence, the number of constraints required in the scenario is different. The step-wise analysis of the data complexity value based on the conditions mentioned in Criteria 2:~~

Expected data filtering cost.

- The first condition consists of ~~4 sub-conditions. Therefore ,~~ four independent $\geqslant$ constraints. Therefore on average, out of $2^4 = 16$ random pairs of output keystreams and a random guess of X, ~~1~~ one satisfies the condition.

- The ~~bit-segment~~ bit segment $\mathcal{I} := [j_2:j_1]$ ~~spans over (j_2-j_1+1) bits. On average, half of those bits of $\hat{X}_b[j_2:j_1]$ are 1. Therefore, by~~ 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 ~~of 2, the averagenumber of conditions required~~ enforces, on average, $|\mathcal{I}|/2$ constraints on $Z_b[j_2:j_1]$ ~~is $\frac{(j_2-j_1+1)}{2}$. By same argument on $Z'_b[j_2:j_1], Z_c[j_4:j_3], Z'_c[j_4:j_3]$, on average, the total number of conditions is $4 \cdot \frac{(j_2-j_1+1)}{2} = 2 \cdot (j_2-j_1+1)$, where each condition is~~. 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 ~~$\frac{1}{2}\cdot$~~ 0.5.

Combining both parts, the expected number of keystream pairs needed to obtain one suitable (Z,Z') pair is

$$E_{D_2} = 2^{\,2\cdot(j_2-j_1+3)}. \tag{12}$$

~~Combining both the parts, we observe that, on average, for any random guess of $\hat{X}, \hat{X}'$, to find one suitable Z, Z' pair , we need to collect $2^{2\cdot(j_2-j_1+1)+4}$ pairs.~~

~~Therefore, the number of samples required to impose conditions on a block of significant bits of size $(j_2 - j_1 + 1)$ is denoted by E_{D_2} and is given as~~

$$\cancel{E_{D_2} = 2^{2\cdot(j_2-j_1+3)}.}$$

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

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

~~As mentioned in~~ In Subsubsection 2.2.1, we mentioned that the maximum limit for the number of samples is 2^{96}. Hence, in the ~~key recovery~~ key-recovery attack, we try to choose the number of samples E_{D_1} and E_{D_2} in such a way that ~~$N^T \leq 2^{96}$~~ $N^T \leqslant 2^{96}$. 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 ~~the modified time complexity value is given as~~ 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~~ 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~~ key-recovery process.

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

Data Collection:

An attacker selects an ~~initialization vector (IV)~~ 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.

~~To provide the differential-linear cryptanalytic~~

Attack on ChaCha

To mount the key-recovery attack, we use the ~~4 round multi-bit differential-linear distinguisher obtained by linearly extending the 3.5~~3.5-round differential-linear distinguisher $\cancel{(X_{13}^{(0)}[6] - \Delta X_2^{(3.5)}[0])}$ ~~given in~~$(\Delta X_{13}^{(0)}[6], \Gamma_2^{(3.5)}[0])$ with correlation $\varepsilon_d = 0.00317$ from [BLT20]. ~~The linear extension results in the following bits~~ $\cancel{(\Delta X_2^{(4)}[0] \oplus \Delta X_8^{(4)}[0] \oplus \Delta X_7^{(4)}[7]))}$ ~~with correlation~~ $\cancel{\varepsilon_l = 1.}$ ~~Using~~ 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 ~~facilitates reducing the time complexity if~~ allows us to lower the overall time complexity when we apply the ~~divide and conquer approach explained by Dey[Dey24]~~ 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 ~~Attack~~ Experimental Results for the attack on ChaCha7.5/256

To ~~deliver an improved attack on~~ improve the attack on the ChaCha7.5~~of the 256-bit key version~~/256, we first ~~obtain the set of~~ collect 15 PNBs by keeping the threshold value ~~$\gamma = 0.4$. As we mentioned in~~ $\gamma = 0.4$. The PNB search was conducted over 2^{20} random state pairs for each key bit position. As discussed in Subsection 5.3, ~~more PNBs can be added to the preliminary set of PNBs to provide a constructive attack.~~

~~The PNBs are obtained by replacing the~~ by replacing $\boxplus$ operation with $\oplus$ in Equation 2 ~~. As mentioned, the additional PNBs will not be as neutral as the Preliminary PNB . Hence, we impose the *carry-lock* Method on these PNBs .~~ 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 ~~bit segments of different dimensions. We obtain 5 such segments~~ the five PNB segments comprising a total of 20 PNBs. The ~~Bias of these elements is mentioned in the last column of the . The remaining 5 PNBs are mentioned in , with the respective bias values. First, we provide the result for the PNB bit segments and the remaining 5 one-bits. Also, we can harmonize some significant key bits to improve the cryptanalysis of ChaCha7.5/256.~~

~~**Key Bit-Segment** Aumasson *et al.* Dey *et al.* Wang *et al.* **This Work**~~ $\cancel{k_2\ [8:6]\ 2^{-1.13}\ 2^{-0.97}\ 2^{-0.88}\ 2^{-0.55}\ [25:22]\ 2^{-2.14}\ 2^{-2.11}\ 2^{-2.05}\ 2^{-0.48}\ k_3\ [11:7]\ 2^{-1.57}\ 2^{-1.35}\ 2^{-1.69}\ 2^{-0.35}\ [27:24]\ 2^{-1.14}\ 2^{-1}\ 2^{-0.93}\ 2^{-0.66}\ k_4\ [30:27]\ 2^{-0.74}\ 2^{-0.68}\ 2^{-0.74}\ 2^{-0.68}}$ ~~Bias Value Comparison of PNB Bit-Segments for ChaCha7.5/256.~~

II. ~~In~~ blocks, along with their correlation, are mentioned Table 4. Here, for each block of PNBs, we compare the ~~bias value for the **PNB bit-segments** obtained for the ChaCha7.5/256. We obtain PNB bit-segments for the keys k_2, k_3, and k_4 as listed in the table. As mentioned, we compare the bias value obtained~~ 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. ~~This depicts~~ 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 ~~is better than the existing attack techniques. In the case of ChaCha7.5/256, we discuss our technique only for the PNB~~ 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_condit` 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 ~~; hence, consists of 20 PNB bits. The bias is obtained using the *carry-lock* method; hence, to find such a pair, we have to impose some conditions . Since~~ (seg_i, $1 \leqslant i \leqslant 5$) of lengths 3, 4, 5~~such bit-segments of~~

~~different dimensions are mentioned in ,~~ , 4 and 4 respectively, the total number of ~~conditions required is 50. Hence, the value $E_{D_1} = 2^{50}$.~~ samples required to apply the *carry-lock* method is $2^{2\times(3+1)} \times 2^{2\times(4+1)} \times 2^{2\times(5+1)} \times 2^{2\times(4+1)} \times 2^{2\times(4+1)} = 2^{50}$.

III. The remaining ~~**5 single PNBs** mentioned below~~ five PNBs with their correlation are mentioned in Table 5~~have a high bias value when evaluated as a preliminary PNB. This implies that there is not much of a requirement to~~. Since we have high correlation values for the single PNBs, we do not apply the *carry-lock* method~~on the PNBs. Hence, the bias value mentioned in the will be considered together with the bias value of~~. Consequently, for all the 25 PNBs, the backward correlation (ε_a) is calculated by multiplying all the correlation values of the bit-segments ~~mentioned in . Therefore, for the PNB list of 25 PNBs, we obtain bias $\varepsilon_a = 0.0330$.~~

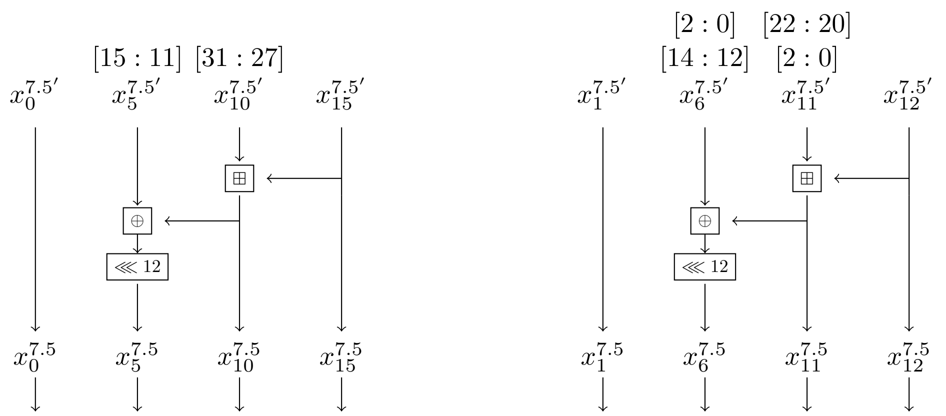
~~Key $k_0 k_1 k_3$ $k_4 k_5 k_6$ k_7 Bit -- 1114 31 31 --- 31 Bias -- 0.780.67 0.83 0.81 --- 0.78 List of Probabilistic Neutral Bits for ChaCha7.5/256.~~ (in *carry-lock* method) and the single PNBs, which is 0.03288.

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).

IV. ~~After examining the additional PNBs and the expected data value, we observe that the additional PNBs can be obtained by harmonizing significant bits as given in~~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` $(X_1, X_6, X_{11}, X_{12})$.

~~We observe that if we consider a bit-segment [31 : 27] of size 5 of significant bits in X_{10}. Corresponding to this bit-segment of significant bitsin X_{10}, we find a bit-segment [11 : 7] of significant bits in X_5. As explained in , instead of separately guessing these bit-segments, we guess the combined value of $(X_5[11:7] \oplus X_{10}[31:27])$. The representation is shown in . Analyzing these bits in this manner will reduce the search space by a factor of 2^5. Some other bits of this type are also mentioned in . Therefore, if we guess the value of the bit-segment $(X_5[11, 7] \oplus X_{10}[31:27])$ during the key-recovery process~~ 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~~. Hence, the data complexity is multiplied by $E_{D_2} = 2^{14}$, as elaborated in .~~

~~Here, two bit-segments of the word X_{11} are guessed together with the corresponding bit-segments of the word X_6. The bit-segment [2 : 0] of X_{11} is guessed together with bit-segment [14 : 12] of the word X_6 and bit-segment [22 : 20] is guessed with bit-segment [2 : 0] of the word X_6. The procedure will result in reducing the number of significant bits~~ Figure 4a. Here, applying the harmonizing trick will give us a significant space reduction by a factor of ~~2^6. However, we have to consider the number of samples we are using to get a guess of both bit segments simultaneously. This results in reducing the dimension of the set of significant bits, but there will be an increase in the data complexity value as explained in .~~ 2^6; however, the resulting samples requirement for this setting is $E_{D_2} = 2^{2\times 8} = 2^{16}$.

V. Combining all the factors, ~~we evaluate the data and time complexity values as discussed in . The PNB set indicates that the total number of PNBs for the attack against ChaCha7.5/256 is 25. Using the attack technique mentioned by Dey [Dey24], we find the PNBs for all the bits involved in the multi-bit output difference position. The three output difference bits are $X_2^{(4)}[0]$, $X_8^{(4)}[0]$ and $X_7^{(4)}[7]$) respectively~~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 ~~bits with their backward bias value are given below:~~ three bits are given in Table 6.

Complexity: ~~For the PNB list of~~ We first obtain 25 PNBs ~~, we obtain bias $\varepsilon_a = 0.0330$ by incorporating our new idea mentioned in . Therefore, as $|K| = 256$, $m = 256 - 25 = 231$. During this computation, we use the idea of harmonizing the significant bits and consider a bit-segment $\mathcal{I}_1 := [31:27]$. Hence, there is a guess of the combined value of $(X_5[11:7] \oplus X_{10}[31:27])$, i.e., a linear combination of 5 significant bits is guessed as explained in . Hence, the dimension of the set of non-PNBs is reduced~~ utilizing the *carry-lock* technique. Because of the *carry-lock* settings in order to get extra PNB, the data complexity would be multiplied by a factor of 2^{50}. Further, 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) search space is reduced to dimension $m = 256 - 25 - 5 = 226$. To execute this, the data requirement increases by a factor of ~~2^5. Therefore, $\dim(\mathcal{G}_{\text{new}}) = 231 - 5 = 226$.~~ 2^{14}. So, the total data inflation is 2^{64}.

From Table 6, ~~we observe that the PNB set for 3 output difference bits $X_2^{(4)}[0]$, $X_8^{(4)}[0]$ and $X_7^{(4)}[7]$ contains 14, 19 and 15 bits with bias values ϵ_i's 0.72, 0.97 and 0.85 respectively. The work of Dey [Dey24] has thoroughly explained how to exploit the PNBs corresponding to each bit of the multi-bit output differential. As mentioned, the bias value $\varepsilon = \varepsilon_d \times \varepsilon_a \times \prod_{i=1}^{3} \epsilon_i = 0.0032 \times 0.033 \times$~~

~~In , substituting the value of $\Phi^{-1}[\Pr_{nd}] = 0.8$, $\varepsilon = 2^{-13.93}$ and keeping $\alpha = 13.2$ we obtain $N = 2^{31.49}$. To find the time complexity of the attack, we substitute the values $R = 7.5$,~~

Table 6: ~~PNB Sets~~ PNBs for each bit of the ~~Three Output Bits~~ mask in ~~the Attack Against~~ ChaCha7.5/256.

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

~~$r = 4$, $k = 3$, and $\dim(\mathcal{G}_{\text{new}}) = 226$ in~~ 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 ~~. The value of m_i corresponding to the 3 output difference bits $X_2^{(4)}[0]$, $X_8^{(4)}[0]$ and $X_7^{(4)}[7]$ are 212, 207 and 211 respectively. Substituting all these values in , we obtain the value of $C = 2^{245.87}$~~ with the experimentally obtained correlation values.

~~In our computation, while improving the PNB count, we have to consider some conditions on PNBs and significant bits, and hence, there is an increase in the expected value of data that satisfies . The conditions are imposed on five-bit segments fromthe PNB set and one bit segment of significant bits. Therefore, combining all the values, the data complexity value increases by a total factor of 2^{64}; hence, the data and time complexity values are $N^T = 2^{95.49}$ and $C = 2^{245.87}$, respectively.~~

7.2 ~~Attack~~ Experimental result for attack on ChaCha7/128

~~Most cryptanalytic attacks are on the 256-bit key version of ChaCha.~~ The most recent attack on the 128-bit key version of ~~the ChaChamentioned in [Dey24] mentioned an attack up to~~ ChaCha, mentioned in [Dey24] mounted an attack on ChaCha6.5/128. ~~Observing~~ Following that work, we ~~introduce our cryptanalysis idea and are able to provide a feasible~~ carry out the first attack against ChaCha7/128. ~~To introduce the attack on ChaCha7/128, we make a set of 22 PNBs by keeping the~~

Using the threshold ~~value $\gamma = 0.15$ and applying the PNB count improvement idea mentioned in . The set of 22 PNBs , along with their bias value, is mentioned in . Key $k_0 k_1$ Bits~~

~~-- 2-3 7-9 15-21 27-29 31 0 8-9 20-21 31 Bias -- 0.45 0.47 0.63 0.45 0.78 0.69 0.45 0.47 0.83 List of Probabilistic Neutral Bits for ChaCha7/128.~~ $\gamma = 0.15$, 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

~~Using our idea, the backward bias value for these 22 PNBs is 0.0057. This implies the dimension of the non-PNB set is $m = 106$. To obtain the bias values, we imposed the conditions on all the PNB bit segments, resulting in an increase in the expected value of the data required. There are 6 such PNB bit segments of different sizes. We also introduced the conditions on single bit $k_3[0]$ to improve its bias value . The bias values mentioned in are obtained by usingthe~~ From Table 7, we obtain seven bit-segments; we apply the *carry-lock* ~~method explained in . The total number of conditions required is 50. To obtain one such pair of Z, Z' for which the conditions are satisfied, the value of data samples required is given by $E_{D_1} = 2^{50}$. Also, in~~ 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 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, 3, 1, 2, 2 and 2, $E_{D_1} = 2^{2\times(2+1)} \times 2^{2\times(3+1)} \times 2^{2\times(7+1)} \times 2^2$
In this case, we did not ~~harmonize the significant key bits as any pair of significant bits did not provide much improvement to the time complexity value, i.e., there is no such bit-segment $\mathcal{I}_i$, which can be harmonized. Hence, $E_{D_2} = 2^0 = 1$. Therefore, there is no change in the dimension of the non-PNB set, i.e., $\dim(\mathcal{G}_{\text{new}}) = m = 106$.~~ have any advantage using the *harmonizing* technique, hence $E_{D_2} = 2^0 = 1$. As a result $\dim(\mathcal{G}_{\text{new}}) = m = 104$.

~~Also, as mentioned in , we have to find~~ Following a similar approach as of ChaCha7.5/256, here we find out the PNBs for ~~the 3 output difference bits $X_2^{(4)}[0]$, $X_8^{(4)}[0]$ and $X_7^{(4)}[7]$. The bias value ϵ_i is 1 for all three output difference bits. The PNBs for the 3 output difference bits are given below:~~ each of the bits of the mask, as noted in Table 8.

~~Therefore, the bias value $\varepsilon = \varepsilon_d \times \varepsilon_a \times \prod_{i=1}^{3} \epsilon_i = 0.0032 \times 0.0057 \times 1 \times 1 \times 1 = 2^{-15.74}$. Putting $\varepsilon = 2^{-15.74}$, and $\Phi^{-1}[\Pr_{nd}] = 0.8$ in , for $\alpha = 2.1$, we obtain $N = 2^{31.15}$.~~

~~For~~

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, $|K| = 128$, $R = 7$, $r = 4$, $k = 3$, ~~be the number of bits in the multi-bit output differential, and $\dim(\mathcal{G}_{\text{new}}) = 106$. The value of m_i corresponding to the 3 output difference bits $X_2^{(4)}[0]$, $X_8^{(4)}[0]$ and $X_7^{(4)}[7]$ are 80, 86 and 95 respectively. Substituting all these values in , we obtain the value of $C = 2^{126.74}$. After imposing the~~ $\dim(\mathcal{G}_{\text{new}}) = 104$, $m_1 = 71$, $m_2 = 86$ and $m_3 = 93$, hence the time complexity C becomes $2^{125.90}$. Since we use the *carry-lock* ~~method on PNBs, the data complexity value is $N^T = 2^{81.15}$.~~

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

~~Key~~ Keywords	Bits		
	~~$X_2^{(4)}[0]$~~ $\Delta X_2^{(4)}[0]$	~~$X_8^{(4)}[0]$~~ $\Delta X_8^{(4)}[0]$	~~$X_7^{(4)}[7]$~~ $\Delta X_7^{(4)}[7]$
k_0	31	31	-
k_1	~~7-31~~ $[31:0]$	-	-
k_2	-	-	~~10-14, 22-26~~$[14:10]$, $[26:2$
k_3	-	~~10-19, 22-30~~ $[19:10]$, 22, $[30:25]$	-
Count	~~26~~ 33	~~20~~ 18	11
~~Bias~~ correlation (ε_i)	1	1	1

~~As we observe in our computation, the data complexity value is much less than the data limit 2^{96}; more conditions can be added to the PNBs, but there will not be much improvement in the time complexity value. Hence, this is the best possible attack with data and time complexity $2^{81.15}$ and $2^{126.74}$ respectively~~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 ~~bias~~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.