Revamped Differential-Linear Cryptanalysis on Reduced Round ChaCha

Sabyasachi Dey[1], Hirendra Kumar Garai[1], Santanu Sarkar[2], Nitin Kumar Sharma[1]

[1] Department of Mathematics, Birla Institute of Technology and Science Pilani, Hyderabad, Jawahar Nagar, Hyderabad 500078, India
sabya.ndp@gmail.com, hirengarai@gmail.com, sharmanitinkumar685@gmail.com
[2] Department of Mathematics, Indian Institute of Technology Madras, Chennai, India
santanu@iitm.ac.in

Abstract. In this paper, we provide several improvements over the existing differential-linear attacks on ChaCha. ChaCha is a stream cipher which has 20 rounds. At CRYPTO 2020, Beierle et al. observed a differential in the 3.5-th round if the right pairs are chosen. They produced an improved attack using this, but showed that to achieve a right pair, we need 2^5 iterations on average. In this direction, we provide a technique to find the right pairs with the help of listing. Also, we provide a strategical improvement in PNB construction, modification of complexity calculation and an alternative attack method using two input-output pairs. Using these, we improve the time complexity, reducing it to $2^{221.95}$ from $2^{230.86}$ reported by Beierle et al. for 256 bit version of ChaCha. Also, after a decade, we improve existing complexity (Shi et al: ICISC 2012) for a 6-round of 128 bit version of ChaCha by more than 11 million times and produce the first-ever attack on 6.5-round ChaCha128 with time complexity $2^{123.04}$.

Keywords: Stream cipher, ARX, ChaCha, Probabilistic Neutral Bits (PNBs), Differential attack.

1 Introduction

Symmetric key cryptography is an essential element of communication networks that protects data secrecy by using a secret key. Symmetric key cryptosystems have the enormous performance advantage of symmetric primitives, such as tweakable block ciphers, stream ciphers, hash functions, or cryptographic permutations, which is the primary reason for their widespread use. Surprisingly, the only way to trust these ciphers is to run a continuous analysis that constantly updates the security margin. With quantum computers on the horizon in the not-too-distant future, the security of today's ciphers has been called into doubt. While most of the commonly used asymmetric primitives would be destroyed, doubling the key size of symmetric constructions offers the same degree of security when searching for keys exhaustively.

The ARX based designs are of immense interest in cryptography. ARX stands for Addition, Rotation, and XOR, which is a family of lightweight symmetric-key algorithms that are primarily designed with elementary operations: Modular addition ($x \boxplus y$), bitwise constant distance rotation ($x \lll n$)and exclusive-OR (XOR, $x \oplus y$). Despite not having the best trade-off in hardware, inexplicit security against the renowned linear and differential cryptanalytic tools [5,18], the encryption developed by ARX has a good software efficiency with compact implementation and fast performance in real life.

The concept of ARX is quite old and dates back to 1987 when the block cipher FEAL [21] used it. ARX machinery is used for both block ciphers (e.g., TEA, Speck) and stream ciphers (e.g., Salsa20, ChaCha). Hash functions and MAC algorithms also utilise the ARX machinery. When applying differential/linear attacks, the only nonlinear operation, *viz.* modular addition needs special attention in ARX. Although the linear and differential properties of modular addition have already been studied [23,16,19], their extension up to the last round is not very simple. As this design is very speedy, designers use many rounds to secure it against linear and differential cryptanalysis.

Both Salsa [3] and ChaCha [4] are well-known symmetric stream ciphers based on ARX machinery. These ciphers have attracted researchers for analysis as well as various companies for commercial use [25]. Salsa with 12-rounds was put forward by Bernstein in the year 2005 as a candidate for the eSTREAM [13] project and was shortlisted among the four finalists in its software profile in April 2007. Bernstein later in 2008 introduced ChaCha [4] as a Salsa variant, which aims at speeding up the diffusion without slowing down encryption. The changes from Salsa to ChaCha are designed to improve the diffusion per turn, increasing resistance to differential cryptanalysis while preserving the time per turn [4]. These designs have a total of 20 rounds. These ciphers also have reduced round variants, of 12-rounds for example. Both these ciphers have 256-bit key version and 128-bit key version. For ChaCha, we use the notation ChaCha256 and ChaCha128 to denote the 256-bit key version and 128-bit key version respectively.

ChaCha encryption extends a 256-bit key to 64 bytes keystream. This cipher has a more conservative design than the AES, and the community quickly gained trust in the safety of the code. Unlike traditional stream ciphers, Salsa and ChaCha both use Pseudo Random Functions (PRFs). Google replaced RC4 with ChaCha in their encryption schemes. ChaCha is in one of the cipher suits of the new TLS 1.3. Google has developed a specific encryption solution called Adiantum for lower-cost entry-level handsets. According to Google [25], Adiantum enables it to use the ChaCha stream cipher in a length-preserving mode that incorporates ideas from AES-based proposals such as HCTR and HCH.

List of protocols and software that implement ChaCha are given in [24].

Previous Works: Since ChaCha is a variant of Salsa, we start with the first cryptanalysis of Salsa in 2005. This attack was designed by Crowley [7], where he

Cipher	Rounds	Time	Data	Memory	Ref.
ChaCha128	6	2^{128}	0	0	Brute-force attack
		2^{107}	2^{30}	0	[1]
		2^{105}	2^{28}	0	[20]
		$2^{84.39}$	$2^{38.66}$	0	[our work]
		$2^{81.58}$	$2^{43.59}$	0	[our work]
	6.5	$2^{123.04}$	$2^{66.94}$	2^{31}	[our work]
ChaCha256	7	2^{256}	0	0	Brute-force attack
		2^{248}	2^{27}	0	[1]
		$2^{246.5}$	2^{27}	0	[20]
		$2^{238.9}$	2^{96}	2^{96}	[17]
		$2^{237.7}$	2^{96}	2^{96}	[6]
		$2^{235.22}$	-	-	[10]
		$2^{230.86}$	$2^{48.83}$	0	[2]
		$2^{221.95}$	$2^{90.20}$	$2^{47.31}$	[our work]

Table 1: Known full key recovery attacks.

cryptanalysed the 5-th round reduced version of Salsa using differential attack. He received a reward from Bernstein for this attack. After that, a further improvement was proposed in the next year by Fischer et al. [14], where they extended the attack to the 6-th round.

In 2008 FSE, the same year in which ChaCha was proposed, Aumasson et al. [1] introduced a concept called 'Probabilistic Neutral Bits'(PNBs) to provide a vital step in the cryptanalysis of both Salsa and ChaCha. Their idea used a meet-in-the-middle approach, in which they identified a set of key bits that have less influence over the position of the output difference at some middle round when we go back from the final state. This idea has been explained in detail in Section 3.2. This approach led to the first attack on 8-round Salsa256 and 7-round ChaCha256. Also, they attacked the 7-round Salsa128 and 6-round ChaCha128. Most of the attacks, hitherto on Salsa and ChaCha became dependent on the premise of PNBs. Interestingly, for the next 13 years, even after several improvements in the cryptanalysis, there is no increment of rounds in the attacks against both the ciphers. Some further improvements have been proposed in this direction by Shi et al. [20] using column chaining distinguisher (CCD) for both 128 and 256-bit version of the ciphers. In 2015, Maitra [17] provided the idea of chosen IV cryptanalysis to provide a good improvement in the key recovery of both the ciphers.

Another important contribution in this direction is by Choudhuri et al. [6], where the single-bit distinguisher of a round was extended to a multiple bit distinguisher of a few next rounds using the linear relation. This differential-linear approach provided the first 6-round distinguisher for Salsa and 5-round distinguisher for ChaCha. This contribution resulted in a massive improvement in the key recovery complexity for smaller rounds. However, for higher rounds, it was not that

effective. After this, Dey et al. [10] improved the set of probabilistic neutral bits for those attacks and then in [12] they provided a theoretical justification of the distinguisher of these ciphers.

Another significant step in this direction came in CRYPTO 2020, where Beierle et al. [2] provided the first 3.5-th round single bit distinguisher for ChaCha. Using this, they provided a partial key recovery of 36 bits for 6-round of ChaCha256 with complexity only 2^{77} and improved the complexity for the 7-round ChaCha. This distinguisher was also observed by Coutinho et al. [8] independently. Soon after that, in Eurocrypt 2021, Coutinho et al. [9] provided a set of a few more distinguishers for 3.5-th round and provided a further improvement using one of the distinguishers. However, in recent work of [11] it is demonstrated that the used distinguisher for the key recovery is incorrect, which makes the attack invalid. This makes the result of [2] the best-known attack till date against 7-round ChaCha256, which has a complexity of $2^{230.86}$.

Our Contribution:

This paper significantly improves the existing differential-linear attacks on ChaCha128 and ChaCha256. The contribution of this paper can be divided into several parts.

At CRYPTO 2020, Beierle et al. [2] demonstrated that by minimizing the Hamming weight of the difference matrix after the first round, we could observe a good differential in the 3.5-th round. However, to achieve one right pair (X, X'), which would produce the minimum difference, we need an average of 2^5 iterations. Also, this is applicable for 70% of the keys, and the other ones cannot produce a proper pair. In this direction, we have two contributions.

First of all, we show that if we slightly modify (relax) the criteria of a suitable pair and allow the Hamming weight to be up to 12 (instead of 10), any key can form a right pair. Also, on average, 8.94 iterations are required to achieve one such IV. We use this result to produce an attack on 6-round of ChaCha128.
Next, we show that we can decompose the key space of the input difference column into two subspaces and construct a memory based on one of these subspaces, which would contain for each member of the list at least one IV with some unique property. This would help to reduce the effort of the 2^5 iterations to achieve the right pair.

Our third contribution is based on improving the PNBs. We provide a three-stage strategy to get a good set of PNBs. Using this strategy in the attack of [9] which achieved their result using 74 PNBs, we have been able to reach up to 79 PNBs to get our best attack. Also, we applied this to produce the first attack on 6.5-round of ChaCha128.

We revisit the complexity calculation formula given by Aumasson et al. [1],

which has been followed afterwards by all the works using PNBs on ChaCha and Salsa. We provide a modified formula that gives an accurate complexity. We explain the scenario in which the previous formula is not applicable. We apply the new formula in one of the attacks against 6-round of ChaCha128. We also provide an alternative attack on the 6-round of ChaCha128, which is more effective than the first one.

Coming to the improvements of the results in the existing attacks, we provide an improvement in 7-round of ChaCha256 by $2^{8.91}$ in the time complexity over the attack in [2]. We improve the 6-round of ChaCha128 attack by $2^{19.44}$. Also, we provide the first-ever attack on 6.5-round of ChaCha128. We provide our attack complexities along with the previous attack complexities in Table 1. Overall, for ChaCha256 with 7 rounds, our attack is as follows: Given an encryption device with a 256 bit secret key, we run it on $2^{90.20}$ different initial vectors, collecting $2^{90.20}$ bits of keystream, from which we recover all the key bits (177 bits in the first stage and the remaining 79 bits in the second stage). The total time complexity required for this attack is $2^{221.95}$ and memory required is $2^{47.31}$. Similarly, for the other versions of the cipher also one can find out the attack setting from the Table 1.

Paper Outline: Our paper consists of 10 sections. In Section 1, we have given an introduction about our work and previous works. In Section 2, we describe the detailed structure of ChaCha256 and ChaCha128. Section 3 describes the idea of differential-linear cryptanalysis on ChaCha, briefly revisit the work of [2], introduce the idea of attack using Probabilistic neutral bits and the complexity calculation. In Section 4, we provide our alternative way of choosing the right pair. In Section 5, we explain the modification of complexity estimation. In Section 6, we propose an idea for improving the PNBs. In Section 7, we explain how using the construction of memory, we can reduce the complexity. In Section 8, we present our results for 7-round ChaCha256. Next in Section 9, we present our results on ChaCha128. Section 10 concludes the paper.

2 Structure of ChaCha

First, we take a look at the algorithm of the stream cipher ChaCha. The R-round variant of ChaCha256 is denoted by R-round ChaCha256. The original cipher design is of 20 rounds.

The cipher works on sixteen 32-bit words and generates a sequence of 512-bit keystream blocks. The ChaCha function takes a 256-bit key $k = (k_0, k_1, \ldots, k_7)$, a 96-bit nonce (number used once) $v = (v_0, v_1, v_2)$ and a 32-bit counter $t = t_0$ as input and produces the keystream blocks Z. There is also a variant of ChaCha which takes 128-bit keys as input. In that case the key is extended to 256-bit by simply putting the key twice. The operations of this cipher includes XOR ($\oplus$), left rotation ($\lll$) and modulo 2^{32} addition ($\boxplus$). Naturally the r-th keystream block,

$0 \leqslant r \leqslant 2^{64} - 1$ of the ChaCha function is dependent on the eight keywords, three nonces and one counter word.

The ChaCha function works on the matrix which consists of 16 words (32-bit bit-string) arranged in the form of 4×4 matrix. Among the 4 rows the first row is a constant string "expand 32-byte k" which is cut into 4 words or constants c_0, c_1, c_2, and c_3, next two rows have the 8 key words $(k_0, k_1, \ldots, k_7)$ of key k and the last row has 1 block counter t_0 and 3 nonces $v_0, v_1,$ and v_2 (for 256-bit key structure). For 128-bit key structure, 4 keywords make a copy of itself and fill up the matrix's second and third row. The four constants for 256-bit key structure are $c_0 =$ `0x61707865`, $c_1 =$ `0x3320646e`, $c_2 =$ `0x79622d32`, $c_3 =$ `0x6b206574`. There is a slight change in the constants for 128-bit key structure. The four constants for 128-bit key structure are $c_0 =$ `0x61707865`, $c_1 =$ `0x3120646e`, $c_2 =$ `0x79622d36`, $c_3 =$ `0x6b206574`. The matrix looks as follows.

$$X = \begin{pmatrix} X_0 & X_1 & X_2 & X_3 \\ X_4 & X_5 & X_6 & X_7 \\ X_8 & X_9 & X_{10} & X_{11} \\ X_{12} & X_{13} & X_{14} & X_{15} \end{pmatrix} = \begin{pmatrix} c_0 & c_1 & c_2 & c_3 \\ k_0 & k_1 & k_2 & k_3 \\ k_4 & k_5 & k_6 & k_7 \\ t_0 & v_0 & v_1 & v_2 \end{pmatrix}.$$

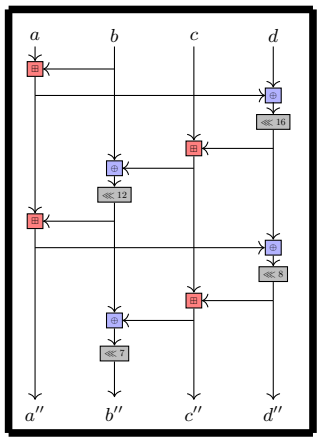

Fig. 1: One quarterround function in ChaCha

In ChaCha, the $\texttt{Round}_R$ function is a nonlinear operation (`quarterround` function) which transforms a vector (a, b, c, d) into (a'', b'', c'', d'') via an intermediate vector (a', b', c', d') by successively calculating (see Figure 2).

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

For an initial state matrix X, $X^{(r)}$ is determined after r-rounds, and rounds are counted from 1, reforming X after every round. Now in the odd rounds the `quarterround` function acts on the four columns of X, *viz.* (X_0, X_4, X_8, X_{12}), $(X_1, X_5, X_9, X_{13}), (X_2, X_6, X_{10}, X_{14})$, and $(X_3, X_7, X_{11}, X_{15})$. That is why the odd number rounds are also called `column rounds`. In the even number of rounds the nonlinear operations of `Round` function is applied to the four diagonals $(X_0, X_5, X_{10}, X_{15}), (X_1, X_6, X_{11}, X_{12}), (X_2, X_7, X_8, X_{13})$, and (X_3, X_4, X_9, X_{14}). Consequently these rounds are called `diagonal rounds`.

Finally the keystream block Z after the R-rounds is computed as $Z = X^{(0)} + X^{(R)}$, where $X^{(0)}$ is denoted as the initial state and $X^{(R)}$ is the state after R-rounds of X. Every round of ChaCha is reversible. From any round $(r+1)$, $X^{(r)}$ can be obtained by operating the reverse quarterround on $X^{(r+1)}$. Interested readers can refer to [4] for more design details.

We provide the quarterround function in a diagram form in Figure 1.

Symbol	Description
X	The state matrix of the cipher consisting of 16 words
$X^{(0)}$	Initial state matrix
$X^{(r)}$	State matrix after application of r-round functions
X_i	i-th word of the state matrix X
$X_i[j]$	j-th bit of i-th word in matrix X
$x \boxplus y$	Addition of x and y modulo 2^{32}
$x \boxminus y$	Subtraction of x and y modulo 2^{32}
$x \oplus y$	Bitwise XOR of x and y
$x \lll n$	Rotation of x by n bits to the left
$x \ggg n$	Rotation of x by n bits to the right
$\Delta X_i^{(r)}[j]$	XOR difference after r-th round of the j-th bit of the i-th word of X and X'
$\dim(K)$	Dimension of the space K
K_{mem}	Set of significant bits/Non-PNBs
K_{nmem}	Set of PNBs
$\mathbb{K}_{mem}$	Key Space corresponding to K_{mem}
$\mathbb{K}_{nmem}$	Key Space corresponding to K_{nmem}
ChaCha256	256-key bit version of ChaCha
ChaCha128	128-key bit version of ChaCha
$\mathcal{ID}$ Column	Column in which input difference is given

Table 2: List of symbols

3 Idea of Differential-Linear Cryptanalysis

Before going to the attack let us first observe the adversary model. The adversary has access to the IVs. He can verify a guess by supplying his own key to the algorithm and check the key's validity by running the algorithm. He can analyse the output of the keystream and examine the key guess.

The differential attack was discovered by Biham and Shamir [5] in 1990. This attack was used initially on block ciphers, but later, it found significant applications in stream ciphers and hash functions. This is a chosen plaintext attack. In this attack, the output of the cipher is observed on the basis of changes in the input. Let P and P' be two plaintexts along with their ciphertexts C and C' respectively. In a differential attack, a correlation between C and C' might be observed. This correlation, in turn, might be exploited to find out the key.

The linear attack was thought up by Matsui [18] in 1992. In this method, the attacker first constructs linear relations between plaintext, ciphertext and key terms with high bias. He then uses these relations together with known plaintext-ciphertext pairs to find the key.

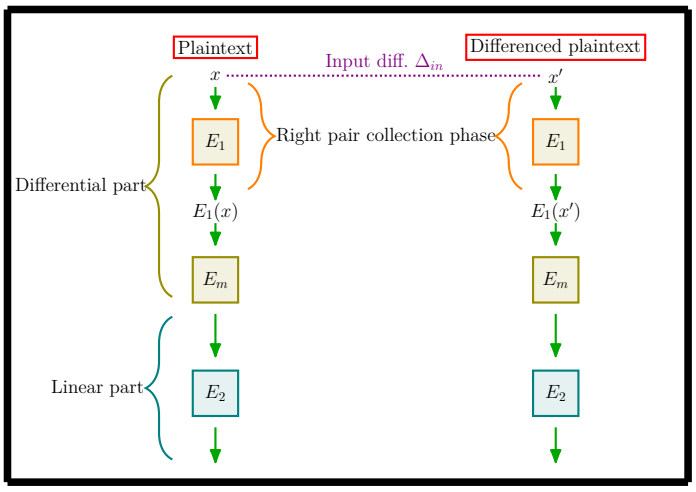

Fig. 2: Differential-linear cryptanalysis

The idea of differential-linear cryptanalysis was introduced by Langford and Hellman [15] in 1994. Here, the cipher E is split into two parts E_1 and E_2 ($E = E_2 \circ E_1$) such that differential and linear cryptanalyses are applied on the subciphers, respectively. Suppose a substantial differential exists in the first subcipher E_1 and linear approximation in the second subcipher E_2. The combination of these two attacks on $E_2 \circ E_1$ is the differential-linear analysis.

See Figure 2. In differential-linear cryptanalysis for each sample, we require two initial states with the same key but different IVs.

Let X and $X' = X \oplus \Delta_{in}$ be two initial states of subcipher E_1 where, Δ_{in} is called input differential ($\mathcal{ID}$). Also ΔX denotes the difference between two states X, X'. The difference after r_1-rounds is observed between the two states. This is called the output differential $\mathcal{OD}$. If a good bias is observed for this, that is exploited to attack the ciphers.

Differential Linear attack in the context of ChaCha: Though these attack methods were initially developed for block ciphers, they have since been successfully applied to stream ciphers. However, the situation is slightly different in the context of stream ciphers. ChaCha being a stream cipher, we explain the scenario in its context. Particularly for ChaCha, where the state can be divided into several words, we denote the j-th bit of the i-th word of X, X' by $X_i[j]$ and $X'_i[j]$ respectively. Here, instead of looking at the output difference of the entire matrices X, X', we choose a particular bit of both the matrices as the position of the output differential. If this is q-th bit of p-th word, then $\Delta X_p[q] = X_p[q] \oplus X'_p[q]$ denotes the difference. Now, instead of plaintexts, we consider the IVs for introducing the input difference. Usually it is given at a single bit of the IV. In the Output difference, we compute the probability of the event $\Delta X_p[q] = 0$. For this probability, the bias is also known as forward bias and is denoted by ϵ_d, i.e. the probability is $\frac{1}{2}(1 + \epsilon_d)$.

After obtaining the output difference after r_1-rounds, we have to look for linear relation on output differential. The linear approximation is observed from r_1-rounds to $(r_1 + r_2)$-rounds of the cipher. The bias for linear approximation is denoted by ϵ_l. Usually for linear approximation, the term correlation is used instead of bias. The linear approximation is observed for both X and X'. So the combined differential-linear bias for $(r_1 + r_2)$-rounds is given as $\epsilon_d \epsilon_l^2$.

3.1 Choosing a Right Pair

In [2], instead of two subciphers E_1, E_2 the authors divided cipher into three subciphers E_2, E_m and E_1 such that $E = E_2 \circ E_m \circ E_1$. Here E_2 is the linear extension as before and $E_m \circ E_1$ construct the differential part. In E_1, they look for a desired difference in $E_1(x) \oplus E_1(x \oplus \Delta_{in})$. Let us denote this intended difference as Δ_m. Now, the underlying aim of this difference is to improve the bias after $E_m \circ E_1$.

The set of initial states X, $X \oplus \Delta_{in} \in \mathbb{F}_2^n$ which satisfy this desired differential Δ_m after the E_1 are called right pairs. Formally, the set of right pairs is defined as $\chi = \{X \in \mathbb{F}_2^n | E_1(X) \oplus E_1(X \oplus \Delta_{in}) = \Delta_m\}$. The differential bias after E_m is found experimentally, assigning the input difference Δ_m before E_m. Particularly for ChaCha, E_1 consists of 1 round and Δ_m is taken as the minimum possible Hamming weight of the difference matrix $X \oplus X'$ after the 1st round, which is 10.

Now, let p be the probability to achieve a right pair when the initial states are randomly chosen, i.e., $\Pr_{X\in\mathbb{F}_2^n}[\ E_1(X) \oplus\ E_1(X \oplus \Delta_{in}) = \Delta_m] = p$. Then the complexity of the attack is computed with the assumption that the initial states are all right pairs. Finally this complexity is multiplied by p^{-1} to get the actual complexity, since on average p^{-1} randomly chosen $(X, X \oplus \Delta_{in})$ pairs give one right pair on average.

In their attack, they mentioned that in a column of ChaCha, there are approximately 30% keys which do not have any IVs to form a right pair. These keys are named as strong keys. For the remaining keys, the probability is $\Pr_{X_i\in\mathbb{F}_2^n}[\ E_1(X_i) \oplus\ E_1(X_i \oplus \Delta_{in}) = \Delta_m] = p \approx \frac{1}{2^5}$.

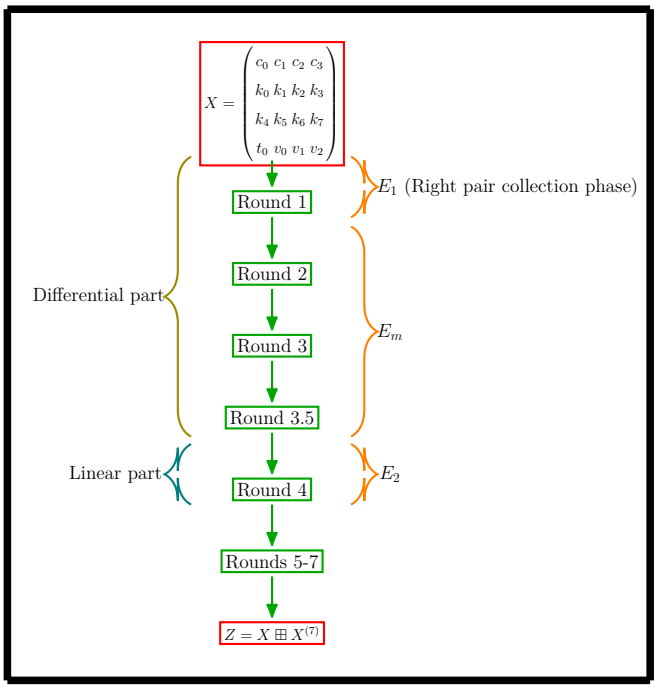

Fig. 3: Differential-linear cryptanalysis of 7-round ChaCha

***$\mathcal{ID} - \mathcal{OD}$* Pair:** In [2] the input difference is $\Delta X_{13}^{(0)}[6] = 1$ and the output difference is observed at $\Delta X_2^{(3.5)}[0]$. Due to column wise symmetric structure one can consider input difference at any of the four columns of X i.e., $\Delta X_{12}^{(0)}[6], \Delta X_{13}^{(0)}[6]$, $\Delta X_{14}^{(0)}[6]$, $\Delta X_{15}^{(0)}[6]$ as they will give same approximations after 3.5-th rounds at $\Delta X_1^{(3.5)}[0]$, $\Delta X_2^{(3.5)}[0]$, $\Delta X_3^{(3.5)}[0]$, $\Delta X_0^{(3.5)}[0]$ respectively.

According to the $E_2 \circ E_m \circ E_1$ approach, for ChaCha E_1 is the first round. As right pairs we consider those initial states for which after the 1st round we obtain

the difference at only the 10 positions shown below:
$\Delta X_1^{(1)}[2]$, $\Delta X_5^{(1)}[5]$, $\Delta X_5^{(1)}[29]$, $\Delta X_5^{(1)}[17]$, $\Delta X_5^{(1)}[9]$, $\Delta X_9^{(1)}[30]$, $\Delta X_9^{(1)}[22]$, $\Delta X_9^{(1)}[10]$, $\Delta X_{13}^{(1)}[30]$, $\Delta X_{13}^{(1)}[10]$.

For E_m the cipher is considered after the first round to 3.5-rounds. For this, the observed forward bias is $\epsilon_d = 2^{-8.3} = 0.00317$ [2] given that the number of differences after the 1st round is 10. In the next stage E_2, a linear relation between 3.5-th round and 4-th round is found as:

$$X_2^{(3.5)}[0] = X_2^{(4)}[0] \oplus X_7^{(4)}[7] \oplus X_8^{(4)}[0] \text{ with } \epsilon_l = 1.$$

In [2], the authors obtained a linear relationship between the 3.5-th round and 5-th round. Nevertheless, as we apply the idea of PNBs, we do not use this relation. So we extend the linear relation only up to the 4-th round with the linear bias of $\epsilon_l = 1$. In our paper, we use the linear combination of bits obtained after 4-th round as our $\mathcal{OD}$ pair. In this paper, input difference ($\mathcal{ID}$) is $\Delta X_{13}^{(0)}[6]$ and output difference ($\mathcal{OD}$) is $X_2^{(4)}[0] \oplus X_7^{(4)}[7] \oplus X_8^{(4)}[0]$.

3.2 Probabilistic Neutral Bits

The differential cryptanalysis of ChaCha is primarily based on the PNB concept, which was first proposed by Aumasson et al. [1] in 2008. We explain the concept assuming a key size of 256. The concept of probabilistic neutrality allows us to divide the secret key bits into two categories: significant key bits (of size m) and non-significant key bits (of size $256 - m$). Significant key bits are those that have a large influence on the position at which the differential is observed. Non-significant key bits, on the other hand, are the polar opposite of significant key bits with a low influence.

To detect the PNBs, we focus on the degree of influence each key bit has on the cipher's output.

If we know the full key, we can compute backwards from a given output to observe the skewed differential bit of round. However, if we do not know the key, we may have to guess it in order to discover the bias. The PNBs' goal is to guess the key in two parts. That is, we make educated guesses about the important bits. That is to say, we guess the key bits that are significant. Even if the remaining key bits are incorrectly predicted, the bias must be visible when we compute backwards bias.

To be more specific, we want to find key bits whose values, if arbitrarily changed, have no effect on the output on a large scale. These neutral bits are thought to have a negligible impact. The traditional method of determining the secret key entails a thorough examination of all 2^{256} feasible possibilities. However, if the remaining $(256 - m)$ bits are PNBs, we can only search over the sub key of size m. As a result, the maximum number of guesses is reduced to 2^m.

In the cipher machinery, after a few rounds, we look for output differentials in the state matrix to use the PNB approach.

General idea of PNB and the formal definition: Let X be the initial state matrix of size 4×4. Assume we have found a suitable non-zero input difference ($\mathcal{ID}$) at the j-th bit of the i-th word in the IV say, $\Delta X_i^{(0)}[j]$. Applying this input difference to X we get another state X'. Running X, X' for r-rounds ($1 \leqslant r < R$) we observe the output difference ($\mathcal{OD}$) at position (p, q), i.e., $\Delta X_p^{(r)}[q]$. The bias is given by ϵ_d, i.e.,

$$\Pr_{v,t}\left[\Delta X_p^{(r)}[q] = 1 \;\middle|\; \Delta X_i^{(0)}[j] = 1\right] = \frac{1}{2}(1 + \epsilon_d), \tag{2}$$

here v, t are IVs and the counter respectively which are considered as random variables.

Keep in mind that after R-round we have the keystream block generated by X as $Z = X + X^{(R)}$. Similarly for initial state X' we have another keystream block after R-rounds, *viz.* Z' we have $Z' = X' + X'^{(R)}$. So, $\Delta X_p^{(r)}[q] = \left(X_p^{(r)} \oplus X'^{(r)}_p\right)[q]$, where the $\oplus$ is the word-wise XOR of the two matrices X and X'.

In this situation, we alter one key bit, say the l-th bit, where $0 \leqslant l \leqslant 255$, of the key. Consequently, we again have two states Y and Y'. Now, we apply the reverse algorithm of ChaCha on $Z - Y$ and $Z' - Y'$ by $(R - r)$ rounds, and achieve the states S and S' respectively. (Since we apply reverse algorithm, we name the S matrices in reverse order).

Let $\Delta S_p[q] = (S_p \oplus S'_p)[q]$. If for the choice of l we have $\Delta S_p[q] = \Delta X_p^{(r)}[q]$ holding with high probability then we assume the key bit to be non-significant. For this, in the initial approach, a threshold value used to be determined, and all the key bits for which the above mentioned probability was higher than the threshold were considered to be non-significant or probabilistically neutral bit.

In that approach, a formal definition for PNB can be given as follows: For a given input difference $\Delta X_i[j] = 1$ and a predetermined threshold value γ, suppose for a key bit l, $\Pr\left[\Delta X_p^{(r)}[q] = \Delta S_p[q] \;\middle|\; \Delta X_i^{(0)}[j] = 1\right] = \frac{1}{2}(1 + \gamma_l)$. Then, l is a PNB if for that particular bit, $\gamma_l > \gamma$.

In the context of PNB, we come across Neutrality Measure, which is defined as γ_l, where $\frac{1}{2}(1 + \gamma_l)$ is the chance that changing the l-th bit of the key does not impact the output. The key bit with neutrality measure 1 does not have any effect on the output, while having a neutrality measure of 0 means that the key bit has high influence. We, in general, create a threshold value of γ, where $0 \leqslant \gamma \leqslant 1$ such that any key bit l with neutrality measure $\gamma_l > \gamma$ is PNB. Naturally, the bigger the γ is, the more negligible effect the PNBs have. We must choose the γ very optimally so that the time complexity is minimised.

3.3 Complexity Computation

In the actual attack the attacker guesses the significant bits and puts random values in the PNBs and run both the states in backward directions. The actual attack algorithm using PNBs as given in [1] is as follows:

1. For an unknown key guess, collect N pairs of keystream blocks, each of which is generated by states with a random nonce and counter (satisfying the relevant $\mathcal{ID}$).
2. For each choice of the m significant key bits do:
   (a) Compute the bias of $\mathcal{OD}$ using the N keystream block pairs.
   (b) Conduct an extra exhaustive search over the $256 - m$ non-significant key bits to test the validity of this filtered significant key bits and to identify the non-significant key bits if the optimum distinguisher certifies the candidate as a potentially correct one.
   (c) If the correct key is found, stop and output the recovered key.

In this attack the bias in the backward direction when the PNBs are assigned random values is called backward bias and is denoted by ϵ_a. Suppose we have m functional or relatively non PNBs, so there is a list of 2^m possible random variables among which only one option is correct, and others are not. We take the null hypothesis H_0 as the selected variable is not correct. Hence $2^m - 1$ variables satisfy H_0 and only one variable satisfies H_1, the alternative hypothesis (the selected is correct). There can be two types of errors in this attack:

1. Non-detection - The selected variable is correct but it is not detected. The probability of this event is $\Pr_{nd}$.
2. The variable selected is incorrect but it gives significant bias, so an incorrect variable is chosen. The probability of the event is $\Pr_{fa}$.

Using the Neyman-Pearson lemma, for $\Pr_{fa} = 2^{-\alpha}$ and $\Pr_{nd} = 1.3 \times 10^{-2}$, required number of samples N to achieve a bound on these probabilities is

$$N \approx \left(\frac{\sqrt{\alpha \log 4} + 3\sqrt{1 - \epsilon_a \epsilon_d{}^2}}{\epsilon_a \epsilon_d} \right)^2 .$$

The complexity of the attack is given by the equation

$$2^m (N + 2^{(256-m)} \Pr_{fa}) = 2^m N + 2^{(256-\alpha)}. \tag{3}$$

4 Modification in the Criteria for a Right Pair

As already mentioned, in [2] the right pair is considered for those key-IVs, for which the Hamming weight of Δ_m is 10 after the first round. It helps to get a

high bias after 3.5-rounds, though the two major drawbacks are the low value of p and unavailability of IVs for 30% of keys. In this context, we dig a little deeper inside the structure and try to find whether some modification is possible so that without significant loss in complexity, the attack can be applicable for all keys.

Now, if we look at the `quarterround` operation, it has four addition operations, out of which the first one does not involve any bit with a difference. So, the minimum Hamming weight is possible if no difference propagates to the next bit in the remaining three addition operations. Now, in the third addition operation $a'' = a' \boxplus b'$, there is a difference at $b'[2]$, whose propagation cannot be fully controlled for any key just by choosing a good IV. Except that, in all the remaining addition operations, the difference propagation can be controlled. However, if we allow a difference propagation to the next bit only (i.e., up to $a''[3]$), proper IVs can control further propagation. This we have proved in the following proposition:

Proposition 1. *In the quarterround function of ChaCha on a tuple (a, b, c, d), for any value of b, c there exists d such that $a'[3] = b'[3]$.*

Proof. As we know (a, b, c, d) denotes elements of m-round and (a', b', c', d') denotes elements of $m + 0.5$-round. Considering the second ARX round of the `quarterround` function we have

$$\begin{aligned} b'[3] &= b[23] \oplus c'[23] \\ &= b[23] \oplus c[23] \oplus d'[23] \oplus Carry_{(c,d')}[23]. \end{aligned}$$

where $Carry_{(c,d')}[i]$ denotes i-th carry bit of the sum of c and $d'(c \boxplus d')$.

Using the first ARX round of the `quarterround` function, $d'=(d \oplus a') \lll 16$. Therefore $d'[23] = d[7] \oplus a'[7]$. This implies

$$b'[3] = \ b[23] \oplus c[23] \oplus a'[7] \oplus d[7] \oplus Carry_{(c,d')}[23].$$

Case 1. If $c[22] = 0$, we make $d'[22] = 0$ by choosing $d[6] = a'[6]$ implies that $Carry_{(c,d')}[23] = 0$. Hence,

$$b'[3] = b[23] \oplus c[23] \oplus a'[7] \oplus d[7].$$

$\therefore$ If we select $d[7] = a[3] \oplus b[3] \oplus c[23] \oplus a'[7]$, we get $b'[3] = a'[3]$.

Case 2. If $c[22] = 1$ and if $d[6]$ is chosen as $a'[6] \oplus 1$, then $d'[22] = 1$ implies that $Carry_{(c,d')}[23] = 0$. So, $b'[3] = b[23] \oplus c[23] \oplus a'[7] \oplus d[7] \oplus 1$. $\therefore$ If we choose $d[7] = a'[3] \oplus 1 \oplus a'[7] \oplus c[23] \oplus b[23]$, we get $b'[3] = a'[3]$.
Thus for both the cases we have $b'[3] = a'[3]$. □

However, if there is a difference in $a''[3]$, then in the immediate next XOR, an extra difference would propagate in $b'[11]$. In the next addition, the difference propagation can be controlled by proper IV.

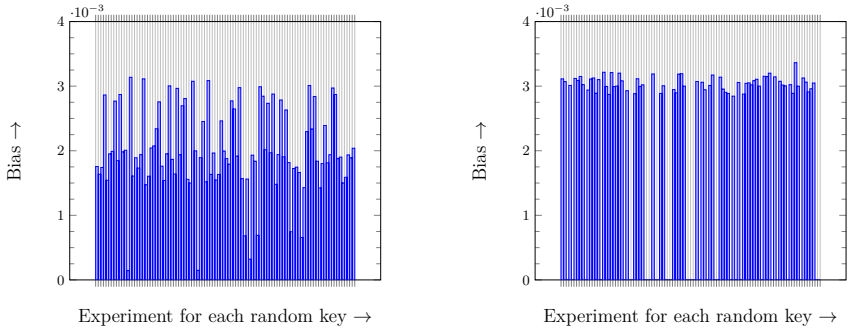

Fig. 4: Comparison of the biases for the case when Hamming weight of Δ_m is 12 (left) vs 10 (right) after the first round. The experiment is done for 100 random keys k_i $(i = 1, 2, \ldots, 100)$. Along x-axis, the i-th vertical line represents the bias corresponding to k_i. For each random key the probability is calculated over 2^{28} random IVs.

Therefore, if we consider the correct pairs with Hamming weight at most 12, we can find IVs for any key that constructs a suitable pair. We have experimentally verified over randomly chosen keys that become a right pair under this relation of the criteria, 100% of the keys become weak. We experimented with randomly chosen 10^6 keys, and for each of them, we selected 4×10^6 IVs randomly. We observed that at least one IV exists for each key, which, after the first round, gives differences at 12 positions at most. In Figure 4, we provide in graphical form the side by side comparison of the biases after 3.5-rounds between the case when the difference is 10 vs the difference is at most 12 after the first round. We give this result for 10^2 randomly chosen keys, and for each of them we experimented over 2^{28} random IVs. Out of these we computed the bias only for those which forms the right pair. It is clearly visible that for Hamming weight $\leqslant 12$ (left), the biases are lower, but it exists for every key. On the other hand, for Hamming weight $= 10$ (right), though the biases are higher, but suitable IVs do not exist for some keys to form right pair. These are represented by blank.

The second benefit of this relaxation is that the probability of achieving the right pair becomes higher, which influence the complexity reduction up to some extent. If we look at the three addition operations where a word with difference is involved, there are five positions where the difference may propagate to the next bit. In each of second and third addition there is one such position and in the last addition there are three such positions. In case of minimum Hamming weight all these five propagation should be restricted. On average, out of 2^5 IVs, on average one satisfies all the five restrictions.

On the other hand, if we allow 12 differences, then the permission to propagate the difference at $a''[2]$ to $a''[3]$ relaxes the restriction. We experimentally observe over 10^5 random keys that this relaxation improves the probability p of achieving a proper IV for right-pair construction to $\frac{1}{8.94} \approx \frac{1}{9}$. In other words, for randomly

chosen 9 IVs, on average, one qualifies to form a right pair. Please note that this probability is computed over any randomly chosen keys, unlike the previous scenario where p has been computed only over the 70% weak keys.

Hamming weight	$= 10$	$\leqslant 12$
ϵ_d	0.00317	0.0021
percentage of weak keys	70	100
probability of satisfying(p)	2^{-5}	$2^{-3.17}$
$p \cdot \epsilon_d^2$	$2^{-21.602}$	$2^{-20.96}$

Table 3: Comparison between the two criteria for right pair on the Hamming weights of the output difference after one round

Besides these two advantages, definitely the apparent disadvantage is that the bias ϵ_d after the 3.5-th round comes down from 0.00317 to 0.0021 (see Table 3). However, p^{-1} decreases from 2^5 to $2^{3.17}$. It is worth noting that the attack complexity is inversely proportional to $p \cdot \epsilon_d^2$. Thus, even though e_d in case 12 is less than in case of 10 difference, the disadvantage does not affect the time complexity because p^{-1} is also less in the first case. Therefore, the product $p \cdot \epsilon_d^2$ is almost same in both the cases. In fact, from the table one can observe that it is slightly higher for the second case ($\leqslant 12$), which will actually improve the complexity.

5 Modification of the Time Complexity Estimation

In the Section 3.3, we have discussed the technique of complexity computation which has been provided by [1]. Here we provide a modification in the formula of the complexity to make it more accurate. This modification gives a better measurement of complexity for any size of PNB.

As we know, only one is correct among 2^m possible sequences of significant keys, so we compute the total number of iterations separately for both the cases.

i. If the guessed key is correct, we run it for N samples to achieve the m significant key bits. Then in the next step, we run an exhaustive search to find n PNBs ($m + n$ = total size of the key), which takes 2^n iterations. So the complexity in this case is $N + 2^n$.

ii. If the guessed key is any of the remaining $2^m - 1$, then for each such guess, we have to run it for N samples first. Assuming the probability of false alarm $= 2^{-\alpha}$, we can say $(2^m - 1) \cdot 2^{-\alpha}$ keys would give false alarm. For these many keys, we perform a thorough search over 2^n sequences of the PNBs. So the

complexity in this case is

$$(2^m - 1) \cdot N + (2^m - 1) \cdot 2^{-\alpha} \cdot 2^n = (2^m - 1) \cdot N + (2^m - 1) \cdot 2^{n-\alpha}.$$

Therefore the total time complexity is

$$\begin{aligned} & N + 2^n + (2^m - 1) \cdot N + (2^m - 1) \cdot 2^{n-\alpha} \\ & = 2^m \cdot N + 2^n + 2^{m+n-\alpha} - 2^{n-\alpha} \\ & \approx 2^m \cdot N + 2^n + 2^{m+n-\alpha} \text{ (Since } 2^{n-\alpha} \text{ is negligible compared to } 2^{m+n-\alpha}) \end{aligned}$$

Since $n = \text{keysize} - m$, we can also write the complexity as

$$2^m \cdot N + 2^{\text{keysize}-m} + 2^{\text{keysize}-\alpha}.$$

Now in the final expression of complexity, we observe an extra $2^{\text{keysize}-m}$ term, which was not there in the formula given by [1]. The reason is since in their attack and all other attacks given so forth against ChaCha, m is significantly higher than α, which makes $2^{\text{keysize}-m}$ a negligible term compared to $2^{\text{keysize}-\alpha}$, therefore it can be ignored.

However, if we arrive in a scenario where m and α are close, i.e., $2^{\text{keysize}-m}$ is not negligible compared to $2^{\text{keysize}-\alpha}$, then we have to consider both the terms in the complexity. On the other hand if m is considerably smaller than α, then instead of $2^{\text{keysize}-m}$, the $2^{\text{keysize}-\alpha}$ can be ignored and the formula becomes $2^m N + 2^{\text{keysize}-m}$.

6 Improving the PNBs

In [2], the authors assigned threshold bias γ to find the PNBs. They received 74 key bits, which gave higher biases than the threshold. This is the conventional method of searching the PNBs, which was given by Aumasson et al. [1]. In this work, we provide a systematic three-step strategy to find a good set of PNBs, which provides better PNBs than the conventional method, and requires significantly less computation than [10].

Stage 1: ***Preliminary Shortlisting and Direct inclusion***

In this stage we shortlist a number of possible candidates for the PNB. This is done by assigning a threshold bias γ_{prelim} and selecting the ones which give a higher bias than the threshold. This step is pretty similar to the conventional method. However, it is not the final set of PNBs. So, in this case, we keep the threshold slightly lower than the conventional method. Suppose n_{prelim} candidates are selected in this step. Among them, we assign a second threshold γ_{direct} which is higher than the previous one. Key bits with higher biases than this threshold are directly included in the PNB set. Let the number of such PNBs is n_1.

Stage 2: ***Selection of the best candidate in each iteration***
Now after the first stage we include some more PNBs (say n_2) as follows: In the i-th iteration we assign random values to the n_1 PNBs selected in Stage 1 and $(i-1)$ PNBs selected in Stage 2. After that, among the remaining $(n_{prelim} - n_1)$ key bits, we alter the value one at a time and observe the backward bias. The key bit with the highest bias is selected as the i-th PNB of Stage 2. This iteration is repeated n_2 times. So we have $(n_1 + n_2)$ PNBs so far.

Stage 3: ***Cherry-pick from the remaining while randomising the selected ones***
The requirement of the third stage comes when the computed biases of the Stage 2 becomes very small. In this scenario, if we further use the technique of Stage 2, not only do we need a significantly huge number of iterations, but also there is a chance of getting the wrong candidate. So, in the third stage, we assign random values to the already selected PNBs in the first and second stage ($n_1 + n_2$ bits) and randomly change one of the remaining shortlisted $n_{prelim} - (n_1 + n_2)$ candidates. Then we compute the backward bias. Up to this, it is similar to Stage 2. Nevertheless, instead of choosing the best one only, we arrange all the candidates in descending order of their biases and choose the required number of PNBs (suppose n_3) from the top.

7 Construction of Memory

Here we propose an alternate method of getting a right pair $\Big((k, v), (k, v')\Big)$ with the use of memory. At first, we partition the set of key bit positions of the $\mathcal{ID}$ column into two subsets K_{mem} and K_{nmem}. We aim to partition them in such a way that for a key, only by looking at the values of K_{mem} bits, we can find out an IV which would construct a right pair for this key. The benefit of this is, we can construct a list containing the possible values of only the K_{mem} bit position and their corresponding IVs. If it can be done, for an attacker it is no more required to run p^{-1} random IVs to get one IV to construct a right pair because based on their guess of key they can look up in the list to find a proper IV.

Issue of PNB: In this approach, the probabilistic neutral bits in the $\mathcal{ID}$ column become a vital issue. Since the attacker puts random values on the PNBs, the favourable IV that the attacker chooses from the list is based on his guess of key. If the actual values at the PNB positions are different from their assigned values, the IV chosen by the attacker from the list may not construct a right pair with the actual key. Suppose k is the actual key and k' is the key guessed by the attacker whose significant bits are correct, i.e., same with k. Now, the IV v which he chooses based on his guess k', may not work for actual key k to

form right pair. In that case, even though the significant bit guess is correct, bias would not be observed.

So, we have to construct our K_{mem} and K_{nmem} subsets in such a way that this problem mentioned above can be solved. The easiest solution to this is to include the PNBs in the K_{nmem} subset. How does it help us we explain it explicitly in Section 7.2.

7.1 Decomposition into Memory and Non-Memory subspace

In the first round all the columns work independently, and the difference propagation is within the input difference column only. So, for convenience, we take our E_1 as the `quarterround` of the column in which the input difference is given. Let $K_{\mathcal{ID}}$ be the set of key bit positions and $\mathbb{K}_{\mathcal{ID}}$ be the key space corresponding to the input difference column. So, $|K_{\mathcal{ID}}| = \dim(\mathbb{K}_{\mathcal{ID}})$. Hence, any key in $\mathbb{K}_{\mathcal{ID}}$ is of the form $(x_0, x_1, \ldots, x_{|K_{\mathcal{ID}}|})$.

Here we introduce one term along with its notation. For any subset K' of $K_{\mathcal{ID}}$, we denote $\mathbb{K}'$ to be the subspace corresponding to K', which we define as the collection of keywords $\{(x_0, x_1, \ldots, x_{|K_{\mathcal{ID}}|})\}$ for which the values corresponding to the bit position which are not in K' are 0, i.e.,

$$\mathbb{K}' = \big\{(x_0, x_1, \ldots, x_{|K_{\mathcal{ID}}|}) \mid x_i = 0 \text{ when } i \notin K'\big\}.$$

Thus the partition of the key bits into K_{mem} and K_{nmem} subsets actually provides a decomposition of the key space into the direct sum of two subspaces. This means, each of the subsets K_{mem} and K_{nmem} corresponds to a subspace of the entire key space of the column. We refer to these subspaces as $\mathbb{K}_{mem}$ subspace and $\mathbb{K}_{nmem}$ subspace.

Then, according to our partition we break the key space into the direct sum of the two subspaces $\mathbb{K}_{mem}$ and $\mathbb{K}_{nmem}$,

$$\mathbb{F}_2^{\dim(\mathbb{K}_{\mathcal{ID}})} = \mathbb{K}_{mem} \oplus \mathbb{K}_{nmem}.$$

Example: Consider a 5 bit key $(x_0, x_1, x_2, x_3, x_4)$. Then the entire key space contains a total 2^5 keys. Now suppose $x_0, x_1, x_2 \in K_{mem}$ and $x_3, x_4 \in K_{nmem}$. Then, $\mathbb{K}_{mem}$ is the collection of keys for which $x_3 = x_4 = 0$, i.e.,

$$\mathbb{K}_{mem} = \big\{(x_0, x_1, x_2, 0, 0)\big\}$$

and $\mathbb{K}_{nmem}$ is the collection of keys for which $x_0 = x_1 = x_2 = 0$, i.e.,

$$\mathbb{K}_{nmem} = \big\{(0, 0, 0, x_3, x_4)\big\}$$

Now a right pair $\Big((k, v), (k, v')\Big)$ in that column should satisfy

$$E_1(k, v) \oplus E_1(k, v') = \Delta_m \text{ where } v' = v \oplus \Delta_{in}. \tag{4}$$

Now for any $k \in \mathbb{K}_{mem}$, $k \oplus \mathbb{K}_{nmem}$ is a coset which contains $2^{\dim(\mathbb{K}_{nmem})}$ elements. We aim to find at least one IV v such that $\forall\ k' \in k \oplus \mathbb{K}_{nmem}, \Big((k', v), (k', v')\Big)$ is a right pair, i.e., for every $k' \in k \oplus \mathbb{K}_{nmem},\ E_1(k', v) \oplus E_1(k', v') = \Delta_m$. In this context we define two terms:
Exploitable key: A key $k \in K_{\mathcal{ID}}$ is a exploitable key with respect to the $\mathbb{K}_{nmem}$ subspace if there exists at least one IV v such that for any key $k' \in k \oplus \mathbb{K}_{nmem}$, $\Big((k', v), (k', v')\Big)$ forms a right pair. It is easy to note that an exploitable key is definitely a weak key.
Favourable IV: For an exploitable key k, an IV v is a favourable IV if for any key $k' \in k \oplus \mathbb{K}_{nmem}$, $\Big((k', v), (k', v')\Big)$ forms a right pair.

Let p_{exp} be the probability of getting a key k which has at least one favourable IV, i.e., among all possible elements of $\mathbb{K}_{mem}$, p_{exp} is the fraction of keys for which such IV exist. We form a list of such keys along with at least one of their favourable IVs. The memory required for this is $p_{exp} \times 2^{\dim(\mathbb{K}_{mem})}$.

Now, we use these exploitable keys in our attack with the help of favourable IVs. So we have to construct the direct sum $\mathbb{K}_{nmem}$ and $\mathbb{K}_{mem}$ in such a way that this p_{exp} is high, i.e., a considerable fraction of the keys are exploitable. Secondly, as the size of the memory required depends on the dimension of $\mathbb{K}_{mem}$, therefore, our aim also is to increase the dimension of $\mathbb{K}_{nmem}$ because it would reduce the memory size. However, it can be understood from the definition of an exploitable key that more the dimension of $\mathbb{K}_{nmem}$, more is the elements in a coset $k \oplus \mathbb{K}_{nmem}$. To become an exploitable key, one IV should construct the right pair with all the keys in the coset. So the fraction of exploitable keys p_{exp} decreases as $\dim(\mathbb{K}_{nmem})$ increases.

7.1.1 Construction of the K_{nmem} subset: Here we provide a heuristic algorithm for how to choose the key bits which would construct a good K_{nmem} subset whose corresponding $\mathbb{K}_{nmem}$ subspace contains a huge fraction of exploitable keys. We begin with the PNBs which are in the input difference column. We call this set $PNB_{\mathcal{ID}}$. We include the PNBs directly into the K_{nmem} subset. For the purpose of this construction we define a temporary subset $K^{temp}_{nmem} \subseteq K_{\mathcal{ID}}$ and the corresponding subspace of $\mathbb{K}^{temp}_{nmem}$. We use this temporary subset to construct the actual K_{nmem} subset. Consider this temporary subset K^{temp}_{nmem} is a variable subset where we assign different elements at different steps according to our requirement.

We start with $K^{temp}_{nmem} = PNB_{\mathcal{ID}}$. Note that, the $PNB_{\mathcal{ID}}$ should be such that for $K^{temp}_{nmem} = PNB_{\mathcal{ID}}$, a huge fraction of weak keys are exploitable keys with respect to the corresponding $\mathbb{K}^{temp}_{nmem}$ subspace. If we don't have such properties for $PNB_{\mathcal{ID}}$, we have to reject some PNBs and start with a smaller PNB set. Now, once we have such a suitable $PNB_{\mathcal{ID}}$, to include further key bits into K_{nmem}, for each of the remaining key bit positions i, we do as follows: We take $K^{temp}_{nmem} = PNB_{\mathcal{ID}} \cup \{i\}$ and consider the corresponding $\mathbb{K}^{temp}_{nmem}$. For a

randomly chosen exploitable key and its favourable IV with respect to $\mathbb{K}_{nmem}^{temp}$ which corresponds to $K_{nmem}^{temp} = PNB_{\mathcal{ID}}$, we find the probability that this key is an exploitable key with the same IV as favourable IV for $\mathbb{K}_{nmem}^{temp}$ where $K_{nmem}^{temp} = PNB_{\mathcal{ID}} \cup \{i\}$. If this probability is higher than some predetermined threshold, we include i into K_{nmem}. We write this in a proper algorithm form in Algorithm 1.

Algorithm 1: Construction of K_{nmem} subset

Input: A set of PNBs in the input difference column ($PNB_{\mathcal{ID}}$), a threshold probability p_{thres}, a collection L of ξ exploitable key-favourable IV combinations for $\mathbb{K}_{nmem}^{temp}$ corresponding to $K_{nmem}^{temp} = PNB_{\mathcal{ID}}$, a counter.

```
for each i ∈ K_ID such that i ∉ PNB_ID do
    counter = 0.
    K_nmem^temp = PNB_ID ⋃ {i}
    for each exploitable key-favourable IV combination (k, v) ∈ L do
        if k is an exploitable key with v as its favourable IV for K_nmem^temp then
            increase the counter.
        end
    end
    if counter/ξ ≥ p_thres then
        include i in the K_nmem.
    end
end
```

7.2 How to Construct the attack

In the PNB based attack, the attacker assigns random values to the PNBs and tries to guess the significant key bits correctly. In our approach, we propose that while guessing the significant key bits, the attacker finds the member from the list of exploitable keys. The list also contains at least one favourable IV for each of the keys. Keep in mind that for each key of the list, we can construct $2^{|K_{nmem}|}$ different keys by changing the values of the K_{nmem} key bits, and the same favourable key would form a right pair with each one of those keys.

In [2], the authors mentioned that instead of having many strong keys which do not form a right pair, we would have with high probability at least one column which has weak keys. Similarly, here if we can keep the percentage of exploitable key high in each column, then with high probability, we will have at least one column in which we have an exploitable key. Let the member be k. As already mentioned, k actually represents a coset $k \oplus \mathbb{K}_{nmem}$. Now, the list also contains at least one favourable IV $v_{\mathcal{ID}}$ for k.

Now, while generating the data for N different IVs, the attackers always use the same $v_{\mathcal{ID}}$ for the input difference column and change the values of the IVs in the remaining columns. According to the structure of ChaCha, we still have 2^{96} different IVs, which ensures that a sufficient number of IVs are always available to get N data samples.

Now according to our construction of K_{nmem} subset, $PNB_{\mathcal{ID}} \subseteq K_{nmem}$. Therefore, the set of significant key bits in the $\mathcal{ID}$ column i.e., Non-$PNB_{\mathcal{ID}} \supseteq \mathbb{K}_{mem}$. Consequently if our guessed non-PNBs/significant bits are correct, this implies that even if the PNBs are incorrectly guessed, the actual key still lies in $k \oplus \mathbb{K}_{nmem}$. Thus we write in a form of a proposition below.

Proposition 2. *If $PNB_{\mathcal{ID}} \subseteq K_{nmem}$, then for any exploitable key k and its favourable IV v with respect to $\mathbb{K}_{nmem}$, if random values are assigned in the bit positions of $PNB_{\mathcal{ID}}$ to form a key k', then v is a favourable IV for k' as well.*

The reason for this is if k, k' is the guessed key and actual key, respectively, both lie in the same coset. Therefore the favourable IV $v_{\mathcal{ID}}$ gives minimum difference after the first round for k' as well. So, if we guess only the non-PNBs correctly, we can find a favourable IV for both the actual key and the guessed key.

Example: Suppose a cipher uses a 10-bit key $(k_0, k_1, k_2, k_3, k_4, k_5, k_6, k_7, k_8, k_9)$ where k_8 and k_9 are PNBs and K_{nmem}=$\{k_7, k_8, k_9\}$. While guessing the significant key bits we look at the list of exploitable keys. Suppose, we choose $(k_0, k_1, k_2, k_3, k_4, k_5, k_6, k_7, k_8, k_9) = (0, 0, \ldots, 0)$ from the list of exploitable keys. Therefore our guess of significant key bits $(k_0, k_1, k_2, k_3, k_4, k_5, k_6, k_7)$ is $(0, 0, 0, 0, 0, 0, 0, 0)$.
Suppose the favourable IV of $(0, 0, 0, 0, 0, 0, 0, 0, 0, 0)$ is v. Therefore v forms a right pair with each key of $(0, 0, 0, 0, 0, 0, 0, 0, 0, 0) \oplus \mathbb{K}_{nmem}$, i.e $(0, 0, 0, 0, 0, 0, 0, 0, 0, 0)$, $(0, 0, 0, 0, 0, 0, 0, 0, 0, 1)$, $(0, 0, 0, 0, 0, 0, 0, 0, 1, 0), \ldots, (0, 0, 0, 0, 0, 0, 0, 1, 1, 1)$. So in the attack, the attacker uses v as its IV. If the guess $(0, 0, 0, 0, 0, 0, 0, 0)$ as the significant key bits is correct then whatever be the actual value of k_8, k_9, v will form a right pair with the key $(0, 0, 0, 0, 0, 0, 0, 0, k_8, k_9)$. So the attacker will achieve the desired bias for the correct guess of the significant key bits.

8 Attack on 7-round ChaCha256

In our simulations, we used the GCC compiler version 9.3.0 and the drand48() function in the programmes for 32-bit random number generation.

In reference to [2] taking the input difference as $\Delta X_{13}^{(0)}[6]$ and obtaining output difference at $\Delta X_2^{(4)}[0] \oplus \Delta X_7^{(4)}[7] \oplus \Delta X_8^{(4)}[0]$, we have bias $2^{-8.3} = 0.00317$. Here we will go 4-rounds forward and 3-rounds backward. In ChaCha256 we apply our proposed strategy from Section 6 in the following manner.

Preliminary Shortlisting of PNB: In the first step we shortlist the preliminary candidates for the PNB by assigning threshold $\gamma_{prelim} = 0.2$. We achieve to-

tal $n_{prelim} = 96$ candidates with higher bias than this. Among this, the bits which are in the input difference column are $\{39, 47, 48, 49, 51, 52, 59, 168, 169, 191\}$.

Construction of K_{nmem}***:*** We apply Algorithm 1 to find the key bits which are from K_{nmem}. For this, we have to start with the PNBs of this column. We observe that if we include all the 9 bits mentioned above, we do not get any favourable IVs for any key. So we reject some PNBs and consider only 5 PNBs in that column which are $PNB_{\mathcal{ID}} = \{39, 47, 48, 168, 191\}$. We assign the threshold $p_{thres} = 0.8$ and find out total 13 bits which gives higher values than p_{thres}. Including these into the K_{nmem} set along with the PNBs we achieve $K_{nmem} = PNB_{\mathcal{ID}} \bigcup \{163, 164, 165, 171, 172, 173, 174, 175, 176, 183, 184, 185, 186\}$.
Therefore, the dimension of K_{mem} is $64 - 18 = 46$. We observe that among all the keys of $K_{\mathcal{ID}}$ approximately 62% are exploitable keys. Please note that only 70% keys are weak keys anyway, so only 8% weak keys do not qualify as exploitable keys. So for each column, we have to construct a list of approximately $0.62 \times 2^{46} = 2^{45.31}$ possible keys along with their favourable IVs. Hence the total memory required is $2^{47.31}$.

Remaining PNB Construction: Among them, by assigning the second threshold as $\gamma_{direct} = 0.45$, we achieve $n_1 = 67$ bits which we include directly in the PNB set. Next, in stage 2, we include $n_2 = 9$ more candidates by choosing the best candidate at each iteration. In stage 3, we assign random values to the chosen 76 PNBs. Now, along with this, we alter the value of one bit at a time from the remaining shortlisted candidates. We choose the best $n_3 = 3$ bits among them. Finally, we have a set of 79 PNBs which we provide below:

{219, 220, 221, 222, 223, 255, 77, 78, 79, 66, 67, 80, 68, 81, 69, 102, 82, 103, 70, 104, 83, 105, 71, 84, 106, 123, 124, 72, 85, 107, 125, 244, 126, 127, 225, 86, 109, 199, 47, 192, 207, 155, 2, 156, 3, 157, 224, 245, 108, 4, 158, 159, 168, 73, 246, 226, 193, 90, 211, 74, 200, 48, 87, 208, 95, 91, 191, 5, 6, 110, 212, 111, 227, 213, 92, 194, 115, 201, 39}.

In Figure 5, we mark the achieved PNBs according to their position in the key. The PNBs selected as direct inclusion (Stage 1), best Candidate in each iteration (Stage 2) and Cherry-picked (Stage 3) are respectively denoted by the colors red, blue and green. Also, for each category, the intensity of the color of the PNB signifies their influence as PNB. For example, among the Stage 1 PNBs (red), highest intensity of red denotes the bit which was included at first into the PNB set, and lowest intensity of red denotes the last (67-th) PNB (in Stage 1). Same is true for Stage 2 and Stage 3 as well.

Complexity: We have the backward bias $\epsilon_a = 0.00057$ for these 79 PNBs. As already mentioned, $\epsilon_d = 0.00317$. For $\alpha = 38.8$, this gives $N = 2^{44.89}$ and time complexity $2^{221.95}$. Now, since we have $2^{45.31}$ exploitable keys in the input difference column, and for each guess we need $N = 2^{44.89}$ data the overall data complexity becomes $2^{44.89} \times 2^{45.31} = 2^{90.2}$.

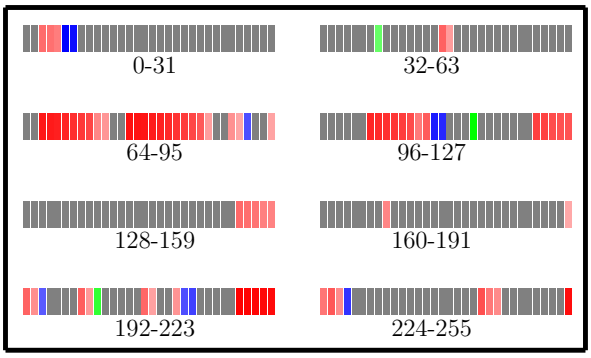

Fig. 5: : Non-PNBs, : Stage 1 PNBs, : Stage 2 PNBs, : Stage 3 PNBs

8.1 Practical observations to confirm the theoretical estimations

To validate statistical assumptions, we performed experiments for 2^{32} many random keys. We take 25 PNBs.

1. In this case we see $\epsilon_a = 0.84822$. Since $\epsilon_d = 0.00317$, the estimated bias is $\epsilon_a \cdot \epsilon_d = 0.00269$. We verify this by experimental observation, which gives the value 0.00266, which is very close to estimated value.
2. Next we take $\Pr_{fa} = 2^{-\alpha}$ for $\alpha = 5.9$. Then, according to the formula, N will be $4749558 = 2^{22.18}$. We assign a fixed value to non-PNBs and random value to PNBs and perform experiment for N times. We need another parameter T to check whether the assigned non-PNBs are correct or not. For more details one can see [22, Section II]. One can check from [22], $\Pr_{fa} = Q\big(|\frac{T}{\sqrt{N}}|\big)$ and $\Pr_{nd} = Q\big(|\frac{2N\epsilon_a\epsilon_d - T}{2\sqrt{N(\frac{1}{4} - \epsilon_a^2\epsilon_d^2)}}|\big)$, where $Q(x) = \frac{1}{\sqrt{2\pi}}\int_x^\infty y^2\, dy$. For $T = 4930$, according to these two error formulas, theoretically we get $\Pr_{fa} < 2^{-\alpha}$ and $\Pr_{nd} < 1.3 \times 10^{-2}$.

   Now, to verify these error probabilities experimentally, we perform our experiment on 2^{14} random keys. We achieve $\Pr_{fa} = \frac{93}{2^{14}} < 2^{-\alpha}$ and $\Pr_{nd} = \frac{7}{2^{14}} < 1.3 \times 10^{-2}$.

Therefore, this experimental verification validates the theoretical estimations.

9 Results on ChaCha128

9.1 Attack on 6.5-round ChaCha128

Now we provide the first-ever key recovery attack on a 6.5-round of ChaCha128. For this we use the memory approach and consider as right pairs those which

have Hamming weight 10 after the first round with input difference at $\Delta X_6^{(0)}[13]$ and output difference at $\Delta X_2^{(4)}[0] \oplus \Delta X_7^{(4)}[7] \oplus \Delta X_8^{(4)}[0]$.

Preliminary shortlisting of PNB: At first we assign a threshold $\gamma_{prelim} = 0.15$ to shortlist the PNBs for the attack. Total 53 bits surpassed γ_{prelim}. Among them there are 5 bits (*viz.* $32, 40, 41, 52, 63$), which comes from the $\mathcal{ID}$ column. Since we have to construct an appropriate K_{nmem} subset containing the PNBs of $\mathcal{ID}$ columns before finalizing the PNBs, we construct the K_{nmem} subspace first.

K_{nmem} ***construction:*** Among the 5 shortlisted PNBs from $\mathcal{ID}$ column, we can only include $63, 40,$ and 41 into our K_{nmem} subset. If we include any of the remaining two, we do not get a high fraction of exploitable keys. We can neither extend this set. The reason for such a small size of K_{nmem} is that in ChaCha128 if we recall the initial state matrix, then we see that X_5 and X_9 are the same, and if any particular key bit is a good candidate for K_{nmem}, then it should be from both X_5 and X_9. In other words, there are very few bits i for which $X_5[i]$ and $X_9[i]$ both are good candidates for K_{nmem}. The key space of ChaCha128 is of dimension 32 only for each column. This helps to make a list of keys of K_{mem} of size 2^{29} only for each column. So, the memory required is 2^{31}.

Final PNB construction: So from our 53 shortlisted PNBs, we reject {32, 40} since those reduces the fraction of exploitable keys. From the remaining 51 bits, at first we directly include 41 bits into the PNB set by assigning threshold $\gamma_{direct} = 0.27$. After that, we use the second step of our strategy to include 2 more PNBs. The third stage is not used here. The list of 43 PNBs is

{7, 12, 13, 14, 26, 27, 28, 29, 30, 31, 40, 63, 64, 65, 66, 71, 72, 79, 80, 83, 84, 85, 86, 91, 92, 93, 94, 95, 96, 97, 98, 99, 104, 105, 115, 116, 117, 118, 119, 124, 127, 87, 15}

Complexity: The forward bias for this PNB set is $\epsilon_d = 0.00317$ as before. The backward bias is $\epsilon_a = 0.0040$. For $\alpha = 8.5$, $N = 2^{37.94}$ the time complexity is $2^{123.04}$. Since there are 2^{29} exploitable keys for the input difference column, the data complexity is $2^{37.94} \times 2^{29} = 2^{66.94}$.

9.2 Attack on 6-round ChaCha128

We provide two attacks on the 6-round ChaCha128 using our PNB search strategy and new criteria for correct pairs. The first attack goes with the usual 2-step technique where we first find the significant bit values and then find the PNB values. In the second attack, we propose an alternative 3-step attack which further reduces the complexity.

9.2.1 First Attack method on 6-round ChaCha128

Preliminary shortlisting of PNB: In the first step, we shortlist the preliminary candidates for the PNB by assigning threshold $\gamma_{prelim} = 0.35$. We achieve $n_{prelim} = 88$ candidates with higher bias than this. Among them, by assigning the second threshold as $\gamma_{direct} = 0.65$, we achieve $n_1 = 73$ bits which we include directly in the PNB set.
Next in step 2, we start with $n_{prelim} - n_1 = 15$ remaining key bits. From there, we include $n_2 = 10$ more candidates by choosing the best candidate at each iteration. We do not go to stage 3 of PNB construction in this attack. So, our total PNB size is $n = 83$. Exploitable keys are unavailable because of the huge number of PNBs with high backward biases in the input difference column. If we reject a significant number of PNBs from this column, the complexity will get affected. This is why go back to the technique proposed in [2] where we randomly guess IVs to achieve the right pairs. However, we choose the criterion of the right pair to be at most 12 differences after the first round. This brings down the forward bias from 0.00317 to 0.0021. However, on the other hand, now, in a column, all the keys are weak.

Complexity: In 6-round ChaCha128 we will go 4 rounds forward and 2-rounds backward. Here, we have forward bias $\epsilon_d = 0.00217$. The PNB set we get is

{0, 1, 7, 12, 13, 14, 15, 16, 17,18, 19, 20, 21, 26, 27, 28, 29, 30, 31, 32, 33, 34, 46, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 79, 80, 81, 82, 83, 84, 85, 90, 91, 92, 93, 94, 95, 96, 104, 115, 116, 117, 118, 119, 120, 121, 122, 127, 35, 86, 22, 123, 47, 8, 105, 97, 109, 2, 39, 64, 40, 65, 71, 98, 124, 87, 36, 110, 48, 41, 9, 23}.

For 83 PNBs we get backward bias $\epsilon_a = 0.025$. Now, on average, we need $p^{-1} = 2^{3.18}$ iterations we achieve the right pair. So, our formula becomes $p^{-1} \cdot (2^m \cdot N + 2^{(128-\alpha)}) + 2^n$. Please note that the p^{-1} does not need to be multiplied with 2^n because this term comes in the second step of only for the case when the significant key bits are correctly guessed, and a favourable IV is achieved to form a right pair. We achieve our best result for $\alpha = 52$. This gives $N = 2^{35.48}$. However, since we have to multiply it by p^{-1}, so the final data complexity is $2^{38.66}$. The time complexity is $2^{84.39}$. If we increase the number of PNBs further, the attack complexity starts increasing. The reason for this is the additional 2^n terms involved in the attack. This affects the overall complexity even though the remaining complexity decreases up to a few more PNBs. We provide in Table 4 the complexities for different sizes of the PNB set.

9.2.2 Alternative Attack on 6-round ChaCha128

Now we propose a slightly modified model of attack. This attack is beneficial when the PNB set size is very high. Instead of finding the significant key bits for a fixed input-output pair, we consider two input-output positions. Due to the column wise symmetric structure of ChaCha, we observe similar biases for the $\mathcal{ID} - \mathcal{OD}$ pairs $(\Delta X_{13}^{(0)}[6],\ X_2^{(4)}[0] \oplus X_7^{(4)}[7] \oplus X_8^{(4)}[0])$ and $(\Delta X_{12}^{(0)}[6],\ X_1^{(4)}[0] \oplus X_6^{(4)}[7] \oplus X_{11}^{(4)}[0])$. we involve both of them into our attack.

PNB size	ϵ_a	α	Complexity
80	0.042	65	$2^{85.39}$
81	0.035	50	$2^{84.84}$
82	0.030	55	$2^{84.46}$
83	0.025	52	$2^{84.39}$
84	0.021	53	$2^{84.65}$
85	0.018	54	$2^{85.26}$

Table 4: Complexities for 6-Round ChaCha for different PNB sizes using the first attack method

Attack Procedure: Suppose the two input-output positions are denoted as $\mathcal{ID} - \mathcal{OD}_1$ and $\mathcal{ID} - \mathcal{OD}_2$. Suppose, we consider m_1 significant key bits and remaining keysize$-m_1$ PNBs for $\mathcal{ID}-\mathcal{OD}_1$. Similarly, for the other input-output difference position, we have a different set of significant bits and PNBs. Among the significant key bits of $\mathcal{ID} - \mathcal{OD}_2$, we consider only those which are not common with the significant key bits of $\mathcal{ID} - \mathcal{OD}_1$. Let there be m_2 such bits. Instead of the usual two-step key recovery technique where we first find the significant key bits and then find PNBs, we propose a 3-step technique. In this, we first find the significant key bits corresponding to $\mathcal{ID}-\mathcal{OD}_1$. Once we achieve it, we go for the remaining m_2 significant key bits corresponding to $\mathcal{ID} - \mathcal{OD}_2$ pair. Once we achieve them, we go for the usual exhaustive search over the remaining bits in the third step. The attack is provided below in an algorithm form in Algorithm 2.

Complexity: For each of the 2^{m_1} possible values of the significant bits, we have to run N_1 pairs. So, we have $2^{m_1} \cdot N_1$ operations. Now, if the probability of false alarm is $2^{-\alpha}$, then for $2^{m_1-\alpha}$ possible wrong keys we receive false alarm. However, if α is considerably higher than m_1, we can ignore the occurrence of any false alarm, i.e., only the right key gives a higher bias than the threshold. In the next step, we go through 2^{m_2} possible key bit values, each run on N_2 samples, which gives a complexity $2^{m_2} \cdot N_2$. Here also, we ignore the occurrence of false alarms since α is high. In the final step, we run an exhaustive search over $2^{128-(m_1+m_2)}$ possible values of non PNBs.
Therefore, if p is the probability of getting a right pair in case of both the $\mathcal{ID} - \mathcal{OD}$ pairs, then the overall complexity $p^{-1}(2^{m_1} \cdot N_1 + 2^{m_2} \cdot N_2) + 2^{128-(m_1+m_2)}$. The data complexity is $p^{-1}(N_1 + N_2)$.
Application on ChaCha128: We consider total 89 PNBs for each $\mathcal{ID} - \mathcal{OD}$ pair. So, we have 39 significant bits for each pair. Instead of listing down the PNBs, we list the significant bits since they are less in numbers.

$\mathcal{ID} - \mathcal{OD}_1$:
$\mathcal{ID} := \Delta X_{13}^{(0)}[6] \quad \mathcal{OD} := (X_2^{(4)}[0] \oplus X_7^{(4)}[7] \oplus X_8^{(4)}[0])$.

Algorithm 2:

Input: N_1 pairs of keystream (Z, Z') corresponding to $\mathcal{ID} - \mathcal{OD}_1$, N_2 pair of keystream (Z, Z'') corresponding to $\mathcal{ID} - \mathcal{OD}_2$.

for *each possible* 2^{m_1} *values of significant key bits of* $\mathcal{ID} - \mathcal{OD}_1$ **do**
  Assign random values to $128 - m_1$ PNBs and run the reverse round by 2 rounds on $Z - X$ and $Z' - X'$.
  if *the bias observed for the* $\mathcal{OD}$ *position higher than the predetermined threshold* **then**
    for *each possible* 2^{m_2} *values of the significant key of* $\mathcal{ID} - \mathcal{OD}_2$ **do**
      Keep the same values of m, significant key bits of $\mathcal{ID} - \mathcal{OD}_1$ pair.
      Assign random values to the PNBs corresponding to $\mathcal{ID} - \mathcal{OD}_2$.
      For N_2 samples of $Z - X, Z'' - X''$ run the reverse round for 2 rounds.
      if *the bias at second* $\mathcal{OD}$ *position is higher than the predetermined threshold* **then**
        perform an exhaustive search over $128 - (m_1 + m_2)$ remaining key bits, and stop if the correct key is found.
      end
    end
  end
end

Significant key bits
{3, 4, 5, 6, 10, 11, 25, 37, 38, 43, 44, 45, 49, 50, 67, 68, 69, 70, 72, 73, 74, 75, 76, 77, 78, 88, 89, 99, 100, 101, 102, 103, 107, 108, 112, 113, 114, 125, 126}

$\mathcal{ID} - \mathcal{OD}_2$:
$\mathcal{ID} := \Delta X_{12}^{(0)}[6] \quad \mathcal{OD} := (X_1^{(4)}[0] \oplus X_6^{(4)}[7] \oplus X_{11}^{(4)}[0])$.

Significant key bits
{5, 6, 11, 12, 13, 17, 18, 35, 36, 37, 38, 41, 42, 43, 44, 45, 46, 55, 56, 57, 67, 68, 69, 70, 71, 75, 76, 80, 81, 82, 94, 99, 100, 101, 102, 106, 107, 120, 121}

One can see that 19 bits are common. So in our notations, $m_1 = 39, m_2 = 20$. We consider α to be significantly higher than m_1. Here, for $\alpha = 50$, we see that, on average, among 2^{50} keys, only one key is there, which will give a false alarm. However, since our total number of guesses is 2^{m_1}, therefore we can ignore the influence of any false alarm. By the same logic, in the second step, also we ignore the false alarm. For 89 PNBs we get $\epsilon_a = 0.0063$ in both the cases. For $\alpha = 50$, $N_1 = N_2 = 2^{39.41}$. However, to get the right pair, we have to go for $2^{3.18}$ random IVs. So, the data complexity is $2^{3.18} \times 2 \times 2^{39.41} = 2^{43.59}$ and the time complexity is $2^{81.58}$.

10 Conclusion

In this paper, we first present an idea to improve the forward bias by the help of list. Also, we show how one can choose a suitable IV to reduce the memory size of this list. Next, we present a new technique to construct probabilistic neutral bit set. This choice gives a significant improvement in the backward bias. As a result, we get around $2^{8.91}$ times better time complexity than [2] for 7-round of ChaCha256. Also, we obtain $2^{23.42}$ times better complexity for 6-round ChaCha128 than the existing work [20], and we report first-time cryptanalysis for 6.5-round ChaCha128. We are very hopeful that our ideas can work on other ARX designs.

Acknowledgement. We are very grateful to Dmitry Khovratovich and EUROCRYPT 2022 reviewers for their detailed comments and helpful suggestions.

References

1. Aumasson, J., Fischer, S., Khazaei, S., Meier, W., Rechberger, C. : New features of latin dances: Analysis of Salsa, ChaCha, and Rumba. In: Nyberg, K. (ed.) FSE 2008, LNCS, vol. 5086, pp. 470–488. https://doi.org/10.1007/978-3-540-71039-4_30
2. Beierle, C., Leander, G., Todo, Y.: Improved differential-linear attacks with applications to ARX ciphers. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, LNCS, vol. 12172, pp. 329–358. https://doi.org/10.1007/978-3-030-56877-1_12
3. Bernstein, D. J. : Salsa20. Technical Report 2005/025, eSTREAM, ECRYPT Stream Cipher Project (2005) https://www.ecrypt.eu.org/stream/papers.html
4. Bernstein, D. J. : ChaCha, a variant of Salsa20 (2008), http://cr.yp.to/chacha.html
5. Biham, E., Shamir, A.: Differential Cryptanalysis of DES-like Cryptosystems. In: Menezes, A., Vanstone, S. A. (eds.) CRYPTO 1990, LNCS, vol. 537, pp. 2–21. https://doi.org/10.1007/3-540-38424-3_1
6. Choudhuri, A. R., Maitra, S.: Significantly improved multi-bit differentials for reduced round Salsa and ChaCha. IACR Trans. Symmetric Cryptol. 2016(2), 261–287 (2016) https://doi.org/10.13154/tosc.v2016.i2.261-287
7. Crowley, P.: Truncated differential cryptanalysis of five rounds of Salsa20. In: SASC 2006 – Stream Ciphers Revisited (2006) http://eprint.iacr.org/2005/375
8. Coutinho, M., Neto, T. C. S.: New multi-bit differentials to improve attacks against chacha. IACR Cryptol. ePrint Arch. 2020, 350 (2020), https://eprint.iacr.org/2020/350
9. Coutinho, M., Neto, T. C. S.: Improved linear approximations to ARX ciphers and attacks against ChaCha. In: Canteaut, A., Standaert, F. (eds.) EUROCRYPT 2021, LNCS, vol. 12696, pp. 711–740. https://eprint.iacr.org/2021/224
10. Dey, S., Sarkar, S.: Improved analysis for reduced round Salsa and Chacha. Discrete Applied Mathematics 227, 58–69 (2017). https://doi.org/10.1016/j.dam.2017.04.034
11. Dey, S., Dey, C., Sarkar, S., Meier, W.:Revisiting cryptanalysis on ChaCha from CRYPTO 2020 and Eurocrypt 2021. https://eprint.iacr.org/2021/1059.pdf

12. Dey, S., Sarkar, S.:Proving the biases of Salsa and ChaCha in differential attack. Des. Codes Cryptogr. 88(9), 1827–1856 (2020). https://doi.org/10.1007/s10623-020-00736-9
13. ECRYPT. eSTREAM, the ECRYPT Stream Cipher Project. See https://www.ecrypt.eu.org/stream/.
14. Fischer, S., Meier, W., Berbain, C., Biasse, J.-F., Robshaw, M. J. B.: Non randomness in eSTREAM candidates Salsa20 and TSC-4. In: Barua, R., Lange, T. (eds.) INDOCRYPT 2006, LNCS, vol. 4329, pp. 2–16.
15. Langford, S. K., Hellman, M. E.: Differential-linear cryptanalysis. In: Desmedt, Y. (ed.) CRYPTO 1994, LNCS, vol. 839, pp. 17–25.
16. Lipmaa, H., Moriai, S.: Efficient Algorithms for Computing Differential Properties of Addition. In: Matsui, M. (ed.) FSE 2001, LNCS, vol. 2355, pp. 336–350. https://eprint.iacr.org/2001/001
17. Maitra, S.: Chosen IV Cryptanalysis on reduced round ChaCha and Salsa. Discrete Applied Mathematics 208, 88–97 (2016) https://doi.org/10.1016/j.dam.2016.02.020
18. Matsui, M., Yamagishi, A.: A New Method for Known Plaintext Attack of FEAL Cipher. In: Rueppel, R. A. (ed.) EUROCRYPT 1992, LNCS, vol. 658, pp. 81–91. https://doi.org/10.1007/3-540-47555-9_7
19. Miyano, H.: Addend dependency of differential/linear probability of addition. IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences 81(1), 106–109 (1998). https://search.ieice.org/bin/summary.php?id=e81-a_1_106
20. Shi, Z., Zhang, B., Feng, D., Wu, W.: Improved key recovery attacks on reducedround Salsa20 and ChaCha. In: Kwon, T., Lee, M.-K., Kwon, D. (eds.) ICISC 2012. LNCS, vol. 7839, pp. 337–351. https://doi.org/10.1007/978-3-642-37682-5_24
21. Shimizu, A., Miyaguchi, S.: Fast data encipherment algorithm FEAL. In: Chaum, D., Price, WL (eds.) EUROCRYPT 1987, LNCS, vol. 304, pp. 267-278. https://doi.org/10.1007/3-540-39118-5_24
22. Siegenthaler, T.: Decrypting a class of stream ciphers using ciphertext only. IEEE Transactions on Computers, 34(1):81–85, (1985). https://doi.org/10.1109/TC.1985.1676518
23. Wallén, J. : Linear Approximations of Addition Modulo 2^n. In: Johansson, T. (ed.) FSE 2003, LNCS, vol. 2887, pp. 261–273. https://doi.org/10.1007/978-3-540-39887-5_20
24. https://ianix.com/pub/chacha-deployment.html
25. https://varindia.com/news/for-the-entry-level-smartphones-google-announced-a-new-encryption-solution--adiantum