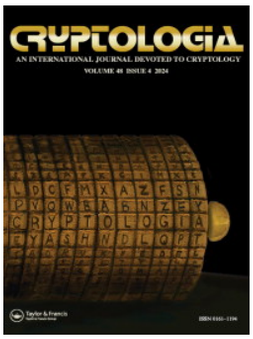

Cryptologia

ISSN: (Print) (Online) Journal homepage: www.tandfonline.com/journals/ucry20

A multi-step key recovery attack on reduced round Salsa and ChaCha

Hirendra Kumar Garai & Sabyasachi Dey

To cite this article: Hirendra Kumar Garai & Sabyasachi Dey (10 Jun 2024): A multi-step key recovery attack on reduced round Salsa and ChaCha, Cryptologia, DOI: 10.1080/01611194.2024.2342918

To link to this article: https://doi.org/10.1080/01611194.2024.2342918

Published online: 10 Jun 2024.

CRYPTOLOGIA
https://doi.org/10.1080/01611194.2024.2342918

A multi-step key recovery attack on reduced round Salsa and ChaCha

Hirendra Kumar Garai and Sabyasachi Dey

ABSTRACT

This paper develops a significantly enhanced attack on the ciphers Salsa and ChaCha. The existing attacks against these ciphers are mainly differential attacks. In this work, we produce an attack on 7.5-round Salsa and 6.5-round ChaCha20. These are the maiden key-recovery attacks on those versions of the two ciphers, in which we recover the key in multiple steps using several distinguishers. In comparison to the previous best-known attack against 7-round Salsa, the new attack method offers an improvement of $2^{7.5}$ times, while on 7.5-round Salsa20 and 6.5-round ChaCha20 our attack is the only existing one.

KEYWORDS

ARX; ChaCha; differential cryptanalysis; key recovery attack; Salsa

1. Introduction

The world of stream ciphers was lacking trustworthy and efficient ciphers in the 2000s. In the search for an efficient yet fast stream cipher, the eSTREAM project emerged. In the third-phase selection, eSTREAM validated the Salsa family of ciphers in 2007, which was submitted by Daniel J. Bernstein in 2005.

Salsa20/12 was one of seven finalists in the eSTREAM project (2005–2008). The Salsa20/20 cipher is appealing for encryption due to its high speed and security. In the next year 2018, Bernstein released a newer version of Salsa—known as ChaCha—by increasing the diffusion in Salsa.

Both Salsa and ChaCha are addition/rotation/XOR (ARX)–based cryptographic primitives. Their keystream generation algorithm comprises three simple operations: Addition modulo 2^{32} ($\boxplus$), constant distance left bit rotation ($\lll$), and bitwise XOR operation ($\oplus$). These operations are swift in any circuit, and hence the cipher can achieve significant speed with a high security margin. Due to their efficient algorithm and fast performance, both ciphers have attracted cryptographic analysis since their release. Most attacks on these two ciphers are of differential and linear attack.

CONTACT Hirendra Kumar Garai hirengarai@gmail.com Mathematics Department, BITS Pilani, Hyderabad Campus.

1.1. Design principles of the Salsa and ChaCha family

The Salsa20 cipher takes a 256-bit key (k), a 128-bit constant (c), and a 128-bit initial vector (v), or IV, and generates a 512-bit keystream. The key is divided into eight 32-bit words $(k_0, k_1, ..., k_7)$. Similarly, the constant and the IV are also broken into 32-bit words (c_0, c_1, c_2, c_3) and (v_0, v_1, t_0, t_1). Due to the use of the 256-bit key, the cipher is also called 256-bit Salsa20. Inputs are stored in a 4×4 matrix as follows:

$$X = \begin{pmatrix} X_0 & X_1 & X_2 & X_3 \\ X_4 & X_5 & X_6 & X_7 \\ X_8 & X_9 & X_{10} & X_{11} \\ X_{12} & X_{13} & X_{14} & X_{15} \end{pmatrix} = \begin{pmatrix} c_0 & k_0 & k_1 & k_2 \\ k_3 & c_1 & v_0 & v_1 \\ t_0 & t_1 & c_2 & k_4 \\ k_5 & k_6 & k_7 & c_3 \end{pmatrix}$$

The round function in Salsa20 is constructed using 4 quarter-round functions, each of which is an ARX function. Together, these ARX functions form the round function. The quarter-round function takes four 32-bit words (x, y, z, w) and updates them to (x', y', z', w') by calculating the following:

$$\begin{aligned} y' &= y \oplus ((x \boxplus w) \lll 7); \\ z' &= z \oplus ((y' \boxplus x) \lll 9); \\ w' &= w \oplus ((z' \boxplus y') \lll 13); \\ x' &= x \oplus ((w' \boxplus z') \lll 18); \end{aligned} \tag{1}$$

The Salsa round function can be broken into two parts. The first half of the round function is formed using the same 4 quarter-round functions, but each quarter-round function is made up of the first two equations of (1). In the same way, the second half of the round function is made using the quarter-round functions constructed by the last two equations of (1).

In each odd-numbered round, the four columns of the matrix X are updated by this quarter-round function; in the even-numbered rounds, this function is applied to the rows of the matrix.

In the original Salsa20 cipher, the number of rounds is 20. After the final round, the initial state X is added modulo (2^{32}) to the updated state X^{20} word-by-word, and the keystream (Z) of 512 bits is achieved, that is, $Z = X \boxplus X^{20}$. This keystream is then bitwise XORed to plaintext to get the ciphertext.

One property of this round function is that we can go backward from any round $r+1$ to its previous-round r; that is, X^{r+1} can be converted to X^r by one reverse Salsa round function (Rev_S^1):$Rev_S^1(X^{r+1}) = X^r$. The reverse Salsa round function similarly consists of 4 quarter-round functions, each of which has ARX operations inside them. For more details, refer to Bernstein (2008). The design of ChaCha mimics the Salsa cipher. ChaCha also takes an input of 512 bits and processes it to a 512-bit keystream. The

input consists of four constants, eight keywords, and four initial vectors. Each of them are 32 bits. They are arranged in a 4×4 matrix that forms the initial state:

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

The matrix is then updated by the ChaCha round function, which is made up of 4 `quarterround` functions. Each `quarterround` function transforms a vector (a, b, c, d) to (a'', b'', c'', d'') in the following way:

$$\begin{aligned} a' &= a \boxplus b; & d' &= ((d \oplus a') \lll 16); \\ c' &= c \boxplus d'; & b' &= ((b \oplus c') \lll 12); \\ a'' &= a' \boxplus b'; & d'' &= ((d' \oplus a'') \lll 8); \\ c'' &= c' \boxplus d''; & b'' &= ((b' \oplus c'') \lll 7); \end{aligned} \tag{2}$$

In case of the ChaCha cipher, in the odd-numbered rounds the columns are updated; in the even-numbered rounds, the diagonals are updated. In the same way as the Salsa cipher, the first half of the ChaCha round functions are defined by the quarter-round functions that use the first four equations of (2). The second half of the round functions uses the next four equations of (2).

After full R ChaCha rounds, the initial state is added to the final state and we have the keystream in the matrix form.

Similar to Salsa, the ChaCha round function is also reversible. Further details can be found in Bernstein (2008).

Table 1. Notations used in this article.

Symbol	Meaning
Salsa20/R	Salsa with R-rounds
ChaCha20/R	ChaCha with R-rounds
$\mathcal{ID}$	Input difference bit
$\mathcal{OD}$	Output difference bit
X	The initial state matrix consisting of 16 words
X'	The state matrix after giving difference to X at $\mathcal{ID}_1$
X''	The state matrix after giving difference to X at $\mathcal{ID}_2$
X^r	The state matrix after r forward Salsa rounds
Rev_S^r	r Reverse Salsa rounds
X_i	i-th word of the state matrix X
$x \boxplus y$	Addition of x and y modulo 2^{32}
$x \oplus y$	Bitwise XOR of x and y
$x \lll n$	Left rotation of x by n bits
(a, b)	The position of b-th bit of the a-th word of the state matrix
S_i	The set of key bits recovered in i-th stage
K_i	Corresponding subkey of S_i.

1.2. Outline and contribution

We have presented here an improved differential attack technique on the Salsa and ChaCha ciphers. We used some extra conditions in choosing the proper IV, which in turn helps improve the existing attack on Salsa20/7. In addition, those choices contributed to the first-ever attack on Salsa20/7.5 and ChaCha 20/6.5. The attack on Salsa20/7 exploits three ($\mathcal{ID} - \mathcal{OD}$) pairs. Here we split the entire key into four subkey sets and recover them one-by-one. This approach significantly improves the complexity by $2^{7.5}$ over the previously existing best attack. The attack on Salsa20/7.5 uses two $\mathcal{ID} - \mathcal{OD}$ s. We also implemented the first-ever attack on ChaCha20/6.5. The paper is organized as follows.

- Section 1 discusses the design principles of the Salsa and ChaCha ciphers.
- In Section 2, we explain the basic outline of the present attack approach on Salsa and ChaCha. In Section 2.1, we report on the existing attacks on Salsa and the correction of complexity in one of the works.
- Section 3 describes our multi-step attack on Salsa20/7 in detail. The attack consists of a preprocessing phase and an online phase. We discuss them respectively in Subsections 3.2 and 3.3.
- In Section 4, we propose a complexity calculation formula for this multi-step attack approach, which is modified from previously existing formulas. We consider the probability of false-alarm errors and compute their influence on overall complexity. After that, using this formula, we compute complexity.
- Sections 5 and 6 address details of the attack on Salsa20/7.5 and ChaCha20/6.5.
- Section 7 offers a conclusion.

2. Cryptanalysis of Salsa and ChaCha

The cryptanalytic techniques used on these ciphers are mostly differential attacks. In a differential attack, the attacker is assumed to have access to

Table 2. Complexities of certain previous key recovery attacks on 256-bit Salsa20/7 and our results.

Cipher	Round	Data complexity	Time complexity	Year	Attack
Salsa	7	–	2^{190}	2007	[Tsunoo et al. 2007]
		2^{26}	2^{151}	2008	[Aumasson et al. 2008]
		2^{61}	2^{149}	2016	[Choudhuri and Maitra 2016]
		$2^{104.47}$	$2^{125.16}$	2023	[Coutinho et al. 2023]
		$2^{96.74}$	$2^{119.74}$	2023	[Our work]
	7.5	$2^{15.54}$	$2^{163.54}$	2023	[Our work]
ChaCha	6.5	$2^{90.74}$	$2^{151.74}$	2023	[Our work]

the IV. As stated by Maitra (2016), the central idea of these attacks is to input differences in the initial state and look for biases in the output. After moving forward as described earlier, we can move backward a few rounds from the final state to get more non-randomness. In a broad sense, the differential attack searches for the high-probability occurrences of ciphertext differences for certain plaintext differences.

In existing attacks against Salsa, an input difference is given in position (i, j) $(\mathcal{ID})$ of the initial state X, which creates X'; after running both r Salsa rounds forward, the attacker look for some high bias at some bit of the output difference matrix $X^r \oplus X'^r$. If such a bias is found at the q-th bit of the p-th word, we call this bit $(\mathcal{OD})$. The corresponding bias is called forward bias and is denoted by ϵ_d. Forward bias can be exploited as a distinguisher of the cipher. In addition, it is used to recover the key, using the concept of probabilistic neutral bits.

2.1. Probabilistic neutral bits

Aumasson et al. (2008) made a major contribution to attacks against the Salsa family of ciphers in FSE 2008, in which they introduced the concept of probabilistic neutral bits. In Salsa20/R, the design principle makes clear that from the final output keystream Z, if we subtract the initial state matrix X (modulo 2^{32}), we achieve the final state X^R. From there, we can reach any state X^r $(r < R)$ by applying the reverse Salsa algorithm. The main idea behind probabilistic neutral bits is to partition all the key bits into two categories based on their influence on the output difference position. The key bits with less influence on the output difference position are categorized as probabilistic neutral bits. The remaining bits that influence the output are called significant bits. This partition helps to determine the key bit values separately, which improves the attack complexity. A detailed technical discussion on probabilistic neutral bits can be found in Aumasson et al. (2008) and Maitra (2016).

2.2. Previous attacks on these ciphers

Salsa was first cryptanalyzed in 2005 by Crowley (2005), who produced a truncated differential attack on Salsa20/5. In 2006, an attack on Salsa20/6 was proposed by Fischer et al. (2006). The following year, this attack was improved upon by Tsunoo et al. (2007), who cryptanalyzed Salsa20/7 with complexity 2^{190}. Then, Aumasson et al. (2008) introduced the idea of probabilistic neutral bits, which helped to improve the complexity of the attack on Salsa20/7 to 2^{151} and also produce an attack on the next round. This work also provided the first cryptanalysis

against ChaCha. Long after, Choudhuri and Maitra (2016) introduced the multiple-bit distinguisher and distinguishers in higher rounds for both Salsa and ChaCha. Despite that they said that the complexity of their attack was 2^{137}, the most recent change imparted to the complexity formula in Dey et al. (2022) indicates that the real run time cost is 2^{149}, which we discuss in the next subsection. This is the best-known result to date. In addition to these, the work of Maitra (2016) and Dey and Sarkar (2017) also cryptanalyzed Salsa and ChaCha. In recent years, several interesting contributions have been made in the cryptanalysis of these two ciphers by Beierle, Leander, and Todo (2020); Coutinho and Neto (2021); and Dey et al. (2022); Dey, Garai, and Maitra (2023); Coutinho et al. (2023).

2.3. Correction of the complexity in the attack of Choudhuri and Maitra (2016)

In the already-existing attacks, the whole key space (k) is divided into two sets—significant key bits (m) and probabilistic neutral bits (n)—and key recovery is made in two steps. The complexity formula initially used in the literature provided by Aumasson et al. (2008) was $2^m \cdot N + 2^{k-\alpha}$, where N is the data complexity and $2^{-\alpha}$ is the probability of a false alarm. A correction of the complexity formula was given later by Dey et al. (2022) and is as follows:

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

Subsequently, Dey, Garai, and Maitra (2023) reported the complexities of several previous attacks against ChaCha as higher than the claim. In the same vein, we report the complexity of the 256-bit Salsa20/7. In the work by Choudhuri and Maitra (2016), the complexity is higher than the claim. They considered 107 significant bits and 149 probabilistic neutral bits in their attack. Thus, the term 2^{k-m} is 2^{149} in the second stage of their attack, 149 probabilistic neutral bits are needed to be exhaustively searched, making the second step's complexity 2^{149}. Their claimed complexity of 2^{137} is thus rectified to 2^{149}.

3. New cryptanalysis of 256-bit Salsa20/7 using multiple (ID−OD) pairs

The primary idea of the multiple ($\mathcal{ID} - \mathcal{OD}$) attack is as follows: In the offline stage, we partition the set of all key bits into four subsets: S_1, S_2, S_3 and S_4. S_1 contains the bits that are significant with respect to ($\mathcal{ID}_1 - \mathcal{OD}_1$). S_2 contains the bits that are significant with respect to ($\mathcal{ID}_2 - \mathcal{OD}_2$) but not significant with respect to ($\mathcal{ID}_1 - \mathcal{OD}_1$). S_3 contains the bits that are not

significant with respect to either $(\mathcal{ID}_1 - \mathcal{OD}_1)$ or $(\mathcal{ID}_2 - \mathcal{OD}_2)$. S_4 contains the rest of the key bits. Now, in the online phase, at the first stage, we recover the values of S_1 key bits with the help of a distinguisher $(\mathcal{ID}_1 - \mathcal{OD}_1)$. Once we achieve these values, we go on to the next stage to recover the values of S_2 key bits, exploiting $(\mathcal{ID}_2 - \mathcal{OD}_2)$. Then, in the third stage, we recover the values of S_3 keybits are recovered using $(\mathcal{ID}_3 - \mathcal{OD}_3)$. The rest S_4 keybits are recovered via exhaustive search. A detailed technical discussion follows.

3.1. New distinguishers and right pairs

At first, we aim to find suitable distinguishers, that is, input difference–output difference pairs. Coutinho et al. (2023) proposed a 5-round distinguisher with input difference at $(7, 31)$ and output difference at $(4, 7)$. Using this approach, the authors used the idea of "right pairs" given by Maitra (2016), in which the author showed that if a suitable choice of IV is made, the difference count after the first round can be restricted to 4, which can significantly improve bias. With the same approach, Coutinho et al. (2023) found the bias of the distinguisher to be $2^{-42.01}$ by using the piling-up lemma on the biases observed for three output difference positions $(0, 0), (4, 7), (12, 0)$.

To find new distinguishers, we take the seventh word of an initial state X and put input differences on each of its 32 bits one-by-one. To make the difference propagate less, we chose the seventh word, as it is the third element of the column vector $(X_{15}, X_3, X_7, X_{11})$. For an input difference in i-th bit $(7,i)$, we get another state X', and then on both the states, we run 1 forward Salsa round, minimizing the number of differences in the first round, and observe the output difference after 4 rounds. We filtered the $(\mathcal{ID} - \mathcal{OD})$ pairs that produce high bias. We observed noticeable bias for the pairs $(\mathcal{ID} = (7, 0),\ \mathcal{OD} = (1, 15))$, $(\mathcal{ID} = (7, 0),\ \mathcal{OD} = (1, 13))$, $(\mathcal{ID} = (7, 13),\ \mathcal{OD} = (1, 26))$ after four forward rounds. We use $(\mathcal{ID} = (7, 0),\ \mathcal{OD} = (1, 13))$ to exploit the second stage of the attack.

For the third stage, we observe that if we choose the $\mathcal{OD}_3$ to be very close to $\mathcal{OD}_2$, there is a huge number of common elements between the sets of two significant bits of the two stages. Thus, among the shortlisted $(\mathcal{ID} - \mathcal{OD})$ pairs, we choose $(\mathcal{ID} = (7, 13),\ \mathcal{OD} = (1, 26))$ as $(\mathcal{ID}_3 - \mathcal{OD}_3)$ whose $\mathcal{OD}$ is far from $\mathcal{OD}_2$. The differentials we have used for our attack along with their bias are therefore as follows Table 3:

We do not claim that the choice of $(\mathcal{ID} - \mathcal{OD})$ is the best possible choice, but it serves our purpose. Now for any key, we observe that out of two randomly chosen IVs, one produces the minimum difference count

Table 3. Distinguishers for the attack against 7-round Salsa.

Stage (i)	Round	$\mathcal{ID}$	$\mathcal{OD}$	Bias	Recovered bits (S_i)
1	5	(7,31)	(4,7)	2^{-39}	23
2	4	(7,0)	(1,13)	0.23	84
3	4	(7,13)	(1, 26)	0.25	90
4	–	–	–	–	59

after the first. According to the terminology introduced in Beierle, Leander, and Todo (2020), such key-IV pairs are called right pairs. Following the idea of Beierle, Leander, and Todo (2020), if p is the probability that a randomly chosen IV would form a right pair, the attack will be repeated p^{-1} times on average. Since $p = \frac{1}{2}$ here, we must multiply the complexity by 2.

3.2. Preprocessing: key-bit partitioning into three subsets

In the preprocessing stage, we split the set of key bits into three subsets: S_1, S_2, and S_3.

Stage 1: By setting the input differences to $\mathcal{ID}_1 = (7, 31)$ of X, we obtain a new state X'. We then execute the 4 Salsa round functions on both the states and observe the value at $\mathcal{OD}_1 = (4, 7)$ of the difference matrix $X^5 \oplus X'^5$. Let us call this difference $\Delta_{\mathcal{OD}_1}$. After that, we generate Z, Z' by running 2 more forward Salsa rounds on both states. Now we produce $\bar{X}$ and $\overline{X'}$ by altering a single key bit in X and X'. We calculate $Z - \bar{X}$ and $Z' - \overline{X'}$, run Rev_S^2, and observe the difference at $\mathcal{OD}_1$. Let us call the difference $\bar{\Gamma}_{\mathcal{OD}_1}$. We then check whether $\Delta_{\mathcal{OD}_1} = \bar{\Gamma}_{\mathcal{OD}_1}$. We consider the key bit to be in S_1, that is, the significant bit corresponding to $(\mathcal{ID}_1, \mathcal{OD}_1)$ if the probability of the event of the equality of the two differences is less than the predetermined threshold γ_1. To construct S_1, this procedure is repeated for each of the key bits. For γ_1= 0.55, we get the following S_1 with cardinality 23:

41, 42, 43, 44, 77, 78, 79, 80, 101, 102, 113, 114, 154, 155, 156, 157, 158, 172, 173, 233, 234, 235, 236

Backward bias: For each pair of states X, X', we put random values at all the key bit positions other than the bits of S_1 and construct the matrices $\tilde{X}, \tilde{X'}$. Next, we apply Rev_S^2 on both $Z - \tilde{X}$ and $Z' - \tilde{X'}$, and observe the difference at $\mathcal{OD}_1$. Let us denote this by $\tilde{\Gamma}_{\mathcal{OD}_1}$. We experimentally compute the bias of the event $\Delta_{\mathcal{OD}_1} = \tilde{\Gamma}_{\mathcal{OD}_1}$.

Stage 2: We repeat the procedure from above by setting the input difference to $\mathcal{ID}_2$ =(7,0), running X and X'' for four forward iterations, and recording the difference at $\mathcal{OD}_2 = (1, 13)$, which we label $\Delta_{\mathcal{OD}_2}$. Following that, we produce the keystreams Z, Z''. First we get a set of 100 non-significant bits. Among them 16 bits that are already present in S_1 are disregarded. Since these bits are not present in S_1, we must take the following action for each of them: We obtain $\bar{X}$ and $\overline{X''}$ by complementing the key bit of X, X', then we compute $Z - \bar{X}$ and $Z'' - \overline{X''}$ and run Rev_S^3 on the states. Then we observe the difference at $\mathcal{OD}_2$ ($\Gamma_{\mathcal{OD}_2}$). S_2 includes bits for which the

chances of $(\Delta_{\mathcal{OD}_2} = \Gamma_{\mathcal{OD}_2})$ are less than a threshold value of 0.25. We get a set of 84 elements in S_2 which are as follows:

3, 4, 5, 6, 7, 8, 9, 12, 21, 22, 23, 24, 40, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 58, 59, 60, 61, 66, 67, 68, 69, 81, 82, 83, 92, 93, 94, 95, 103, 104, 105, 130, 131, 132, 136, 137, 138, 139, 140, 144, 145, 146, 151, 152, 153, 177, 178, 179, 180, 181, 182, 183, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 226, 227, 228, 231, 232, 237, 238, 239, 240

Backward bias: For each pair of states X, X', we put random values at all the key bit positions other than the bits of S_1 and S_2, and construct the matrices $\tilde{X}$ and $\tilde{X''}$. In a similar manner as in Stage 1, we apply Rev_S^3 on both of $Z - \tilde{X}, Z'' - \tilde{X''}$, observe the difference at $\mathcal{OD}_2$ and compute backward bias ϵ_{a_2} =0.0011.[1]

Stage 3: In the third stage, we use the input difference $\mathcal{ID}_3$ =(7,13), run X and X'' for 4 forward rounds, and note the difference at $\mathcal{OD}_3$ =(1,26), which we label $\Delta_{\mathcal{OD}_3}$. Applying the same techniques as in the two earlier stages with threshold 0.80, we produce S_3, which has 90 elements. The set S_3 is as follows:

0, 1, 2, 13, 14, 15, 16, 17, 18, 19, 20, 25, 32, 33, 34, 35, 36, 37, 38, 39, 57, 62, 63, 64, 70, 71, 72, 73, 74, 75, 76, 88, 89, 90, 91, 111, 112, 115, 116, 117, 118, 128, 129, 133, 134, 135, 141, 142, 143, 147, 148, 149, 150, 159, 160, 161, 162, 163, 164, 187, 188, 189, 190, 191, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253

Backward bias: For the 59 elements that are not in $S_1 \cup S_2 \cup S_3$, $\epsilon_{a_3} = 0.63$. Once S_1, S_2, and S_3 have been built, the remaining 59 key bits that are not present in either of those sets are compiled into the fourth set, S_4, and they will be searched online.

3.3. Online phase

From the perspective of the attacker, the first step in launching an attack is to gather the keystreams that will lead to the key.

3.3.1. Data collection

In the first stage, the attacker chooses N_1 different IVs and forms N_1 states. Then, by putting the desired $\mathcal{ID}_1$ into each state X, a corresponding X' is created. After the IVs are chosen, the attacker applies the Salsa round function 7 times on both X and X' and collects the corresponding Z and Z' from the original machinery. The attacker then collects N_1 pairs of keystreams and their corresponding IVs. For the second stage of key recovery, for N_2 different IVs the attacker puts the input difference at $\mathcal{ID}_2$ and creates X'', then collects N_2 pairs of keystreams Z, Z''.

[1]The C++ source code of the backward bias determination is uploaded in GitHub (Garai 2023).

3.3.2. Key recovery

This phase comprises three stages. Let us denote the subkey corresponding to the bits of S_i as K_i $(i = 1, 2, 3)$. At first, the attacker fixes an IV at X_7, X_7'.

Stage 1: The attacker begins by attempting to guess K_1 while assigning random values to K_2 and K_3. For each of the assigned IV values at X_6, X_8, X_9 in the original cipher during data collection, the attacker does as follows: they construct a state matrix G using those mentioned above K_1, K_2, K_3 and X_6, X_8, X_9. By putting the difference at $\mathcal{ID}_1$, the attacker constructs G'. Now they run 3 reverse Salsa rounds on states $Z - G$ and $Z' - G'$, where Z, Z' correspond to the output keystream achieved by the same values of (X_6, X_8, X_9). The attacker then observes the difference between the two states at the $\mathcal{OD}_1$ position. This process is done for all N_1 states by changing the IVs at the sixth, eighth, and ninth words. If the difference at the $\mathcal{OD}_1$ position occurs for more than the predetermined threshold T_1, the attacker considers this guess to be a potential candidate of K_1 and proceeds to stage 2. We call this an "alarm." Otherwise, the same process is carried out for another guess for K_1. If we exhaust all possible guesses of K_1 without recovering the key, we return to the initial stage, change the value of X_7, and start the process again.

Stage 2: In this stage, the attacker aims to recover K_2. Here the attacker has a K_1 subkey for which they have already received an alarm. Next, they guess K_2, keep K_1 the same as before, and put random values in K_3. The attacker chooses an IV from the N_2 IVs and produces states G and G'', similarly to the previous stage, where the difference is at $\mathcal{ID}_2$. Then they run $Z - G$ and $Z'' - G''$ by 3 reverse Salsa rounds and check the difference at the $\mathcal{OD}_2$ position. If the count of the difference is more than a predetermined threshold (T_2), they gets an "alarm" and move forward to the next stage. If the number of occurrences of difference at the $\mathcal{OD}_2$ position does not cross T_2, the attacker chooses another guess for K_2 and repeats the same process. If all guesses are exhausted but the correct key is not found, the algorithm goes back to stage 1.

Stage 3: This stage is a repetition of the previous stage. Here the attacker recovers K_3 with the same technique used in the previous two steps. To guess K_3 they check whether the number of occurrences crosses the threshold T_3. If all the guesses are exhausted, then the attacker moves to the previous step.

Stage 4: This stage exhaustively searches K_4. For each guess of K_4 and the already assigned guesses of K_1, K_2, and K_3, Salsa20/7 is run on G, and the output keystream is matched with the original keystream. If a match is found for some guess, that guessed value of K_4 along with already-assigned values of K_1, K_2, and K_3 are concluded to be correct. Otherwise, the algorithm goes back to stage 3.

Table 4. Complexity details for the attack against Salsa20/7.

Stage (i)	S_i	p_i^{-1}	α_i	N_i	Time
1	23	2^3	30	$2^{90.72}$	$2^{113.72}$
2	84	2	100	$2^{31.61}$	$2^{116.61}$
3	90	2	100	$2^{13.14}$	$2^{103.14}$
4	59	–	–	–	2^{59}

4. Complexity formula

If we dig into the literature, we see that Aumasson et al. (2008) used hypothesis testing to compute the complexity of the attack. They considered two possible errors, which are as follows:

1. Non-detection error, where the algorithm can not detect it instead of the correct guess. The probability of this error is denoted by $\Pr_{nd}$.
2. False-alarm error, where a wrong guess gives a high bias, leading the algorithm to accept this guess. The probability of this error is denoted by $\Pr_{fa}$.

$\Pr_{nd}$ is taken to be at most 1.3×10^{-3}, and $\Pr_{fa}$ is chosen suitably to minimize attack complexity. All cryptanalysis on this cipher follows this approach. Dey, Garai, and Maitra (2023) modified the complexity formula for the case of multiple $(\mathcal{ID} - \mathcal{OD})$ pairs. They considered the probability of a false-alarm error in each stage separately. The probability at the i-th stage is $\Pr_{fa_i} = 2^{-\alpha_i}$. If we consider the data complexity for the first and second stages to be N_1 and N_2, respectively, then N_i can be given by:

$$N_i \approx \left(\frac{\sqrt{(\alpha_i) \ln 4} - \Phi^{-1}\left[\frac{1}{q}\Pr_{nd}\right] \sqrt{1 - (\epsilon_{d_i}\epsilon_{a_i})^2}}{\epsilon_{d_i}\epsilon_{a_i}} \right)^2 \tag{3}$$

Here, Φ is the cumulative distribution function of the standard normal distribution. q is the number of $\mathcal{ID} - \mathcal{OD}$ pairs used. The remaining notations have been introduced already.

4.1. Complexity formula for our new attack approach

In our attack approach, we first search for the K_1 subkey, using N_1 pairs of states. Then we move forward to recover K_2, using N_2 pairs of states. K_3 bits are searched exhaustively. The corresponding complexities of these tasks are as follows:

- Complexity to recover K_i : $2^{|S_i|} \cdot N_i$ for $i \in \{1, 2, 3\}$
- Complexity to recover K_4 : $2^{|S_4|}$

In each of the first three stages, a false-alarm error is possible. We express these probabilities of false-alarm error as $2^{-\alpha_i}$. Let Comp_2 denote the total complexity to recover S_2 and S_3 key bits. Since the probability of false-alarm error in the first stage is $2^{-\alpha_1}$, out of $2^{|S_1|}$ guesses, $2^{|S_1|-\alpha_1}$ will give a false alarm and the algorithm will proceed to the next stage. Extra computation of complexity Comp_2 is to be performed for each of these.

Also, for the correct guess of S_1 key bits, the algorithm proceeds to the second stage and Comp_2 would be added further to the complexity. So the total complexity of this multi-step approach is as follows:

$$p_1^{-1}(2^{|S_1|} \cdot N_1 + 2^{|S_1|-\alpha_1} \cdot \text{Comp}_2) + \text{Comp}_2. \tag{4}$$

Here p_1^{-1} is the number of times we have to repeat the attack in order to achieve a right pair.

Therefore:

$$\text{Comp}_2 = p_2^{-1}(2^{|S_2|} \cdot N_2 + 2^{|S_2|-\alpha_2} \cdot \text{Comp}_3) + \text{Comp}_3 \tag{5}$$

Let us compute the complexity Comp_3. Here, the first job is to recover K_3, which is done with complexity $2^{|S_3|} \cdot N_3$. Since the false-alarm error probability is $2^{-\alpha_3}$, $2^{|S_3|-\alpha_3}$ wrong guesses produce false alarms. The algorithm is carried on to the third stage by each of them, adding an extra $2^{|S_3|}$ in the complexity. Also note that the algorithm advances to stage 3 when the correct guess is made.

$$\text{Comp}_3 = p_3^{-1}(2^{|S_3|} \cdot N_3 + 2^{|S_3|-\alpha_3} \cdot 2^{|S_4|} + 2^{|S_4|}) \tag{6}$$

The overall data complexity is $p_1^{-1}N_1 + p_2^{-1}N_2 + p_3^{-1}N_3$.

4.2. Complexity of our attack on 256-bit Salsa20/7

The values of N_1, N_2, N_3 are calculated by formula 3. In our case, $q = 3$. So $\Phi^{-1}[\frac{1}{3}\Pr_{nd}] = -3.4$. For $\alpha_1 = 30, \alpha_2 = 100, \alpha_3 = 30$, we have $N_1 = 2^{90.72}, N_2 = 2^{31.61}, N_3 = 2^{13.14}$, respectively. Therefore, $\text{Comp}_3 = 2^{103.14}$ from equation 6. Using this value, from equation 5 we get $\text{Comp}_2 = 2(2^{84} \cdot 2^{31.61} + 2^{84-100} \cdot 2^{103.14}) + 2^{103.14} = 2^{116.61}$. Finally, putting this value into equation 4, the final time complexity is as follows:

$$2(2^{23} \cdot 2^{95.74} + 2^{23-30} \cdot 2^{116.61}) + 2^{116.61} = 2^{119.74}.$$

The total data needed to carry out this attack are approximately $2^{93.72}$.

Table 5. Distinguishers for the attack against Salsa20/7.5.

Stage	Round	$\mathcal{ID}$	$\mathcal{OD}$	Bias	Recovered Bits
1	5	(7,31)	(4,7)	2^{-39}	47
2	4	(7,0)	(1,15)	0.39	149
3	–	–	–	–	60

Table 6. Distinguishers for the attack against ChaCha20/6.5.

Stage	Round	$\mathcal{ID}$	$\mathcal{OD}$	Bias	Recovered bits
1	5	$(15,29) \oplus (15,9)$	$(2,0) \oplus (6,7) \oplus (6,19) \oplus (10,12) \oplus (14,0)$	$2^{-34.15}$	61
2	4	(12,6)	$(1,0) \oplus (6,7) \oplus (11,0)$	0.003	109
3	–	–	–	–	86

5. Attack on 256-bit Salsa20/7.5

We give the first-ever attack on 256-bit Salsa20/7.5 with the help of a multi-step approach. We break down the key-bit sets into three subsets—S_1, S_2, and S_3—in the preprocessing stage.

For the first stage, we have used the 5-round differential distinguisher $\mathcal{ID}_1 - \mathcal{OD}_1 = ((7,31)\text{-}(4,7))$ as given by Coutinho et al. (2023). To attack 7.5 rounds, we have to go back 2.5 rounds to reach the desired differential position. Using the threshold $\gamma = 0.55$, we obtain 47 elements for S_1, which follow:

0, 1, 2, 3, 40, 41, 42, 43, 44, 77, 78, 79, 80, 101, 102, 111, 112, 113, 114, 129, 130, 131, 132, 134, 135, 136, 137, 153, 154, 155, 156, 157, 158, 170, 171, 172, 173, 227, 228, 229, 230, 233, 234, 235, 236, 247, 248

The other 209 key bits give us a backward bias (ϵ_a) of 0.07764 experimentally. In the second stage, we have used the 4-round distinguisher $\mathcal{ID}_2 - \mathcal{OD}_2 = ((7,0)\text{-}(1,15))$. In this stage, we obtained a set of 149 elements that are different from the set S_1. S_2 consists of all those 149 elements.

4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 29, 30, 31, 38, 39, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 81, 82, 83, 84, 85, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 103, 104, 105, 106, 107, 108, 109, 110, 115, 116, 117, 118, 119, 120, 128, 133, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 159, 160, 161, 162, 163, 164, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 225, 226, 231, 232, 237, 238, 239, 240, 241, 242

Since there is no common elements between S_1 and S_2, we have a total of 196 elements in $S_1 \cup S_2$. The other 60 elements are labeled as S_3, and they are searched exhaustively. Figure 1 gives an overview of the attack.

5.1. Complexity

The key-bit set is broken into two—hence $q = 2$—so in formula 3, $\Phi^{-1}[\frac{1}{2}\Pr_{nd}] = -3.2$. With the value of $\alpha_1 = 60$, we get $N_1 = 2^{98.64}$. For $\alpha_2 = 160$, the data complexity for the second stage becomes $N_2 = 2^{14.54}$. Therefore, the total complexity for stage 2 and stage 3 becomes $2^{163.54}$, which in turn makes the total time complexity of the attack approximately $2^{163.54}$.

6. Attack on 256-bit ChaCha20/6.5

Here we present the first-ever attack on ChaCha20/6.5. We applied the attack with two $\mathcal{ID} - \mathcal{OD}$s. The first $\mathcal{ID} - \mathcal{OD}$ is given by Bellini et al. (2023). The $\mathcal{ID} - \mathcal{OD}$s we have considered here are as follows:

1. $\mathcal{ID}_1 - \mathcal{OD}_1 : ((15, 29) \oplus (15, 9), (2, 0) \oplus (6, 7) \oplus (6, 19) \oplus (10, 12) \oplus (14, 0))$ (Bellini et al. 2023)
2. $\mathcal{ID}_2 - \mathcal{OD}_2 : ((12, 6), (1, 0) \oplus (6, 7) \oplus (11, 0))$

In the first stage of preprocessing, using $\gamma = 0.6$ we get 61 significant bits, which are directly included in S_1; the other 195 key bits give a backward bias (ϵ_a) of 0.0845. The set $S_1 =$

16, 17, 18, 19, 20, 21, 22, 23, 67, 68, 69, 73, 74, 75, 79, 80, 81, 105, 106, 107, 112, 113, 114, 120, 121, 122, 123, 124, 125, 126, 132, 133, 134, 135, 136, 137, 138, 139, 195, 196, 197, 201, 202, 203, 215, 216, 217, 227, 228, 229, 230, 236, 237, 238, 239, 240, 241, 242, 252, 253, 254

In the second stage, we have used the 4-round distinguisher. We have set $\gamma = 0.7$ and obtained S_2 to be a set of 109 elements after removing the common elements from S_1.

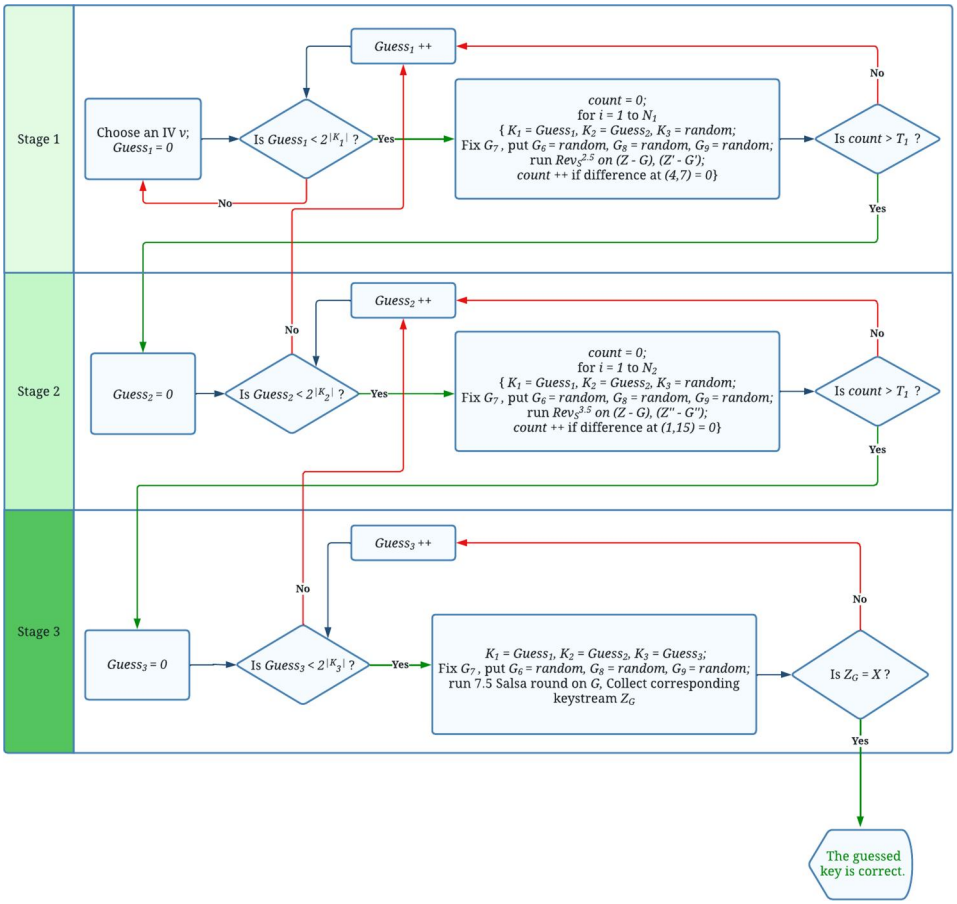

Figure 1. Multi-step approach on 256-bit Salsa20/7.5.

0, 1, 2, 3, 9, 10, 11, 13, 14, 15, 24, 25, 26, 27, 28, 29, 30, 35, 36, 37, 48, 49, 50, 54, 55, 56, 57, 58, 82, 86, 87, 88, 89, 90, 91, 92, 93, 94, 96, 97, 98, 109, 110, 111, 115, 116, 117, 118, 119, 127, 129, 130, 131, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 161, 162, 163, 164, 165, 166, 168, 169, 170, 171, 172, 173, 174, 183, 184, 185, 194, 198, 199, 204, 205, 206, 207, 208, 209, 210, 220, 221, 222, 224, 225, 226, 231, 232, 233, 234, 235, 243, 244, 245, 246, 247, 248, 249

The rest of the 86 elements that are not in S_1 or S_2 are considered to be in S_3. These 86 elements give a backward bias of 0.0873.

6.1. Complexity

Here, as with Salsa20/7.5, $q = 2$, so using formula 3 we get $N_1 = 2^{83.74}$ for $\alpha_1 = 75$. In the next stage, using $\alpha_2 = 120$, $N_2 = 2^{31.66}$. But the whole process has to be repeated 2^5 times to get a suitable IV. We have $\text{Comp}_2 = 2^{109} \cdot 2^{36.66} + 2^{195-120} + 2^{86} \approx 2^{145.66}$. Therefore, the total complexity becomes $2^{151.74}$, while the overall data complexity is $2^{90.74} + 2^{36.66} \approx 2^{90.74}$.

7. Conclusion

This work focused primarily on improving forward bias and modified attack technique. Use of a multi-step approach produces a significant improvement in the attack complexity. This attack opens several directions for further research. The first is whether we can tweak the attack model slightly to achieve improvement in complexity, and the second is analyzing the scenario of using more than two $\mathcal{ID} - \mathcal{OD}$ pairs to improve the attack further. We believe that this work will lead to achieving better cryptanalytic results on Salsa family of ciphers.

About the Authors

Hirendra Kumar Garai earned his Master of Science (M.Sc.) in Mathematics from Visva-Bharati University, India in 2018. Presently, he is in his fourth year as a doctoral student at BITS Pilani, Hyderabad Campus, India. His research primarily centers on symmetric key cryptanalysis.

Sabyasachi Dey got his Ph.D. in mathematics from the Indian Institute of Technology Madras in Chennai, India, in 2018. Presently, he is an Assistant Professor at the Birla Institute of Technology and Science (BITS) in Pilani, India. Symmetric key cryptology is one of his main research interests.

Disclosure statement

No potential conflict of interest was reported by the author(s).

Funding

Human Resource Development Centre, Council of Scientific and Industrial Research.

References

Aumasson, J., S. Fischer, S. Khazaei, W. Meier, and C. Rechberger. 2008. New features of Latin dances: Analysis of Salsa, ChaCha, and Rumba. Fast Software Encryption, 15th International Workshop, Lausanne, Switzerland, Revised Selected Papers, 5086, 470–88. doi: 10.1007/978-3-540-71039-4_30.

Beierle, C., G. Leander, and Y. Todo. 2020. Improved differential-linear attacks with applications to ARX ciphers. Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, Santa Barbara, CA, USA, Proceedings, Part III, 12172, 329–58. doi: 10.1007/978-3-030-56877-1_12.

Bellini, E., D. Gerault, J. Grados, R. H. Makarim, and T. Peyrin. 2023. Boosting differential-linear cryptanalysis of chacha7 with MILP. *IACR Transactions on Symmetric Cryptology* 2023 (2):189–223. doi: 10.46586/tosc.v2023.i2.189-223.

Bernstein, D. J. 2008. ChaCha, a variant of Salsa20. *Workshop Record of SASC* 8:3–5. https://cr.yp.to/chacha/chacha-20080128.pdf.

Bernstein, D. J. 2008. *The Salsa20 family of stream ciphers*, 84–97. Berlin, Heidelberg: Springer Berlin Heidelberg. doi: 10.1007/978-3-540-68351-3_8.

Choudhuri, A. R., and S. Maitra. 2016. Significantly improved multi-bit differentials for reduced round Salsa and ChaCha. *Transactions on Symmetric Cryptology*. 2016 (2):261–87: doi: 10.13154/tosc.v2016.i2.261-287.

Coutinho, M., and T. C. S. Neto. 2021. Improved linear approximations to ARX ciphers and attacks against ChaCha. *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Zagreb, Croatia, Proceedings, Part I, 12696, 711–40. doi: 10.1007/978-3-030-77870-5_25.

Coutinho, M., I. Passos, J. C. G. Vásquez, S. Sarkar, F. L. L. de Mendonça, R. T. de Sousa, Jr., and F. Borges. 2023. Latin dances reloaded: Improved cryptanalysis against salsa and chacha, and the proposal of forró. *Journal of Cryptology* 36 (3):18. doi: 10.1007/s00145-023-09455-5.

Crowley, P. 2005. Cryptology ePrint Archive, Paper 2005/375. Truncated differential cryptanalysis of five rounds of salsa20. https://eprint.iacr.org/2005/375.

Dey, S., and S. Sarkar. 2017. Improved analysis for reduced round Salsa and Chacha. *Discrete Applied Mathematics* 227:58–69. doi: 10.1016/j.dam.2017.04.034.

Dey, S., H. K. Garai, and S. Maitra. 2023. Cryptanalysis of reduced round ChaCha - new attack & deeper analysis. *IACR Transactions on Symmetric Cryptology* 2023 (1):89–110. doi: 10.46586/tosc.v2023.i1.89-110.

Dey, S., H. K. Garai, S. Sarkar, and N. K. Sharma. 2022. Revamped differential-linear cryptanalysis on reduced round ChaCha. *Advances in Cryptology - EUROCRYPT 2022 - 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Trondheim, Norway, Proceedings, Part III, 13277, 86–114. doi: 10.1007/978-3-031-07082-2_4.

Fischer, S., W. Meier, C. Berbain, J.-F. Biasse, and M. J. B. Robshaw. 2006. Non-randomness in estream candidates salsa20 and TSC-4. In R. Barua and T. Lange, editors, *Progress in Cryptology - INDOCRYPT 2006, 7th International Conference on Cryptology in India, Kolkata, India, December 11-13, 2006, Proceedings, volume 4329 of Lecture Notes in Computer Science*, 2–16. Springer. doi: 10.1007/11941378_2.

Garai, H. K. 2023. 256-bit-salsa20-7. GitHub repository. https://github.com/namenotpublished/256-bit-Salsa20-7.git.

Maitra, S. 2016. Chosen IV cryptanalysis on reduced round ChaCha and Salsa. *Discrete Applied Mathematics* 208:88–97. doi: 10.1016/j.dam.2016.02.020.

Tsunoo, Y., T. Saito, H. Kubo, T. Suzaki, and H. Nakashima. 2007. *Differential cryptanalysis of Salsa20/8*. SASC- The State of the Art of Stream Ciphers.