IACR Transactions on Symmetric Cryptology
ISSN 2519-173X, Vol. 2022, No. 3, pp. 152–190. DOI:10.46586/tosc.v2022.i3.152-190

New Cryptanalysis of ZUC-256 Initialization Using Modular Differences

Fukang Liu[1], Willi Meier[4], Santanu Sarkar[5], Gaoli Wang[6,7], Ryoma Ito[2], Takanori Isobe[1,2,3]

[1] University of Hyogo, Hyogo, Japan
liufukangs@gmail.com,takanori.isobe@ai.u-hyogo.ac.jp

[2] National Institute of Information and Communications Technology, Tokyo, Japan
itorym@nict.go.jp

[3] PRESTO, Japan Science and Technology Agency, Tokyo, Japan

[4] University of Applied Sciences and Arts Northwestern Switzerland (FHNW), Windisch, Switzerland
willimeier48@gmail.com

[5] Indian Institute of Technology Madras, Chennai, India
santanu@iitm.ac.in

[6] East China Normal University, Shanghai, China

[7] Key Lab of Cryptologic Technology and Information Security, Ministry of Education, Shandong University, Jinan, China
glwang@sei.ecnu.edu.cn

Abstract. ZUC-256 is a stream cipher designed for 5G applications by the ZUC team. Together with AES-256 and SNOW-V, it is currently being under evaluation for standardized algorithms in 5G mobile telecommunications by Security Algorithms Group of Experts (SAGE). A notable feature of the round update function of ZUC-256 is that many operations are defined over different fields, which significantly increases the difficulty to analyze the algorithm.

As a main contribution, with the tools of the modular difference, signed difference and XOR difference, we develop new techniques to carefully control the interactions between these operations defined over different fields. At first glance, our techniques are somewhat similar to those developed by Wang et al. for the MD-SHA hash family. However, as ZUC-256 is quite different from the MD-SHA hash family and its round function is much more complex, we are indeed dealing with different problems and overcoming new obstacles.

As main results, by utilizing complex input differences, we can present the first distinguishing attacks on 31 out of 33 rounds of ZUC-256 and 30 out of 33 rounds of the new version of ZUC-256 called ZUC-256-v2 with low time and data complexities, respectively. These attacks target the initialization phase and work in the related-key model with weak keys. Moreover, with a novel IV-correcting technique, we show how to efficiently recover at least 16 key bits for 15-round ZUC-256 and 14-round ZUC-256-v2 in the related-key setting, respectively. It is unpredictable whether our attacks can be further extended to more rounds with more advanced techniques. Based on the current attacks, we believe that the full 33 initialization rounds provide marginal security.

Keywords: 5G · stream cipher · ZUC-256 · differential attack · modular difference · signed difference

Received: 2022-03-01 Revised: 2022-06-01 Accepted: 2022-08-01 Published: 2022-09-09

1 Introduction

The modular difference has been a prominent tool in the cryptanalysis of the MD-SHA hash family due to a series of work [WLF+05, WY05, WYY05]. Since such a major breakthrough in 2005, similar techniques have been applied to many MD4-like hash functions and there is a large number of related publications [CR06, SSA+09, MNS11, SBK+17, LP19, LP20]. The effectiveness of this technique contributes to the dedicated control of the sign of the difference. That is, while the standard XOR difference [BS90] captures the fact that a bit has been changed, the signed difference [WLF+05] will capture how the bit value is changed, i.e., from 1 to 0 or from 0 to 1. This feature of the signed difference makes it interact well with the modular difference. As the addition modulo 2^n ($n \in \{32, 64\}$) and some simple boolean functions are used in the round update functions of these MD4-like hash functions in a hybrid way, the attackers can view the modular difference from the perspective of the signed difference when processing the difference transitions in the boolean functions. In addition, they can cancel the difference from the perspective of the modular difference when processing the modular addition. With these strategies, it is possible to carefully deduce a collision-generating differential characteristic.

Despite the fact that it is a famous and powerful technique, there seem to be few successful applications of this technique to cryptographic primitives following a quite different design strategy from that of the MD-SHA hash family. A notable application is to construct collision-generating differential characteristics for ARX constructions like the hash function Skein [Leu13]. It should be mentioned that ARX constructions are still similar to the MD-SHA hash family, which use modular **A**ddition, bit **R**otation and **X**OR operation. For many other works on ARX constructions like [AFK+08, BVC16], the used techniques are then quite different.

In this work, we demonstrate the usage of the modular difference in the cryptanalysis of the stream cipher ZUC-256 [The18], which obviously follows a different design strategy from that of ARX constructions and the MD-SHA hash family. In a nutshell, the round update function of ZUC-256 involves such operations as addition modulo $2^{31} - 1$, addition modulo 2^{32}, the XOR operation, the S-box transformation over $GF(2^8)$ and the linear transformation over $GF(2^{32})$. At the first glance, as many operations are defined in different fields, developing non-trivial cryptanalytic techniques for ZUC-256 seems rather challenging, especially when devising an attack by taking the interactions between all these operations into account. Moreover, the prime field $GF(2^{31} - 1)$ seems to be only used in the ZUC family, i.e., ZUC-128 and ZUC-256.

Although the modular additions, i.e., modulo $2^{31} - 1$ and 2^{32}, are used in ZUC-256, it does not necessarily imply that the cryptanalysis with the tool of the modular difference will be effective for it. The main obstacle is that there are 8-bit S-box transformations and 32-bit linear transformations in the round function of ZUC-256. How to control the difference transitions between the modular additions, the 8-bit S-boxes and the 32-bit linear transformations is thus becoming a critical problem to solve in order to devise advanced differential attacks. As far as we know, there is no corresponding technique developed for this problem and it is in general difficult.

Backgrounds for the ZUC family. ZUC-128 is a stream cipher with 128-bit security and has been adopted as the third suite of the 3GPP confidentiality and integrity algorithms called 128-EEA3 and 128-EIA3.

As the successor of ZUC-128, ZUC-256 [The18] is designed for 5G applications with 256-bit security, which differs from ZUC-128 only in the initialization phase and message authentication codes generation phase. Since its proposal in 2018, a distinguishing attack [YJM20] on the keystream generation phase and a practical 28-round distinguishing attack [BM20] on the initialization phase in the related-key setting have been published. Regarding the distinguishing attack on the keystream phase [YJM20], the used techniques are somewhat complicated and it requires 2^{236} data and time complexity. Although this yields an academic attack on full ZUC-256, finding attacks on other aspects of ZUC-256 (e.g., the initialization phase) is still of great interest and importance to understand the security of ZUC-256.

Very recently, a new version of ZUC-256 was published by the ZUC team [ETS21, Tea21], where only the loading scheme at the initialization phase is changed. For convenience, we call it ZUC-256-v2. Moreover, the designers describe a 27-round distinguishing attack in [Tea21] in the related-key setting and expect that each state bit of ZUC-256-v2 will have sufficient randomness after 32 rounds and finally conclude that the full 33 initialization rounds are secure. It is not difficult to find that the 27-round attack is indeed implied in the 28-round attack [BM20] because the differences in the 256-bit key are chosen in the same way. More details will be discussed later.

The attack scenario. The described attack scenario of the distinguishing attacks in [BM20, Tea21] is not commonly used in the cryptanalysis of stream ciphers because it requires the attackers to know some information (e.g., some internal state bits) which cannot be derived from the allowed observable outputs (the keystream). Such an attack scenario in the related-key/single-key settings can date back to 2011, which is used to evaluate the security of ZUC-128 by SAGE under 3GPP's request before standardizing ZUC-128. This in a way explains why the ZUC team took this attack vector into account again. Similar to [BM20, Tea21], our distinguishing attacks also work under this attack scenario in the related-key setting. In addition, by restricting that the attackers can only derive information from the keystream, which is much more realistic and commonly used, we also demonstrate the key-recovery attacks on a smaller number of rounds in the related-key setting. We have to emphasize that it is unclear whether our distinguishing attacks in this unusual attack scenario will affect the standardization process. However, as will be seen, our developed techniques do advance the understanding of the security of ZUC-256 initialization phase and we view this as a more important contribution.

Our contributions. Our contributions are summarized below:

1. By using the modular/signed/XOR differences, we develop novel techniques to carefully control the interactions between the difference transitions through the modular addition, the 8-bit S-box transformations and the 32-bit linear transformations in ZUC-256. Due to the usage of signed differences, all the attacks in this paper naturally work in the *weak-key setting*, i.e. it is required to add conditions on some key bits.

2. To extend the 28-round attack [BM20] to more rounds, we develop new techniques to utilize complex input differences to attack 31-round ZUC-256 and 30-round ZUC-256-v2, respectively. These attacks are of *low time and data complexity*, as shown in Table 1. Moreover, the attacks have been experimentally verified. Our results suggest that the full 33 rounds of initialization provide marginal security regarding this type of distinguishing attack.

3. Based on the discovered input difference, we propose a novel *IV-correcting technique* to achieve partial key-recovery attacks in the related-key setting. By observing the first 32-bit keystream word, we can recover at least 16 key bits for both 15-round ZUC-256 and 14-round ZUC-256-v2 with low time complexity.

We emphasize that finding the complex input differences in our attacks is very technical and requires some deep understanding of the relations between the signed difference, modular difference and XOR difference. Especially, contradictions easily occur at the phase to search for such complex input differences. To search for valid input differences, we generally utilize a guess-and-determine technique by constantly checking contradictions. It is found that our algorithm can produce a solution of the input difference in seconds.

Table 1: Summary of the attacks on ZUC-256 and ZUC-256-v2, where at least 16 key bits are recovered in the key-recovery attacks. All the attacks are in the related-key setting. In addition, when the target is the initialization phase, attackers can access some internal state bits. When the target is the actual cipher, attackers can only access the keystream words.

Target	Attack Type	Rounds	Time	Data	Ref.
ZUC-256 initialization	distinguisher	28 (out of 33)	2^{23}	2^{23}	[BM20]
ZUC-256 initialization	distinguisher	31 (out of 33)	2^{29}	2^{29}	Section 7
ZUC-256-v2 initialization	distinguisher	30 (out of 33)	$2^{39.8}$	$2^{39.8}$	Section 7
ZUC-256 cipher	key recovery	15 (out of 33)	2^{47}	2^{47}	Section 7
ZUC-256-v2 cipher	key recovery	14 (out of 33)	2^{58}	2^{58}	Section 7

Organization of this paper. First, we introduce the notation used in this paper and the specification of ZUC-256 and ZUC-256-v2 in Section 2. Then, the basic ideas of our attacks are explained in Section 3. The relations between the XOR difference, modular difference and signed difference will be studied in Section 4. Our critical observations and how to identify advanced strategies to choose input differences will be detailed in Section 5. The search for the input difference is then described in Section 6. The discovered biased linear relations are demonstrated in Section 7. Finally, the paper is concluded in Section 8.

2 Preliminaries

2.1 Notation

Throughout this paper, $p = 2^{31} - 1$, $[a, b]$ represents the set $\{i | a \leq i \leq b\}$ and $Pr[\zeta]$ represents the probability that the event ζ occurs.

1. $\oplus$, $\vee$, $\wedge$, $\gg$ and $\ll$ represent the bitwise exclusive OR, OR, AND, right shift and left shift, respectively.

2. $\boxplus_{32}$ and $\boxminus_{32}$ represent addition and subtraction modulo 2^{32}, respectively. $\boxplus$ and $\boxminus$ represent addition and subtraction modulo p, respectively.

3. $a||b$ represents the concatenation of strings a and b. $a \cdot b$ represents $a \times b \bmod p$. a^{-1} $(a \neq 0 \mod p)$ represents the inverse of a in $GF(p)$, i.e., $a \cdot a^{-1} = 1$.

4. a_L and a_H represent the rightmost 16 bits and the leftmost 16 bits of integer a, respectively. Note that a can be a 31-bit or 32-bit value and this should be clear from the contexts. We emphasize that a_L and a_H are always 16-bit values.

5. $a[i]$ and $a[j : i]$ represent $(a \gg i) \wedge \texttt{0x1}$ and $(a \gg i) \wedge (2^{j-i+1} - 1)$, respectively.

6. Δa represents the XOR difference $a' \oplus a$.

7. δa represents the modular difference $a' \boxminus a$.

8. ∇a represents the signed difference of $a \in GF(p)$. The i-th element of ∇a denoted by $\nabla a[i]$ is written as follows:

$$\nabla a[i] = \begin{cases} \texttt{n} & (a[i] = 0, a'[i] = 1) \\ \texttt{u} & (a[i] = 1, a'[i] = 0) \\ \texttt{=} & (a[i] = a'[i]) \\ \texttt{0} & (a[i] = a'[i] = 0) \\ \texttt{1} & (a[i] = a'[i] = 1) \end{cases} \tag{1}$$

For example, if $a =$ `0x10080001` and $a' =$ `0x01808001`, ∇a can be written as

$$\begin{array}{rcl} a & = & \texttt{00\underline{1}\ 000\underline{0}\ \underline{0}000\ \underline{1}000\ \underline{0}000\ 0000\ 0000\ 0001,} \\ a' & = & \texttt{00\underline{0}\ 000\underline{1}\ \underline{1}000\ \underline{0}000\ \underline{1}000\ 0000\ 0000\ 0001,} \\ \nabla a & = & \texttt{==u\ ===n\ n===\ u===\ n===\ ====\ ====\ ====.} \end{array}$$

9. ∇a_H and ∇a_L represent the signed difference of the leftmost and rightmost 16 bits of a, respectively.

We notice that in the ZUC-256 specification, each element in $GF(p)$ belongs to the set $\{i|1 \leq i \leq p\}$ rather than $\{i|0 \leq i < p\}$, though the two sets are identical in $GF(p)$. Therefore, for $z = x \boxplus y$, we will have $x, y, z \in \{i|1 \leq i \leq p\}$. However, in the sections of cryptanalysis, when $\delta z = \delta x \boxplus \delta y = p$, we will simply write $\delta z = 0$ for readability as $0 = p \bmod p$.

2.2 Description of ZUC-256

The ZUC-256 stream cipher [The18] is a successor of the ZUC-128 stream cipher [ETS11] with only minor modifications, regarding the initialization phase and the message authentication codes generation phase. As we target the security of the initialization phase, in the following, we will describe the specification of the ZUC-256 initialization. More details of ZUC-256 can be referred to [The18].

The ZUC-256 initialization is depicted in Figure 1. It can be observed that the state update of ZUC-256 involves three parts. The first part is a 496-bit linear feedback shift register (LFSR) defined over $GF(p)$, which is composed of sixteen 31-bit words $(S_{15}, S_{14}, \ldots, S_0)$ with $1 \leq S_i \leq p$ $(0 \leq i \leq 15)$. The second part is called bit reorganization (BR), where four 32-bit words (X_0, X_1, X_2, X_3) will be computed according to some words in the LFSR. The last part is called finite state machine (FSM), where there are two 32-bit registers (R_1, R_2) used as the memory of FSM.

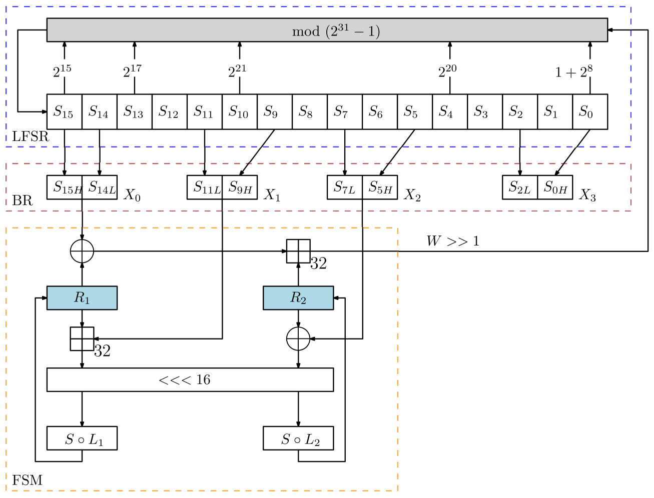

Figure 1: The initialization phase of ZUC-256.

There are in total $32 + 1 = 33$ initialization rounds. For the first 32 clocks, the state is updated in the following way, where $t \in [0, 31]$.

$$X_0^t = S_{15H}^t || S_{14L}^t, \tag{2}$$
$$X_1^t = S_{11L}^t || S_{9H}^t, \tag{3}$$

$$\begin{aligned}
X_2^t &= S_{7L}^t || S_{5H}^t, && (4)\\
W^t &= (R_1^t \oplus X_0^t) \boxplus_{32} R_2^t, && (5)\\
S_i^{t+1} &= S_{i+1}^t \ (0 \leq i \leq 14), && (6)\\
S_{15}^{t+1} &= (W^t \gg 1) \boxplus 257 \cdot S_0^t \boxplus 2^{20} \cdot S_4^t \boxplus 2^{21} \cdot S_{10}^t \boxplus 2^{17} \cdot S_{13}^t \boxplus 2^{15} \cdot S_{15}^t, && (7)\\
R_1^{t+1} &= S \circ L_1\big((R_1^t \boxplus_{32} X_1^t)_L || (R_2^t \oplus X_2^t)_H\big), && (8)\\
R_2^{t+1} &= S \circ L_2\big((R_2^t \oplus X_2^t)_L || (R_1^t \boxplus_{32} X_1^t)_H\big). && (9)
\end{aligned}$$

In the above, operations (L_1, L_2, S) are used. For the operation S, four 8-bit S-boxes will be applied to a 32-bit value in parallel, while the L_1 and L_2 are linear transformations over $GF(2^{32})$. Their details can be referred to [ETS11].

At the 33rd clock, i.e., the last round, we only need to modify Eq. 6 as follows, while keeping the remaining unchanged.

$$S_{15}^{t+1} \quad = \quad 257 \cdot S_0^t \boxplus 2^{20} \cdot S_4^t \boxplus 2^{21} \cdot S_{10}^t \boxplus 2^{17} \cdot S_{13}^t \boxplus 2^{15} \cdot S_{15}^t.$$

Specifically, the only difference is that W^t is no longer used to update S_{15}^{t+1}.

The first 32-bit keystream word. After 33 initialization rounds, the first 32-bit keystream word Z will be computed in the following way, where $t = 33$.

$$X_0^t = S_{15H}^t || S_{14L}^t, \ X_3^t = S_{2L}^t || S_{0H}^t, \ Z = \big((R_1^t \oplus X_0^t) \boxplus_{32} R_2^t\big) \oplus X_3^t.$$

Loading the key and IV. We now describe how the initial values of $(S_{15}^0, \ldots, S_0^t)$ and (R_1^0, R_2^0) are defined, i.e., how to load the key and IV. For ZUC-256, the 256-bit key K can be written as $(K_{31}, K_{30}, \ldots, K_0)$ with $K_i \in \mathbb{F}_2^8$ $(0 \leq i \leq 31)$ and IV can be written as $(IV_{24}, IV_{23}, \ldots, IV_0)$ with $IV_i \in \mathbb{F}_2^8$ $(0 \leq i \leq 16)$ and $IV_j \in \mathbb{F}_2^6$ $(17 \leq j \leq 24)$. There are also some specified constants in ZUC-256, which can be written as $(d_{15}, d_{14}, \ldots, d_0)$ with $d_i \in \mathbb{F}_2^7$ $(0 \leq i \leq 15)$ and are defined as follows:

$$\begin{aligned}
&d_0 = \texttt{0x22},\ d_1 = \texttt{0x2f},\ d_2 = \texttt{0x24},\ d_3 = \texttt{0x2a}, d_4 = \texttt{0x6d},\ d_5 = \texttt{0x40},\\
&d_6 = \texttt{0x40},\ d_7 = \texttt{0x40},\ d_8 = \texttt{0x40},\ d_9 = \texttt{0x40},\ d_{10} = \texttt{0x40},\ d_{11} = \texttt{0x40},\\
&d_{12} = \texttt{0x40},\ d_{13} = \texttt{0x52},\ d_{14} = \texttt{0x10},\ d_{15} = \texttt{0x30}.
\end{aligned}$$

The loading scheme is specified as follows:

$$\begin{aligned}
&R_1^0 = 0, \quad R_2^0 = 0,\\
&S_0^0 = K_0 || d_0 || K_{21} || K_{16}, \quad S_1^0 = K_1 || d_1 || K_{22} || K_{17},\\
&S_2^0 = K_2 || d_2 || K_{23} || K_{18}, \quad S_3^0 = K_3 || d_3 || K_{24} || K_{19},\\
&S_4^0 = K_4 || d_4 || K_{25} || K_{20}, \quad S_5^0 = IV_0 || (d_5 \vee IV_{17}) || K_5 || K_{26},\\
&S_6^0 = IV_1 || (d_6 \vee IV_{18}) || K_6 || K_{27}, \quad S_7^0 = IV_{10} || (d_7 \vee IV_{19}) || K_7 || IV_2,\\
&S_8^0 = K_8 || (d_8 \vee IV_{20}) || IV_3 || IV_{11}, \quad S_9^0 = K_9 || (d_9 \vee IV_{21}) || IV_{12} || IV_4,\\
&S_{10}^0 = IV_5 || (d_{10} \vee IV_{22}) || K_{10} || K_{28}, \quad S_{11}^0 = K_{11} || (d_{11} \vee IV_{23}) || IV_6 || IV_{13},\\
&S_{12}^0 = K_{12} || (d_{12} \vee IV_{24}) || IV_7 || IV_{14}, \quad S_{13}^0 = K_{13} || d_{13} || IV_5 || IV_8,\\
&S_{14}^0 = K_{14} || (d_{14} \vee K_{31}[7:4]) || IV_{16} || IV_9, \quad S_{15}^0 = K_{15} || (d_{15} \vee K_{31}[3:0]) || K_{30} || K_{29}.
\end{aligned}$$

The new loading scheme. Recently, the ZUC team published a new loading scheme [Tea21], where the length of IV is reduced to 128 bits. To distinguish it from the above version, we call ZUC-256 with the new loading scheme as **ZUC-256-v2**. In the new loading scheme, IV is written as $(IV_{15}, IV_{14}, \ldots, IV_0)$ with $IV_i \in \mathbb{F}_2^8$ $(0 \leq i \leq 15)$. The constants are also changed and we write them as $(D_{15}, D_{14}, \ldots, D_0)$. $D_i \in \mathbb{F}_2^7$ $(0 \le i \le 15)$ are specified as follows:

$$D_0 = \texttt{0x64},\ D_1 = \texttt{0x43},\ D_2 = \texttt{0x7b},\ D_3 = \texttt{0x2a},\ D_4 = \texttt{0x11},\ D_5 = \texttt{0x05},$$
$$D_6 = \texttt{0x51},\ D_7 = \texttt{0x42},\ D_8 = \texttt{0x1a},\ D_9 = \texttt{0x31},\ D_{10} = \texttt{0x18},\ D_{11} = \texttt{0x66},$$
$$D_{12} = \texttt{0x14},\ D_{13} = \texttt{0x2e},\ D_{14} = \texttt{0x01},\ D_{15} = \texttt{0x5c}.$$

For ZUC-256-v2, the initial state is defined as below:

$$R_1^0 = 0,\quad R_2^0 = 0,\quad S_i^0 = K_i || D_i || K_{16+i} || K_{24+i}\ (0 \le i \le 6),$$
$$S_i^0 = K_i || D_i || IV_{i-7} || IV_{i+1}\ (7 \le i \le 14),\quad S_{15}^0 = K_{15} || D_{15} || K_{23} || K_{31}.$$

3 Basic Ideas of Our Attacks

The core idea of our attacks is to find an input XOR difference for (K, IV) where some bits of (K, IV) are restricted to specific values. For example, we have the common XOR input difference where

$$\Delta K_0 = \texttt{0x10}, \Delta K_1 = \texttt{0x01}.$$

Then, we further restrict that

$$K_0[4] = 0, K_1[0] = 1.$$

In this way, when generating two inputs (K_0, K_1) and (K_0', K_1') satisfying $(\Delta K_0 = \texttt{0x10}, \Delta K_1 = \texttt{0x01})$, there are only in total 2^{14} rather than 2^{16} possible (K_0, K_1, K_0', K_1'). Indeed, the conditions

$$\Delta K_0 = \texttt{0x10}, \Delta K_1 = \texttt{0x01}, K_0[4] = 0, K_1[0] = 1$$

can be equivalently described with the signed difference, which reflects how the bit changes, as shown below:

$$\nabla K_0 = [\texttt{===n ====}],\ \nabla K_1 = [\texttt{==== ===u}].$$

One should observe that the signed difference is just a convenient tool to simultaneously describe the conditions on the XOR difference Δa and the conditions on some bits of a. Moreover, the signed difference is directly related to the modular difference. For example, for

$$\nabla K_0 = [\texttt{===n ====}],\ \nabla K_1 = [\texttt{==== ===u}],$$

we can always have

$$K_0' - K_0 = 2^4, K_1' - K_1 = -2^0.$$

However, if we simply consider the XOR difference, $K_0' - K_0$ and $K_1' - K_1$ can take several different values, i.e. they are not unique.

By using signed differences for (K, IV), we can know both the XOR differences and modular differences of the state words in LFSR, which makes it much easier to study the difference transitions through LFSR, BR and FSM. Hence, in our attacks, we aim to find a suitable signed difference for (K, IV), i.e., conditions on $(\Delta K, \Delta IV)$ and conditions on (K, IV), such that:

1. The difference of *some bits in LFSR* after several rounds has nonrandom statistic properties, which corresponds to the distinguishing attacks on the initialization phase.
2. The difference of *some bits in the first keystream word* after several rounds has nonrandom statistic properties, which corresponds to the key-recovery attacks on the actual stream cipher.

It can be found later that to attack as many rounds as possible, we should activate the S-boxes in FSM as late as possible, which is then reduced to the problem to find collisions. Moreover, at the very first few rounds, active S-boxes are allowed to appear in FSM, but we make the difference transitions in FSM at these rounds hold with probability 1 by carefully controlling some bits of (K, IV), which we later call the IV-correcting technique.

4 On Modular/XOR/Signed Differences

As the LFSR of ZUC-256 works in $GF(p)$, we first explain some basic relations between the modular difference, XOR difference and signed difference in $GF(p)$.

4.1 Relations Between δa and ∇a

Any element $a \in GF(p)$ can always be written as

$$a = \sum_{i=0}^{30} \mu_i \cdot 2^i \mod p,$$

where $\mu_i \in \{-1, 0, 1\}$. Note that $-1 = p - 1$ in $GF(p)$. For example, 0x2020 can be written as

$$\texttt{0x2020} = 2^{13} + 2^5 = 2^{14} - 2^{13} + 2^7 - 2^6 - 2^5 = \cdots,$$

i.e., there are many possible assignments to μ_i to represent the same $a \in GF(p)$. Moreover, given any assignment to μ_i for all $i \in [0, 30]$,

$$\sum_{i=0}^{30} \mu_i \cdot 2^i \mod p$$

must correspond to a unique element in $GF(p)$.

Since δa is an element in $GF(p)$, we can always write it as

$$\delta a = \sum_{i=0}^{30} \mu_i \cdot 2^i \mod p.$$

Then, we can build a relation between δa and ∇a with μ_i, as shown below:

$$\nabla a[i] = [\texttt{n}] \Leftrightarrow \mu_i = 1,\ \nabla a[i] = [\texttt{u}] \Leftrightarrow \mu_i = -1,\ \nabla a[i] = [\texttt{=}] \Leftrightarrow \mu_i = 0,$$

which is commonly used in the cryptanalysis of the MD-SHA hash family. In other words, ∇a is just another way to represent δa with some extra information of how the bit changes in (a, a') where $\delta a = a' \boxminus a$.

Fact 1. Given a signed difference ∇a, the modular difference δa is uniquely determined.

Fact 2. If we restrict that $\nabla a[i]$ takes either n or = for $0 \leq i \leq 30$, i.e. μ_i is restricted to $\{0, 1\}$, for a given modular difference δa, the signed difference ∇a is uniquely determined, i.e., $\nabla a[i] = \texttt{n}$ if $\delta a[i] = 1$ and $\nabla a[i] = \texttt{=}$ if $\delta a[i] = 0$.

4.2 A Relation Between δa and Δa

In this paper, we will intensively exploit the following relation between δa and Δa, as specified below:

Proposition 1. *To ensure that there exists a pair $a, a' \in GF(p)$ satisfies $\Delta a[j:i] = 0$ $(0 \leq i \leq j \leq 30)$ for a given (i, j), a necessary and sufficient condition is $\delta a[j:i] = 0$ or $\delta a[j:i] = 2^{j-i+1} - 1$.*

The proof is intuitive and we present it in Appendix A.2. We emphasize that it still requires some effort as the addition is modulo $2^{31} - 1$.

4.3 Relations in ∇a, Δa and δa

In this part, we describe some relations in ∇a, Δa and δa. For readers familiar with Wang et al.'s techniques [WLF+05, WY05, WYY05], the contents here may look trivial. However, they are important for the completeness of this paper. Note that the part will be heavy and we start it with some simple concrete examples. We also suggest readers first skip this heavy part and directly move to the next section to quickly grasp our attack framework and why we can improve the attacks [BM20, Tea21].

Problem : Supposing two elements (a, a') in $GF(p)$ satisfy $\delta a =$ `0x1`, what are the possible values of Δa, Δa_L and Δa_H? How to list all possible Δa, Δa_L and Δa_H?

Answer : This problem is very easy. First, we should know that there are 32 possible values of ∇a, 17 possible values of ∇a_L and 18 possible values of ∇a_H. Specifically, ∇a can take the following 32 possible values:

```
[=== ==== ==== ==== ==== ==== ==== ===n],
[=== ==== ==== ==== ==== ==== ==== ==nu],
...
[nuu uuuu uuuu uuuu uuuu uuuu uuuu uuuu],
[uuu uuuu uuuu uuuu uuuu uuuu uuuu uuu=].
```

One may feel confused of the last possible value of ∇a. It is correct because

$$-2^1 - 2^2 - 2^3 - \cdots - 2^{30} = -(2^{31} - 2) = -2^{31} + 2 + p = 1 \mod p.$$

Hence, Δa can take 32 values: $\{$`0x1`, `0x3`, `0x7`, $\cdots$, `0x7fffffff`, `0x7ffffffe`$\}$.

∇a_L can take the following 17 possible values:

```
[==== ==== ==== ===n], [==== ==== ==== ==nu],
[==== ==== ==== =nuu], [==== ==== ==== nuuu], ··· ,
[uuuu uuuu uuuu uuuu], [uuuu uuuu uuuu uuu=].
```

Hence, there are 17 possible values of Δa_L, which are $\{$`0x1`, `0x3`, `0x7`, $\cdots$, `0xffff`, `0xfffe`$\}$.

∇a_H can take the following 18 possible values:

```
[==== ==== ==== ====], [==== ==== ==== ===n],
[==== ==== ==== ==nu], [==== ==== ==== =nuu],
[==== ==== ==== nuuu], ··· ,
[nuuu uuuu uuuu uuuu], [uuuu uuuu uuuu uuuu].
```

However, there are 17 possible values of Δa_H, which are $\{$`0x0`, `0x1`, `0x3`, `0x7`, $\cdots$, `0xffff`$\}$.

Naïve Enumeration algorithms: For ZUC-256, we consider some slightly different problems because some state bits are fixed to constants according to the loading scheme. Specifically, we consider the element a in $GF(p)$ where some bits in the rightmost 16 bits of a are fixed to constants. For convenience, suppose $a_H[i] = c_i$ for $i \in \mathrm{SET_I} = \{i_1, ..., i_n\}$ $(1 \leq i_j \leq 15)$, where $c_1, c_2, \ldots, c_n$ are fixed binary constants.

Enumerating all Δa_H is described with the procedure `ENU-H` in Algorithm 1, where the set of all possible Δa_H is denoted by $\mathrm{SET_{\Delta a_H}}$.

Moreover, in our attacks, given an arbitrary δa, we need to enumerate all possible Δa_L with $\Delta a_H = 0$, which implies $\delta a_H \in \{\mathtt{0x0}, \mathtt{0xffff}\}$. Denote the set of all possible Δa_L by $\mathrm{SET_{\Delta a_L}}$. Finding $\mathrm{SET_{\Delta a_L}}$ is described with the procedure `ENU-L` in Algorithm 1.

Algorithm 1 Enumerating Δa_H and Δa_L

```
1:  procedure ENU-H($\delta a, \mathrm{SET_{\Delta a_H}}$)
2:      clear the set $\mathrm{SET_{\Delta a_H}}$.
3:      arr[2] = [0x0,0x7fff]
4:      for all $a_H$ where $n$ bits are fixed to $(c_1, c_2, \ldots, c_n)$ do
5:          for all $i$ where $0 \leq i \leq 1$ do
6:              $a = a_H \lll 16 + \mathtt{arr}[i]$,  $a' = a \boxplus \delta a$,  $\Delta a = a \oplus a'$
7:              if $\Delta a_H[i] = 0$ for $i \in \mathrm{SET_I}$ then
8:                  add $\Delta a_H$ to $\mathrm{SET_{\Delta a_H}}$
9:  procedure ENU-L($\delta a, \mathrm{SET_{\Delta a_L}}$)
10:     clear the set $\mathrm{SET_{\Delta a_L}}$.
11:     arr[2] = [0x0,0xffff]
12:     for all $i$ where $0 \leq i \leq 1$ do
13:         for all $j$ where $0 \leq j < 2^{15}$ do
14:             $a = \mathtt{arr}[i] \lll 16 + j$,  $a' = a \boxplus \delta a$,  $\Delta a = a \oplus a'$
15:             if $\Delta a_H = 0$ then
16:                 add $\Delta a_L$ to $\mathrm{SET_{\Delta a_L}}$
```

The core idea in Algorithm 1 is that we compute either the rightmost or the leftmost 16 bits by guessing the necessary carry, which is why we need to traverse the two elements stored in `arr`. Note that the addition is modulo p, which is why we also need to consider the carry when computing the leftmost 16 bits, as can be observed from the given concrete example.

Efficient checking algorithms: To search for valid input differences for our attacks, we always need to efficiently check whether two elements (a_H, a'_H) with $\Delta a_H \in \mathrm{SET_{\Delta a_H}}$ can lead to a given modular difference δa. Moreover, we assume that $a_H[i] = c_i$ for $i \in \mathrm{SET_I}$ has been satisfied, which can be easily checked. A more formal way to describe the problem is whether there are two 15-bit values b, b' such that

$$(a'_H \lll 16 + b') = (a_H \lll 16 + b) \boxplus \delta a.$$

For example, if $\delta a = \mathtt{0x1}$, it is impossible that there exists a pair (a, a') where $a_H = \mathtt{0x1fff}$ and $a'_H = \mathtt{0xefff}$ such that $\delta a = a' \boxminus a$. However, when $a'_H = \mathtt{0x0000}$ and $a_H = \mathtt{0xffff}$, it is possible. In both cases, we have $\Delta a_H = \mathtt{0xffff}$.

Observe that with `ENU-H` to compute $\mathrm{SET_{\Delta a_H}}$, we have traversed all possible pairs (a'_H, a_H) such that $a' = a \boxplus \delta a$. Based on this fact, the validity of (a'_H, a_H) can be efficiently checked, as shown in the procedure `VER-H` in Algorithm 2.

Two variant problems: In our attacks, we indeed also need to handle two slightly different problems. Specifically, given a modular difference δa satisfying $a[15:0] \in \{\mathtt{0}, \mathtt{0xffff}\}$,

Algorithm 2 Verification

```
1: procedure VER-H($\delta a, a_H, a'_H$)
2:     arr[2] = [0x0,0x7fff]
3:     for all $i$ where $0 \leq i \leq 1$ do
4:         $z = a_H \ll 16 + \texttt{arr}[i]$,  $z' = z \boxplus \delta a$
5:         if $z_H = a'_H$ then
6:             return TRUE
7:     return FALSE
```

how to enumerate all possible Δa_H where $\Delta a_L = 0$? In this case, how to efficiently check the validity of $(\delta a, a_H, a'_H)$?

The problems can be easily solved by slightly modifying Line 7 of Algorithm 1 and Line 5 of Algorithm 2. Specifically, the condition in Line 7 is replaced with "$\Delta a_H[i] = 0$ for $i \in \mathrm{SET_I}$ and $\Delta a_L = 0$". The condition in Line 5 is replaced with "$z_H = a'_H$ and $(z' \oplus z)_L = 0$". For convenience, the modified `ENU-H` and `VER-H` are called `ENU-H-M` and `VER-H-M`, respectively.

Enumerating Δa_L and Δa_H differently: For high automation of our program, we also need to deal with another problem. Specifically, for a given δa, how to find two sets $(\mathrm{SET_{\Delta a_H}}, \mathrm{SET_{\Delta a_L}})$ such that there always exist two elements (z, z') in $GF(p)$ such that

$$z_L \oplus z'_L = \Delta a_L, z_H \oplus z'_H = \Delta a_H, z' = z \boxplus \delta a$$

for each $\Delta a_H \in \mathrm{SET_{\Delta a_H}}$ and $\Delta a_L \in \mathrm{SET_{\Delta a_L}}$? This implicitly imposes a condition that $\Delta a_H[0] = \Delta a_L[15]$ as they correspond to the same bit of a in $GF(p)$.

If we can find such two sets, we can independently pick a value from $\mathrm{SET_{\Delta a_H}}$ and $\mathrm{SET_{\Delta a_L}}$ to form a 31-bit XOR difference Δa and we know that there must exist two elements (z, z') in $GF(p)$ such that

$$z \oplus z' = \Delta a, z' = z \boxplus \delta a.$$

For example, when $\delta a = \texttt{0x1}$, we can make

$$\mathrm{SET_{\Delta a_H}} = \{0\},\ \mathrm{SET_{\Delta a_L}} = \{\texttt{0x1}, \texttt{0x3}, \texttt{0x7}, \ldots, \texttt{0x7fff}\}.$$

However,

$$\mathrm{SET_{\Delta a_H}} = \{\texttt{0x0}, \texttt{0x1}\},\ \mathrm{SET_{\Delta a_L}} = \{\texttt{0x1}, \texttt{0x3}, \texttt{0x7}, \ldots, \texttt{0xffff}\}.$$

are not the desired sets because if we choose `0x1` from $\mathrm{SET_{\Delta a_H}}$, we can only choose `0xffff` from $\mathrm{SET_{\Delta a_L}}$. Moreover, if we choose `0x0` from $\mathrm{SET_{\Delta a_H}}$, we cannot choose `0xffff` from $\mathrm{SET_{\Delta a_L}}$. These should be very clear according to the signed differences.

Finding all the required $(\mathrm{SET_{\Delta a_H}}, \mathrm{SET_{\Delta a_L}})$ is described in the procedure `ENU-A` in Algorithm 3. The core idea of this algorithm is to guess the carry (depending on `caseNum`) for the computation of leftmost/rightmost 16 bits and to guess the shared bit (depending on `mid`) between the leftmost/rightmost 16 bits.

As the addition is modulo p, when $a + \delta a \geq p$, it is necessary to use $a + \delta a - 2^{31} + 1$ as the modular sum and we call such a situation **cycle-carry**. However, it can be observed in `ENU-A` that we assume cycle-carry exists only when $a + \delta a > p$. One reason is that for an arbitrary given δa, there is only one value of a satisfying $a + \delta a = p$ among all the $2^{31} - 1$ different values. We have carefully checked our input differences and there is no mistake in the used $(\mathrm{SET_{\Delta a_H}}, \mathrm{SET_{\Delta a_L}})$.

Algorithm 3 Enumerating two sets

```
1: procedure ENU-A($\delta a$)
2:     for all $i$ where $0 \leq i \leq 1$ do
3:         for all $j$ where $0 \leq j \leq 3$ do
4:             FIND-TWO-SETS($i, j, \mathrm{SET}_{\Delta \mathrm{a_H}}, \mathrm{SET}_{\Delta \mathrm{a_L}}, \delta a$)
5:             if both ($\mathrm{SET}_{\Delta \mathrm{a_H}}$ and $\mathrm{SET}_{\Delta \mathrm{a_L}}$) are nonempty then
6:                 valid ($\mathrm{SET}_{\Delta \mathrm{a_H}}, \mathrm{SET}_{\Delta \mathrm{a_L}}$) are found
7: procedure Find-Two-Sets(mid, caseNum, $\mathrm{SET}_{\Delta \mathrm{a_H}}, \mathrm{SET}_{\Delta \mathrm{a_L}}, \delta a$)
8:     for all $a_H$ where $n$ bits are fixed to $(c_1, c_2, \ldots, c_n)$ and $a_H[0] =$ mid do
9:         $z0 = z1 = a_H$, $z0' = a_H + \delta a_H$, $z1' = a_H + \delta a_H + 1$
10:        if caseNum$= 0$ and $\Delta z0[i] = 0$ for $i \in \mathrm{SET_I}$ and $z0' < 2^{16}$ then
11:            add $(z0 \oplus z0') \wedge$ 0xffff to $\mathrm{SET}_{\Delta \mathrm{a_H}}$
12:        if caseNum$= 1$ and $\Delta z1[i] = 0$ for $i \in \mathrm{SET_I}$ and $z1' < 2^{16}$ then
13:            add $(z1 \oplus z1') \wedge$ 0xffff to $\mathrm{SET}_{\Delta \mathrm{a_H}}$
14:        if caseNum$= 2$ and $\Delta z0[i] = 0$ for $i \in \mathrm{SET_I}$ and $z0' \geq 2^{16}$ then
15:            add $(z0 \oplus z0') \wedge$ 0xffff to $\mathrm{SET}_{\Delta \mathrm{a_H}}$
16:        if caseNum$= 3$ and $\Delta z1[i] = 0$ for $i \in \mathrm{SET_I}$ and $z1' \geq 2^{16}$ then
17:            add $(z1 \oplus z1') \wedge$ 0xffff to $\mathrm{SET}_{\Delta \mathrm{a_H}}$
18:    for all $a_L$ where $a_L[15] =$ mid do
19:        $z0 = z1 = a_L[14:0]$, $z0' = z_0 + \delta a_L[14:0]$, $z1' = z1 + \delta a_L[14:0] + 1$
20:        if caseNum$= 0$ and $z0' < 2^{15}$ then
21:            $a_L' = a_L + \delta a_L$, add $(a_L \oplus a_L') \wedge$ 0xffff to $\mathrm{SET}_{\Delta \mathrm{a_L}}$
22:        if caseNum$= 1$ and $z0' \geq 2^{15}$ then
23:            $a_L' = a_L + \delta a_L$, add $(a_L \oplus a_L') \wedge$ 0xffff to $\mathrm{SET}_{\Delta \mathrm{a_L}}$
24:        if caseNum$= 2$ and $z1' < 2^{15}$ then
25:            $a_L' = a_L + \delta a_L + 1$, add $(a_L \oplus a_L') \wedge$ 0xffff to $\mathrm{SET}_{\Delta \mathrm{a_L}}$
26:        if caseNum$= 3$ and $z1' \geq 2^{15}$ then
27:            $a_L' = a_L + \delta a_L + 1$, add $(a_L \oplus a_L') \wedge$ 0xffff to $\mathrm{SET}_{\Delta \mathrm{a_L}}$
```

Some definitions. We will make some definitions before introducing our attacks, as listed below:

Definition 1. *A signed difference ∇a_0 is said to be expanded from δa only when $\delta a_0 = \delta a$.*

For example, both the signed differences [`uuu uuuu uuuu uuuu uuuu uuuu uuuu uuu=`] and [`=== ==== ==== ==== ==== ==== ==== nuuu`] can be said to be expanded from the modular difference $\delta a =$ `0x1`.

Definition 2. *The Hamming weight of the signed difference ∇a denoted by $\mathbb{H}(\nabla a)$ is defined as the number of $\nabla a[i] \in \{\mathtt{n}, \mathtt{u}\}$ for $i \in [0, 30]$.*

For example, the signed difference $\nabla a =$ [`nn= ==== ==== ==== ==== ==== ==nu ===u`] has hamming weight 5, i.e., $\mathbb{H}(\nabla a) = 5$.

Definition 3. *The weight of the modular difference $\delta a \in GF(p)$ denoted by $\mathbb{W}(\delta a)$ is defined as the number of pairs (i, j) with $0 \leq i \leq j \leq 30$ satisfying one of the following conditions:*

Condition 1: $a[v] = 1$ $(v \in [i, j],\ a[i-1] = 0,\ a[j+1] = 0,\ i \neq 0,\ j \neq 30)$.

Condition 2: $a[v] = 1$ $(v \in [i, j],\ a[j+1] = 0,\ i = 0,\ j \neq 30)$.

Condition 3: $a[v] = 1$ $(v \in [i, j],\ a[i-1] = 0,\ i \neq 0,\ j = 30)$.

Definition 4. *The Hamming weight of the modular difference $\delta a \in GF(p)$ denoted by $H(\delta a)$ is defined as $min(\mathbb{W}(\delta a), \mathbb{W}(p - \delta a))$, where $min(x, y) = x$ if $x \leq y$ and $min(x, y) = y$ otherwise.*

For example, $H($`0x7fff`$) = 1$ and $H($`0x7fff7fff`$) = 1$.

5 Cancelling Differences Using Modular Differences

In a public design and evaluation report [ETS11] on ZUC-128, which was undertaken in response to the request made by 3GPP, it is written that:

“ [ETS11]Chosen IV/Key attacks target at the initialization stage of stream ciphers. For a good stream cipher, after the initialization, each bit of the IV/Key should contribute to each bit of the internal states, and any difference of the IV/Key will result in an almost-uniform and unpredictable difference of the internal states."

The attack scenario: The above statement may be not very clear. According to the description of the corresponding attacks on ZUC-128 in [ETS11], we can indeed interpret it. Specifically, the attack scenario means that an attacker can choose differences in the IV bits and key bits, and check after a certain number of initialization rounds whether the differences of some LFSR state bits have some undesirable properties (e.g., the distribution is non-uniform). If such properties can be detected, a distinguishing attack on the same number of initialization rounds can be claimed. The shortcut is to focus on how to detect undesirable properties in the state bits of S_{15}^{r}. If there are, we can claim an attack on $r + 15$ rounds due to $S_0^{r+15} = S_{15}^{r}$. In other words, we have 15 free rounds and this is why the currently claimed attacks [Tea21, BM20] can reach such a large number of rounds, i.e., 27 and 28. Note that this attack scenario is not commonly used in the cryptanalysis of stream ciphers. However, the ZUC team still claimed security in this attack scenario, even in the related-key setting [Tea21].

5.1 Revisiting Babbage-Maximov's 28-Round Attack

Regarding the above distinguishing attack, Babbage and Maximov proposed a 28-round distinguishing attack in [BM20] and the ZUC team also took into account this attack vector [Tea21] and proposed a 27-round attack. The basic idea is the same, which is to inject differences only in (S_2^0, S_{6L}^0). Note that for the old and new loading schemes, (S_2^0, S_{6L}^0) are only related to the key bits. Therefore, both the attacks work in the related-key setting. The following explanation for the 28-round attack will make it clear why the 27-round attack [Tea21] is implied in the 28-round attack.

For the completeness of this paper, we briefly describe how Babbage and Maximov found the input difference to mount distinguishing attacks in Appendix B.

The input difference used in the 28-round attack [BM20] is

$$\Delta S_2^0 = \texttt{0x01000000},\ \Delta S_6^0 = \texttt{0x00001010}.$$

After two clocks, (S_6^0, S_2^0) will be shifted to (S_4^2, S_0^2). Thus, at the 3rd clock, if $2^{20} \cdot \delta S_6^0 \boxplus 257 \cdot \delta S_2^0 \neq 0$, there will be $\Delta S_{15}^3 \neq 0$. For such a choice of $(\Delta S_6^0, \Delta S_2^0)$, we indeed can view it from the perspective of signed differences. Specifically, if

$$\begin{aligned}\nabla S_6^0 &= \texttt{=== ==== ==== ==== ===u ==== ===u ====},\\ \nabla S_2^0 &= \texttt{=== ===n ==== ==== ==== ==== ==== ====},\end{aligned}$$

or

$$\begin{aligned}\nabla S_6^0 &= \texttt{=== ==== ==== ==== ===n ==== ===n ====},\\ \nabla S_2^0 &= \texttt{=== ===u ==== ==== ==== ==== ==== ====},\end{aligned}$$

there must be $2^{20} \cdot \delta S_6^0 \boxplus 257 \cdot \delta S_2^0 = 0$.

When $2^{20} \cdot \delta S_6^0 \boxplus 257 \cdot \delta S_2^0 = 0$, $\Delta S_{15}^t = 0$ for $t \in [0, 6]$. Then, after 7 clocks, as $\delta S_0^6 = \delta S_6^0 \neq 0$, we have $\delta S_{15}^7 \neq 0$, $\delta S_i^7 = 0$ for $i \in [0, 14]$ and $(\Delta R_1^7 = 0, \Delta R_2^7 = 0)$. After 4 more clocks, i.e., after 11 clocks, S_{15}^7 is shifted to S_{11}^{11}. Therefore, at the 12th clock, active S-boxes will appear in FSM for the first time. At the 13th clock, as S_{15}^{13} is computed before updating FSM, the difference caused by FSM at the 12th clock will affect the difference of the state words in LFSR for the first time, i.e., ΔS_{15}^{13} is affected by the difference appearing in FSM.

In other words, we can equivalently say that ΔS_{15}^{13} is affected by only 1 round of update in FSM, where active S-boxes start appearing. Since $\Delta S_0^{28} = \Delta S_{15}^{13}$, it is reasonable to detect a biased linear relation in ΔS_0^{28} with a practical number of samples, i.e., only the one-round update in FSM needs to be approximated.

The above analysis also implies that the authors of [BM20] randomly picked both the key pair and IV pair for each sample in the experiments. Otherwise, if they randomly choose a key pair and fix it and then randomly pick many IV pairs, there will be cases (probability of $6/8 = 0.75$) that a valid biased linear relation for 28 initialization rounds cannot be detected as there are too many rounds of update in FSM required to be approximated.

Based on the above analysis, if we carefully choose a key pair satisfying $2^{20} \cdot \delta S_6^0 \boxplus 257 \cdot \delta S_2^0 = 0$ for each sample, i.e., according to their signed difference, it is expected that the bias can be improved. To support this claim, we repeated the experiments by always choosing a key pair which can make $2^{20} \cdot \delta S_6^0 \boxplus 257 \cdot \delta S_2^0 = 0$ and found that

$$Pr[\Delta S_0^{28}[9] \oplus \Delta S_0^{28}[10] = 1] \approx 0.5 - 2^{-8.6},$$

while $Pr[\Delta S_0^{28}[9] \oplus \Delta S_0^{28}[10] = 1] \approx 0.5 - 2^{-10.46}$ in [BM20].

The weak-key setting: Note that due to the usage of signed differences, our new method naturally works in the weak-key setting. However, as explained above, the so-called 28-round attack [BM20] indeed only works in the weak-key setting due to the constraint $2^{20} \cdot \delta S_6^0 \boxplus 257 \cdot \delta S_2^0 = 0$. Similarly, the 27-round attack [Tea21] also only works in the weak-key setting given each specified XOR input difference because it also relies on the same simple constraint $2^{20} \cdot \delta S_6^0 \boxplus 257 \cdot \delta S_2^0 = 0$.

5.2 More Observations

The above observation reveals that using signed differences rather than simple XOR differences will lead to a better bias. This is because signed differences are directly related to modular differences, which can be cancelled with probability 1 due to the modular addition in LFSR. To further improve the attack, we carefully investigated the round update function of ZUC-256 and found some extra important observations.

The first observation: The first observation is that we can study the distribution of δS_{15}^t rather than ΔS_{15}^t if targeting a $(t + 15)$-round distinguisher. This observation has been confirmed via experiments. Specifically, with the key difference discovered in [BM20], we repeated the experiments by always choosing a key pair which can make $2^{20} \cdot \delta S_6^0 \boxplus 257 \cdot \delta S_2^0 = 0$. Instead of collecting the distribution of ΔS_{15}^{13}, we collected the distribution of δS_{15}^{13} and eventually found the following biased linear relation:

$$Pr[\delta S_0^{28}[11] = 1] \approx 0.5 + 2^{-6}.$$

Obviously, this further improves the bias.

The second observation: When targeting the distinguisher reaching the largest number of rounds, according to our analysis, it is inevitable to activate some 8-bit S-boxes and the S-boxes are applied in parallel to a 32-bit state word in FSM. In addition, there is a 1-bit right shift operation on W^{t-1} at the t-th clock, whose value is highly related to the two registers in FSM. Therefore, we will only treat the following four different types of linear relations as potential biased linear relations when targeting an attack on $15 + t$ rounds:

1. The first type of linear relations are only in terms of $\delta S_{15}^t[i]$ for $i \in [0, 14]$.
2. The second type of linear relations are only in terms of $\delta S_{15}^t[i]$ for $i \in [7, 22]$.
3. The third type of linear relations are only in terms of $\delta S_{15}^t[i]$ for $i \in [15, 30]$.
4. The fourth type of linear relations are only in terms of $\delta S_{15}^t[i]$ for $i \in \{i|i \in [0, 6]\} \cup \{i|i \in [23, 30]\}$, where $\cup$ is the union of sets.

Another benefit is that the memory complexity can be reduced from 2^{31} to about 3×2^{16} as we no more need to store the full distribution table[1] of δS_{15}^t, i.e., storing the number of times that δS_{15}^t takes the value i for each $i \in GF(p)$. Instead, we only need to use 4 smaller tables to store the number of occurrences of $\delta S_{15}^t[14 : 0]$, $\delta S_{15}^t[22, 7]$, $\delta S_{15}^t[30 : 15]$ and $\delta S_{15}^t[6 : 0]||\delta S_{15}^t[30 : 23]$, respectively. The reduction in memory complexity also allows to efficiently use multi-threaded programming as each thread only consumes negligible memory.

5.3 Strategies to Inject Differences

With all the above observations in mind, we start considering whether it is possible to use complex input differences to significantly improve the attack by fully utilizing the degrees of freedom provided by the 256-bit key. To reach as many rounds as possible, the following critical observation on the round update function will play a vital role to guide us to select the best strategy to inject differences.

[1]In [BM20], it is necessary to store it in order to detect a biased linear relation from it via Walsh-Hadamard Transform (WHT).

A critical observation on the round update function: As the active S-boxes will significantly decrease the bias of a potential linear relation, it is necessary to make the active S-boxes appear as late as possible. Suppose after t_0 clocks, $S_{15}^{t_0}$ is activated for the first time, i.e., $\Delta S_{15}^{t} = 0$ for $t \in [0, t_0 - 1]$ and $\Delta S_{15}^{t_0} \neq 0$.

Then, after 4 more clocks, we have $S_{11}^{t_0+4} = S_{15}^{t_0}$. If $\Delta S_{11L}^{t_0+4} \neq 0$, at clock $t_0 + 5$, during the update in FSM, the active S-boxes will appear. Therefore, for a good input difference, $\Delta S_{15}^{t_0}$ should satisfy the following constraint after t_0 clocks. In this way, at clock $t_0 + 5$, no active S-box will appear even if $\Delta S_{15}^{t_0} \neq 0$.

$$\Delta S_{15L}^{t_0} = 0, \Delta S_{15H}^{t_0} \neq 0.$$

Indeed, we can further impose that after $t_0 + 1$ clocks, $\Delta S_{15}^{t_0+1}$ should satisfy

$$\Delta S_{15L}^{t_0+1} = 0, \Delta S_{15H}^{t_0+1} \neq 0.$$

In this way, at clock t_0+6, still no active S-box appears in FSM since $\Delta S_{11L}^{t_0+5} = \Delta S_{15L}^{t_0+1}$. In other words, only starting from clock $t_0 + 7$, the active S-boxes will appear since $\Delta S_{9H}^{t_0+6} = \Delta S_{15H}^{t_0} \neq 0$. Without the above constraints on $\Delta S_{15}^{t_0}$, the active S-boxes will appear starting from clock $t_0 + 5$. Without the further constraints on $\Delta S_{15}^{t_0+1}$, the active S-boxes will appear starting from clock $t_0 + 6$. Therefore, by properly choosing an input difference, there is a great potential to extend a simple attack by two rounds, where only 1 round of update in FSM is required to be approximated. An intuitive explanation can be referred to Figure 2.

Based on the above analysis, it is now clear that to reach as many rounds as possible, it is necessary to identify an input difference such that t_0 is as large as possible and that the above constraints on $\Delta S_{15}^{t_0}$ and $\Delta S_{15}^{t_0+1}$ should hold.

According to Proposition 1, to ensure $\Delta S_{15L}^{t_0} = 0$ and $\Delta S_{15L}^{t_0+1} = 0$, there must be $\delta S_{15L}^{t_0} \in \{0, \texttt{0xffff}\}$ and $\delta S_{15L}^{t_0+1} \in \{0, \texttt{0xffff}\}$. However, we emphasize that even if $\delta a_L \in \{0, \texttt{0xffff}\}$, it is still possible that $\Delta a_L \neq 0$ since there still exist some signed differences expanded from δa such that $\nabla a[i] \in \{\texttt{n}, \texttt{u}\}$ for some $i \in [0, 15]$.

However, if $\delta S_{15L}^{t_0} \in \{0, \texttt{0xffff}\}$ and $\delta S_{15L}^{t_0+1} \in \{0, \texttt{0xffff}\}$ does not hold, there could not be $\Delta S_{15L}^{t_0} = 0$ or $\Delta S_{15L}^{t_0+1} = 0$. In other words, if $\delta S_{15L}^{t_0} \in \{0, \texttt{0xffff}\}$ and $\delta S_{15L}^{t_0+1} \in \{0, \texttt{0xffff}\}$ hold, it is possible to have $\Delta S_{15L}^{t_0} = 0$ and $\Delta S_{15L}^{t_0+1} = 0$ in sufficiently many samples. In addition, for some δa, there is a high probability that $\Delta a_L = 0$, e.g. $\delta a = \texttt{0x10000}$.

Can we simply improve the attack? The above analysis is simple. Indeed, the 28-round related-key distinguisher in [BM20] is obtained without the above constraints taken into account. Then, it is natural to ask whether we can slightly modify the input difference in [BM20] to attack more rounds. Specifically, is there an input difference $(\delta S_2^0, \delta S_6^0)$ such that $2^{20} \cdot \delta S_6^0 \boxplus 257 \cdot \delta S_2^0 = 0$, $\delta S_2^0[22 : 16] \in \{0, \texttt{0x7f}\}$, $\delta S_{6H}^0 \in \{0, \texttt{0xffff}\}$ and $\delta S_{15L}^7 \in \{0, \texttt{0xffff}\}$, where $\delta S_{15}^7 = 257 \cdot \delta S_6^0$?

A simple loop for $2^{16} - 2$ possible values of δS_6^0 shows that there does not exist a non-zero δS_6^0 which can make all the above conditions hold.

Advanced strategies: inject differences in 11 state words in LFSR. As stated before, to reach as many rounds as possible, it is necessary to make t_0 as large as possible such that $\delta S_{15}^{t_0} \neq 0$ and $\delta S_{15}^{i} = 0$ for $i \in [0, t_0 - 1]$. In addition, there should be $\delta S_{15L}^{t_0} \in \{0, \texttt{0xffff}\}$ and $\delta S_{15L}^{t_0+1} \in \{0, \texttt{0xffff}\}$. In this way, it is expected that we can find a biased linear relation in $\delta S_0^{t_0+23} = \delta S_{15}^{t_0+4+2+2} = \delta S_{15}^{t_0+8}$ by pure simulations as only 1 round of update in FSM needs to be approximated. In other words, it is possible to construct a distinguisher for up to $(t_0 + 23)$ initialization rounds with practical time complexity.

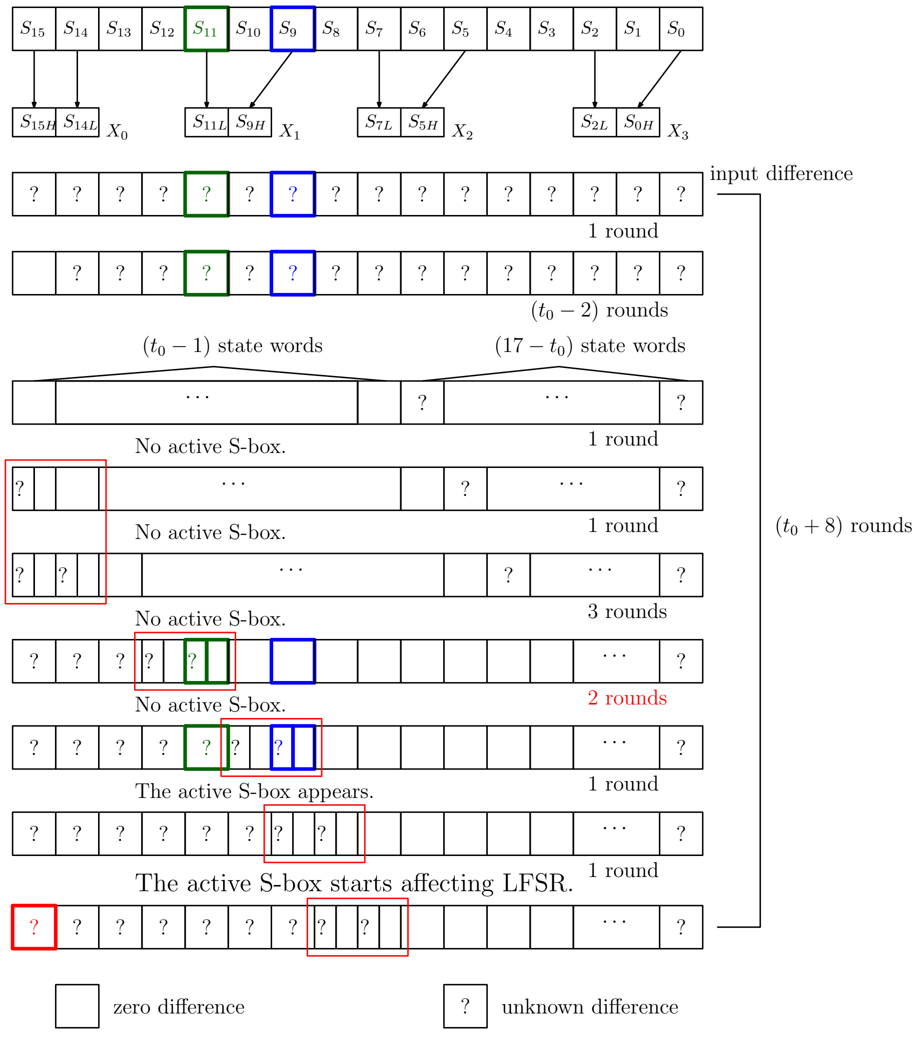

Figure 2: The big picture of our attacks.

After careful analysis, for ZUC-256, we choose $t_0 = 8$ and will inject differences in S_i^0 for $i \in [0, 10]$. If there is a solution to δS_i^0 $(i \in [0, 10])$, we can expect a practical attack on 31 rounds of ZUC-256.

For ZUC-256-v2, due to the fact that too many state bits of S_i^0 are restricted to constants, we choose $t_0 = 7$ and will again inject differences in S_i^0 for $i \in [0, 10]$. If there exists a solution to δS_i^0 $(i \in [0, 10])$, we can expect a practical attack on 30 rounds of ZUC-256-v2.

The pattern of the input difference is depicted in Figure 3.

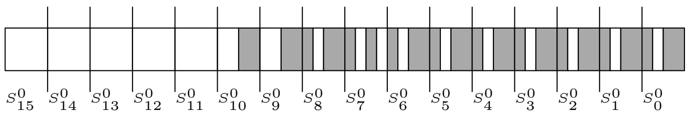

Figure 3: The illustration of the input difference (marked in gray).

The big picture of our attacks: In a word, we expect to find an input difference satisfying the pattern illustrated in Figure 3 such that

$$\begin{aligned} \Delta S_{15}^t &= 0 \text{ for } t \in [0, t_0 - 1], \\ \Delta S_{15L}^{t_0} &= 0, \\ \Delta S_{15L}^{t_0+1} &= 0 \end{aligned}$$

hold with a probability close to 1. This has already been illustrated in Figure 2. As the input difference is so complex and the constraints are so strong, finding such an input difference requires lots of effort and is rather challenging. Indeed, it is not difficult to observe this problem is very similar to finding collisions. In the following, we will describe how to construct equations to make these constraints hold and how to solve the corresponding equations.

5.4 More Details of the Strategies

To mount the attack on 31 rounds of ZUC-256, the problem now becomes how to find δS_i^0 $(i \in [0, 10])$ such that $\delta S_{15}^t = 0$ $(t \in [1, 7])$. To achieve this, we need to consider the following conditions:

Clock 1: At the first clock, it is required that

$$\begin{aligned} 2^{21} \cdot \delta S_{10}^0 \boxplus 2^{20} \cdot \delta S_4^0 \boxplus 257 \cdot \delta S_0^0 &= 0, \\ \Delta S_{5H}^0 &\neq 0, \\ \Delta S_{7L}^0 &= 0, \\ \Delta S_{9H}^0 &= 0. \end{aligned}$$

In this way, after the first clock, $\delta S_{15}^1 = 0$ holds. In addition, $\Delta R_1^1 = 0$ and $\Delta R_2^1 \neq 0$, i.e., there will be differences appearing in FSM.

Clock 2: At the second clock, it is required that

$$\begin{aligned} ((R_2^1 \oplus \Delta R_2^1) \gg 1) \boxminus (R_2^1 \gg 1) \boxplus 2^{20} \cdot \delta S_5^0 \boxplus 257 \cdot \delta S_1^0 &= 0, \\ \Delta S_{8L}^0 &= \Delta R_{2H}^1, \\ \Delta S_{10H}^0 &= 0. \end{aligned}$$

In this way, after the second clock, $\Delta R_1^2 = 0$ and $\Delta R_2^2 \neq 0$.

Clock 3: At the 3rd clock, we need

$$\begin{aligned}((R_2^2 \oplus \Delta R_2^2) \gg 1) \boxminus (R_2^2 \gg 1) \boxplus 2^{20} \cdot \delta S_6^0 \boxplus 257 \cdot \delta S_2^0 &= 0,\\ \Delta S_{9L}^0 &= \Delta R_{2H}^2.\end{aligned}$$

Again, after 3 clocks, $\delta S_{15}^3 = 0$, $\Delta R_1^3 = 0$ and $\Delta R_2^3 \neq 0$.

Clock 4: At the 4th clock, we need

$$\begin{aligned}((R_2^3 \oplus \Delta R_2^3) \gg 1) \boxminus (R_2^3 \gg 1) \boxplus 2^{20} \cdot \delta S_7^0 \boxplus 257 \cdot \delta S_3^0 &= 0,\\ \Delta S_{10L}^0 &= \Delta R_{2H}^3,\\ \Delta S_{8H}^0 &= \Delta R_{2L}^3.\end{aligned}$$

Similarly, there will be $\delta S_{15}^4 = 0$. Due to the last two equations, $\Delta R_1^4 = 0$ and $\Delta R_2^4 = 0$ will hold. This implies that the difference in FSM is cancelled after 4 clocks.

Clock 5: At the 5th clock, the conditions become much simpler, as shown below:

$$\begin{aligned}2^{20} \cdot \delta S_8^0 \boxplus 257 \cdot \delta S_4^0 &= 0,\\ \Delta S_{9H}^0 &= 0.\end{aligned}$$

In this way, $\delta S_{15}^5 = 0$, $\Delta R_1^5 = 0$ and $\Delta R_2^5 = 0$.

Clock 6: At the 6th clock, we need

$$\begin{aligned}2^{20} \cdot \delta S_9^0 \boxplus 257 \cdot \delta S_5^0 &= 0,\\ \Delta S_{10H}^0 &= 0.\end{aligned}$$

Then, $\delta S_{15}^6 = 0$, $\Delta R_1^6 = 0$ and $\Delta R_2^6 = 0$.

Clock 7: At the 7th clock, we need the following equation to make $\delta S_{15}^7 = 0$.

$$2^{20} \cdot \delta S_{10}^0 \boxplus 257 \cdot \delta S_6^0 = 0.$$

Clock 8: At the 8th clock, it is required that

$$(257 \cdot \delta S_7^0)[15:0] \in \{0, \texttt{0xffff}\}.$$

Clock 9: At the 9th clock, it is required that

$$(2^{15} \cdot (257 \cdot \delta S_7^0) \boxplus 257 \cdot \delta S_8^0)[15:0] \in \{0, \texttt{0xffff}\}.$$

With all the above conditions satisfied, we can expect to find an attack on 31 rounds of ZUC-256. It is not difficult to imagine that the most technical and difficult part is how to cancel the difference in FSM after 4 clocks, where XOR differences, modular differences and value transitions are involved. For better understanding, how to cancel the difference in FSM after 4 clocks is depicted in Figure 4.

The obstacle to attack 32 or more initialization rounds: After presenting the strategy for the 31-round attack, it is natural to ask whether we have tried to attack 32 initialization rounds by making $t_0 = 9$. Indeed, we have made some analysis of it. However, choosing $t_0 = 9$ implies that $2^{20} \cdot \delta S_{11}^0 \boxplus 257 \cdot \delta S_7^0 = 0$ should hold. As $\delta S_7^0 \neq 0$, it is necessary to have $\delta S_{11}^0 \neq 0$. Obviously, we expect that $\Delta S_{11H}^0 = 0$ in order that the difference in FSM can be cancelled as early as possible, just as in our 31-round attack where $\Delta S_{9H}^0 = 0$

and $\Delta S^0_{10H} = 0$. Otherwise, whether it is possible to cancel the difference in FSM is questionable. If $\Delta S^0_{11H} = 0$, there must be $\Delta S^0_{11L} \neq 0$. Then, we need to cancel the difference in FSM after 5 clocks, which is one more clock than that in the 31-round attack. However, at the fifth clock, there should also be $\Delta S^0_{9H} \neq 0$ in order to fully cancel the difference in FSM. This is due to the MDS property of the linear transform L_2. Specifically, supposing $a = (a_3, a_2, a_1, a_0) \in \mathbb{F}^4_{2^8}$ and $b = (b_3, b_2, b_1, b_0) \in \mathbb{F}^4_{2^8}$ are the input and output of L_2, respectively, when there are two bytes in a that are zero, there will be at least 3 non-zero bytes in b. Once $\Delta S^0_{9H} \neq 0$, it is required to cancel the non-zero difference caused by the two registers in FSM. Currently, we cannot find a feasible way to handle the propagation of the differences in both registers in FSM. Hence, we leave it as an open problem to further extend our attacks to more rounds, e.g. the full 33 rounds.

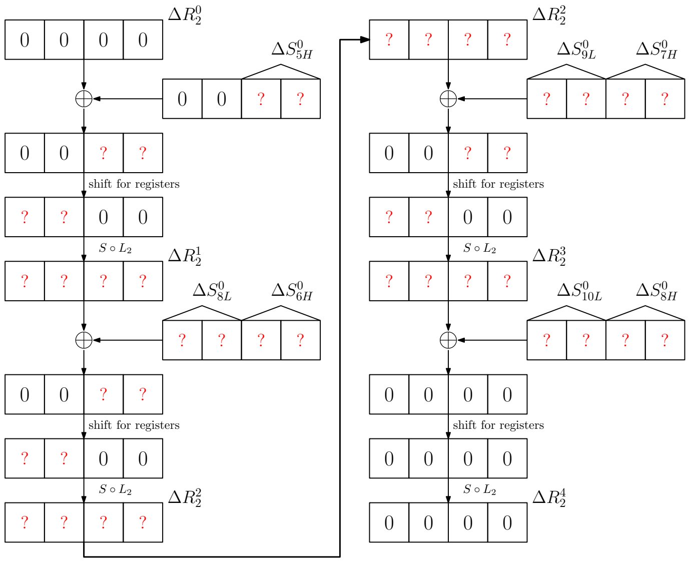

Figure 4: The difference transitions in FSM for the first 4 clocks.

Tweaking the strategy for ZUC-256-v2: Another question that naturally arises is whether it is possible to apply this strategy to 31 initialization rounds of ZUC-256-v2. Note that in this case we need $(257 \cdot \delta S^0_7)[15:0] \in \{0, \texttt{0xffff}\}$ and $\Delta S^0_{7L} = 0$. Due to the modification of the loading scheme, $\Delta S^0_7[22:16] = 0$ should also hold as these 7 bits are constant. However, for the old loading scheme, we only need to ensure 1-bit extra condition $\Delta S^0_7[22] = 0$. In other words, there are at most $2^9 - 2$ possible non-zero values left for δS^0_7, i.e., $\delta S^0_7[22:0] \in \{0, \texttt{0x7fffff}\}$. A simple loop for all possible values of δS^0_7 suggests that there does not exist a value satisfying $(257 \cdot \delta S^0_7)[15:0] \in \{0, \texttt{0xffff}\}$. Consequently, the above strategy cannot be simply applied to 31 initialization rounds of ZUC-256-v2. However, we emphasize that this does not prove the resistance against this attack vector as there may exist some more advanced strategies to inject differences and to control the difference transitions in FSM.

The strategy to inject differences for 30-round ZUC-256-v2: Since the 31-round attack fails for ZUC-256-v2, we turn to the attack on 30-round ZUC-256-v2. The overall strategy to inject differences is the same, i.e., the difference will be still injected in 11 state words, i.e., S^0_i for $i \in [0, 10]$. Specifically, the conditions at clock i for $i \in [1, 6]$ are the same as that in the 31-round attack. For clock 7 and clock 8, we need to modify the conditions as follows:

1. At the 7th clock, we need

$$(2^{20} \cdot \delta S_{10}^0 \boxplus 257 \cdot \delta S_6^0)[15:0] \quad \in \quad \{0, \texttt{0xffff}\}.$$

2. At the 8th clock, it is required that

$$(2^{15} \cdot (2^{20} \cdot \delta S_{10}^0 \boxplus 257 \cdot \delta S_6^0) \boxplus 257 \cdot \delta S_7^0)[15:0] \quad \in \quad \{0, \texttt{0xffff}\}.$$

As for clock 9, we no longer add strict conditions. One may ask why we need to inject a difference at S_{10}^0 in the 30-round attack since the following conditions also have the potential to reach 30 rounds.

$$\begin{aligned}(257 \cdot \delta S_6^0)[15:0] \quad &\in \quad \{0, \texttt{0xffff}\},\\ (2^{15} \cdot (257 \cdot \delta S_6^0) \boxplus 257 \cdot \delta S_7^0)[15:0] \quad &\in \quad \{0, \texttt{0xffff}\}.\end{aligned}$$

However, not injecting a difference in S_{10}^0 also implies that the difference in FSM should be cancelled after 3 clocks due to the MDS property of L_2, i.e., $\Delta R_2^3 = 0$ and $\Delta S_{8H}^0 = 0$. Then, there will be the following condition as $\Delta R_2^3 = 0$:

$$2^{20} \cdot \delta S_7^0 \boxplus 257 \cdot \delta S_3^0 \quad = \quad 0.$$

Note that for the new loading scheme, $\Delta S_i^0[22:16] = 0$ for $i \in [0, 15]$ as these 7 bits are constant. Hence, $\delta S_3^0[22:16] \in \{0, \texttt{0x7f}\}$, $\delta S_6^0[22:16] \in \{0, \texttt{0x7f}\}$ and $\delta S_7^0[22:0] \in \{0, \texttt{0x7fffff}\}$. A simple loop for the $2^9 - 2$ possible values of δS_7^0 and the $2^{16} - 2$ possible values of $(257 \cdot \delta S_6^0)$ indicates that there does not exist any valid solution to $(\delta S_3^0, \delta S_6^0, \delta S_7^0)$ satisfying the above three constraints.

Therefore, we need to inject a difference in S_{10}^0 as it relaxes the constraint on $(\delta S_7, \delta S_3)$. Specifically, there will be $\Delta R_2^3 \neq 0$ and the constraint on $(\delta S_7^0, \delta S_3^0)$ becomes

$$((R_2^3 \oplus \Delta R_2^3) \gg 1) \boxminus (R_2^3 \gg 1) \boxplus 2^{20} \cdot \delta S_7^0 \boxplus 257 \cdot \delta S_3^0 \quad = \quad 0.$$

Another benefit to use this strategy to attack 30-round ZUC-256-v2 is that we can reuse the code to search for the input differences to attack 31-round ZUC-256 since the core problem is the same, i.e., how to cancel the differences in FSM after 4 clocks. In the following, we will describe how to tackle this core problem.

5.5 Searching for Valid Differences

As explained above, to mount attacks on 31-round ZUC-256 and 30-round ZUC-256-v2, respectively, it is necessary to use complex input differences satisfying a set of equations. The equations are rather complicated as the modular difference, the XOR difference and the value transitions are all involved. To efficiently find a solution to these equations, we utilize a three-step method, as stated below:

Step 1: Pick a solution to the modular differences $(\delta S_0^0, \delta S_4^0, \delta S_8^0, \delta S_{10}^0, \delta S_6^0, \delta S_7^0)$ that does not contradict the equations. Then, based on the enumeration algorithms described in Subsection 4.3, compute the set of XOR differences $\mathrm{SET}_{\Delta S_{6H}^0}$, $\mathrm{SET}_{\Delta S_{7H}^0}$, $\mathrm{SET}_{\Delta S_{10L}^0}$, $(\mathrm{SET}_{\Delta S_{8H}^0}, \mathrm{SET}_{\Delta S_{8L}^0})$ for ΔS_{6H}^0, ΔS_{7H}^0, ΔS_{10L}^0 and $(\Delta S_{8H}^0, \Delta S_{8L}^0)$, respectively, where $(\Delta S_{8H}^0, \Delta S_{8L}^0)$ can always correspond to a valid signed difference expanded from δS_8^0.

Step 2: Pick a solution to δS_9^0 such that $\Delta S_{9H}^0 = 0$ and compute $\delta S_5^0 = 257^{-1} \cdot (p \boxminus 2^{20} \cdot \delta S_9^0)$. According to ENU-L, compute the set of all possible ΔS_{9L}^0 denoted by $\mathrm{SET}_{\Delta S_{9L}^0}$. According to ENU-H, compute the set of all possible ΔS_{5H}^0 denoted by $\mathrm{SET}_{\Delta S_{5H}^0}$

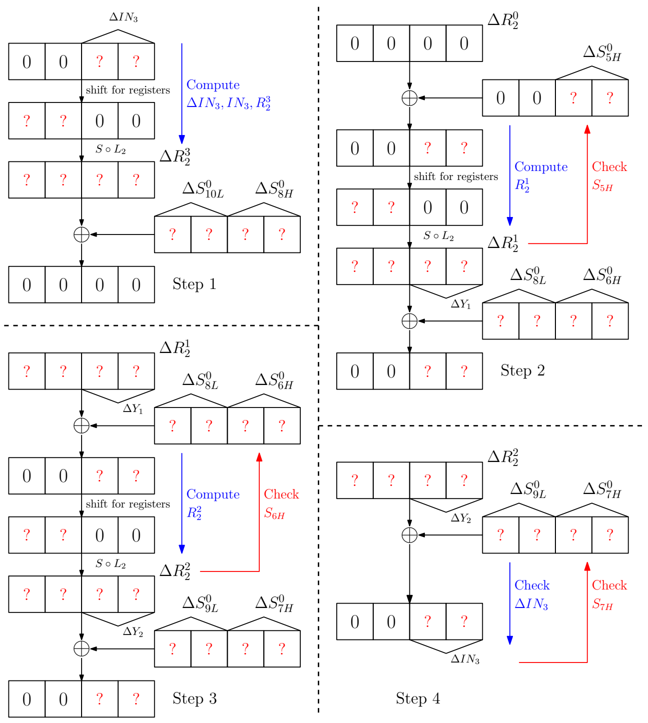

Figure 5: The procedure to find valid difference transitions and value transitions in FSM for the first 4 clocks.

Step 3: Only $(\delta S_1^0, \delta S_2^0, \delta S_3^0)$ are unknown. To determine whether there exists a solution to $(\delta S_1^0, \delta S_2^0, \delta S_3^0)$ and to find the solution if there exists one, `Procedure-DiCancel` [described in the following part] will be called, which is used to find valid difference transitions and value transitions in FSM such that the differences in FSM can be cancelled after 4 clocks. If there is no output in `Procedure-DiCancel`, move to Step 2. Otherwise, a solution to the input difference is found.

We emphasize that the values of $(\delta S_0^0, \delta S_4^0, \delta S_8^0, \delta S_{10}^0, \delta S_6^0, \delta S_7^0, \delta S_9^0)$ will be carefully picked, the details of which will be explained later. In the following, we mainly focus on how to cancel the differences in FSM after 4 clocks given the knowledge of $(\delta S_5^0, \delta S_6^0, \delta S_7^0)$ and the set of XOR differences: $\mathrm{SET}_{\Delta \mathrm{S}_{5\mathrm{H}}^0}$, $\mathrm{SET}_{\Delta \mathrm{S}_{6\mathrm{H}}^0}$, $\mathrm{SET}_{\Delta \mathrm{S}_{7\mathrm{H}}^0}$, $(\mathrm{SET}_{\Delta \mathrm{S}_{8\mathrm{H}}^0}, \mathrm{SET}_{\Delta \mathrm{S}_{8\mathrm{L}}^0})$, $\mathrm{SET}_{\Delta \mathrm{S}_{9\mathrm{L}}^0}$ and $\mathrm{SET}_{\Delta \mathrm{S}_{10\mathrm{L}}^0}$.

Cancelling the differences in FSM after the first 4 clocks: Given the knowledge of the above modular differences and the sets of XOR differences, in the following, we will describe how to find valid solutions of (R_2^1, R_2^2, R_2^3), $(\Delta R_2^1, \Delta R_2^2, \Delta R_2^3)$ and $(\delta S_1^0, \delta S_2^0, \delta S_3^0)$ satisfying

$$\begin{aligned}
((R_2^1 \oplus \Delta R_2^1) \gg 1) \boxminus (R_2^1 \gg 1) \boxplus 2^{20} \cdot \delta S_5^0 \boxplus 257 \cdot \delta S_1^0 &= 0,\\
((R_2^2 \oplus \Delta R_2^2) \gg 1) \boxminus (R_2^2 \gg 1) \boxplus 2^{20} \cdot \delta S_6^0 \boxplus 257 \cdot \delta S_2^0 &= 0,\\
((R_2^3 \oplus \Delta R_2^3) \gg 1) \boxminus (R_2^3 \gg 1) \boxplus 2^{20} \cdot \delta S_7^0 \boxplus 257 \cdot \delta S_3^0 &= 0,\\
\Delta S_{8L}^0 &= \Delta R_{2H}^1,\\
\Delta S_{9L}^0 &= \Delta R_{2H}^2,\\
\Delta S_{10L}^0 &= \Delta R_{2H}^3,\\
\Delta S_{8H}^0 &= \Delta R_{2L}^3.
\end{aligned}$$

For better understanding, we recommend to refer to Figure 5 when reading this part. It should be emphasized that there are conditions on some bits of $(S^0_{5H}, S^0_{6H}, S^0_{7H})$ imposed by the constant bits. For ZUC-256, the conditions are

$$S^0_{5H}[7] = 1, \quad S^0_{6H}[7] = 1, \quad S^0_{7H}[7] = 1.$$

For ZUC-256-v2, the conditions are

$$S^0_{5H}[7:1] = D_5, \quad S^0_{6H}[7:1] = D_6, \quad S^0_{7H}[7] = D_7.$$

For simplicity, denote these conditions on $(S^0_{5H}, S^0_{6H}, S^0_{7H})$ by $(\text{Con}_5, \text{Con}_6, \text{Con}_7)$.

The whole procedure can be divided into 4 steps, as detailed below. In general, the main idea is to use the depth-first search and a meet-in-the-middle strategy. Let us call this procedure `Procedure-DiCancel`.

Step 1: Handle the difference transitions at the 3rd clock, i.e., make $\Delta R^3_2 = \Delta S^0_{10L}||\Delta S^0_{8H}$. For simplicity, let $\Delta IN_3 = \Delta R^2_{2L} \oplus \Delta S^0_{7H}$, i.e., ΔIN_3 is a 16-bit value. Traverse all the 2^{16} possible values of ΔIN_3 and compute $\Delta T_3 = L_2(\Delta IN_3 \lll 16)$ for each ΔIN_3. For each ΔT_3, traverse all possible $\Delta S^0_{10L}||\Delta S^0_{8H}$ where $\Delta S^0_{10L} \in \text{SET}_{\Delta S^0_{10L}}$ and $\Delta S^0_{8H} \in \text{SET}_{\Delta S^0_{8H}}$. For each pair $(\Delta T_3, \Delta S^0_{10L}||\Delta S^0_{8H})$, check whether $\Delta T_3 \to \Delta S^0_{10L}||\Delta S^0_{8H}$ is a valid difference transition according to the differential distribution table (DDT) of the used 4 parallel S-boxes. If it is a valid difference transition, compute the corresponding pair of outputs $(R^3_2, R^3_2 \oplus (\Delta S^0_{10L}||\Delta S^0_{8H}))$ satisfying this difference transition and compute δS^0_3 as follows:

$$\delta S^0_3 \quad = \quad 257^{-1} \cdot (p \boxminus (((R^3_2 \oplus (\Delta S^0_{10L}||\Delta S_{8H})) \gg 1) \boxminus (R^3_2 \gg 1) \boxplus 2^{20} \cdot \delta S^0_7)).$$

If $\delta S^0_3[22:16] \in \{0, \texttt{0x7f}\}$, compute $IN_3 = (L_2^{-1} \circ S^{-1}(R^3_2)) \gg 16$ and insert the tuple $(\Delta T_3, R^3_2, IN_3, \delta S^0_3, \Delta S^0_{10L}, \Delta S^0_{8H})$ into the ΔIN_3-th row of the 2-dimensional array $\texttt{ARR}_3$. Otherwise, try another valid pair of outputs. If all valid pairs of outputs are traversed, consider the next candidate of $\Delta S^0_{10L}||\Delta S^0_{8H}$ and repeat the same procedure. After all the possible values of ΔIN_3 are traversed, move to Step 2.

Step 2: Handle the difference transitions at the 1st clock. Specifically, traverse each element in $\text{SET}_{\Delta S^0_{5H}}$. For each $\Delta S^0_{5H} \in \text{SET}_{\Delta S^0_{5H}}$, compute $\Delta T_1 = L_2(\Delta S^0_{5H} \lll 16)$. For each ΔT_1, traverse all possible $\Delta S^0_{8L}||\Delta Y_1$ where $\Delta S^0_{8L} \in \text{SET}_{\Delta S^0_{8L}}$ and $\Delta Y_1 \in [0, 2^{16}-1]$. For each pair $(\Delta T_1, \Delta S^0_{8L}||\Delta Y_1)$, check whether $\Delta T_1 \to \Delta S^0_{8L}||\Delta Y_1$ is a valid difference transition according to the DDT of the used 4 parallel S-boxes. If it is a valid difference transition, compute the corresponding pair of outputs $(R^1_2, R^1_2 \oplus (\Delta S^0_{8L}||\Delta Y_1))$ satisfying this difference transition and compute $(\delta S^0_1, S^0_{5H})$ as follows:

$$\begin{aligned} \delta S^0_1 &= 257^{-1} \cdot (p \boxminus (((R^1_2 \oplus (\Delta S^0_{8L}||\Delta Y_1)) \gg 1) \boxminus (R^1_2 \gg 1) \boxplus 2^{20} \cdot \delta S^0_5)), \\ S^0_{5H} &= (L_2^{-1} \circ S^{-1}(R^1_2)) \gg 16. \end{aligned}$$

When $\delta S^0_1[22:16] \in \{0, \texttt{0x7f}\}$, the tuple $(S^0_{5H}, S^0_{5H} \oplus \Delta S^0_{5H}, \delta S^0_5)$ can pass the test of `VER-H` (see Subsection 4.3) and Con_5 holds, move to Step 3. If these constraints cannot be satisfied, try another pair of outputs until all pairs are traversed.

Step 3: Handle the difference transitions at the 2nd clock. For each $\Delta S^0_{6H} \in \text{SET}_{\Delta S^0_{6H}}$, compute $\Delta T_2 = L_2((\Delta S^0_{6H} \oplus \Delta Y_1) \lll 16)$. For each ΔT_2, traverse all possible $\Delta S^0_{9L}||\Delta Y_2$ where $\Delta S^0_{9L} \in \text{SET}_{\Delta S^0_{9L}}$ and $\Delta Y_2 \in [0, 2^{16}-1]$. For each pair $(\Delta T_2, \Delta S^0_{9L}||\Delta Y_2)$, check whether $\Delta T_2 \to \Delta S^0_{9L}||\Delta Y_2$ is a valid difference transition according to the DDT of the used 4 parallel S-boxes. If it is, compute the corresponding pair of outputs $(R_2^2, R_2^2 \oplus (\Delta S_{9L}^0||\Delta Y_2))$ satisfying this difference transition and compute $(\delta S_2^0, S_{6H}^0)$.

$$\begin{aligned}\delta S_2^0 &= 257^{-1} \cdot (p \boxminus (((R_2^2 \oplus (\Delta S_{9L}^0||\Delta Y_2)) \gg 1) \boxminus (R_2^2 \gg 1) \boxplus 2^{20} \cdot \delta S_6^0)),\\ S_{6H}^0 &= ((L_2^{-1} \circ S^{-1}(R_2^2)) \gg 16) \oplus R_{2L}^1.\end{aligned}$$

When $\delta S_2^0[22:16] \in \{0, \texttt{0x7f}\}$, the tuple $(S_{6H}^0, S_{6H}^0 \oplus \Delta S_{6H}^0, \delta S_6^0)$ can pass the test of `VER-H` and Con_6 holds, move to Step 4. Otherwise, try another pair of outputs until all of them are traversed.

Step 4: Check the validity of S_{7H}^0. For each $\Delta S_{7H}^0 \in \mathrm{SET}_{\Delta S_{7H}^0}$, check the $(\Delta S_{7H}^0 \oplus \Delta Y_2)$-th row of $\mathtt{ARR}_3$. If this row is non-empty, traverse all the stored tuples in this row. For each tuple, get the corresponding value IN_3 and compute S_{7H}^0 as follows:

$$S_{7H}^0 = IN_3 \oplus R_{2L}^2.$$

If the tuple $(S_{7H}^0, S_{7H}^0 \oplus \Delta S_{7H}^0, \delta S_7^0)$ can pass the test of `VER-H-M` and Con_7 holds, a solution to the input difference is found and output the corresponding $(\delta S_1^0, \delta S_2^0, \delta S_3^0)$, (R_2^1, R_2^2, R_2^3), $(S_{5H}^0, \Delta S_{5H}^0)$, $(S_{6H}^0, \Delta S_{6H}^0)$, $(S_{7H}^0, \Delta S_{7H}^0)$, and $(\Delta R_2^1, \Delta R_2^2, \Delta R_2^3) = (\Delta S_{8L}^0||\Delta Y_1, \Delta S_{9L}^0||\Delta Y_2, \Delta S_{10L}^0||\Delta S_{8L}^0)$. Otherwise, consider the next tuple in this row until all of them are exhausted.

In the above procedure, (R_2^1, R_2^2, R_2^3) are almost treated as independent of S_i^0 for $i \in [0, 15]$, which is indeed not the fact. In the following, the IV-correcting technique will be used to deal with such an assumption.

5.6 The IV-Correcting Technique

For an arbitrary solution to (R_2^1, R_2^2, R_2^3) found in `Procedure-DiCancel`, we demonstrate that it is always possible to find an assignment to (K, IV) leading to this solution. The basic idea is to carefully study the update on the two registers in FSM at the first 3 clocks, as specified below:

$$\begin{aligned}R_1^1 &= S \circ L_1(S_{9H}^0||S_{7L}^0),\\ R_2^1 &= S \circ L_2(S_{5H}^0||S_{11L}^0),\\ U &= R_1^1 \boxplus_{32} (S_{12L}^0||S_{10H}^0),\\ R_1^2 &= S \circ L_1(U_L||(R_{2H}^1 \oplus S_{8L}^0)),\\ R_2^2 &= S \circ L_2((R_{2L}^1 \oplus S_{6H}^0)||U_H),\\ V &= R_1^2 \boxplus_{32} (S_{13L}^0||S_{11H}^0),\\ R_2^3 &= S \circ L_2((R_{2L}^2 \oplus S_{7H}^0)||V_H).\end{aligned}$$

Note that $(S_{5H}^0, S_{6H}^0, S_{7H}^0)$ have been determined in `Procedure-DiCancel` and they will not contradict with (R_2^1, R_2^2, R_2^3). Hence, the next task is to determine

$$(S_{9H}^0, S_{7L}^0, S_{11L}^0, S_{12L}^0, S_{10H}^0, S_{8L}^0, S_{13L}^0, S_{11H}^0),$$

which can be finished as follows:

1. Modify S_{11L}^0 with $S_{11L}^0 = (L_2^{-1} \circ S^{-1}(R_2^1))_L$.
2. Compute U_H with $U_H = (L_2^{-1} \circ S^{-1}(R_2^2))_L$.
3. For arbitrarily given (S_{9H}^0, S_{7L}^0), compute R_1^1 with $R_1^1 = S \circ L_1(S_{9H}^0||S_{7L}^0)$.
4. For arbitrarily given S_{10H}^0, compute U_L with $U_L = (R_1^1 + S_{10H}^0) \wedge \texttt{0xffff}$.

5. Modify S^0_{12L} with $S^0_{12L} = ((U_H||U_L) \boxminus_{32} R^1_1)_H$.

6. For arbitrarily given S^0_{8L}, compute R^2_1 with $R^2_1 = S \circ L_1(U_L||(R^1_{2H} \oplus S^0_{8L}))$.

7. Compute V_H with $V_H = (L_2^{-1} \circ S^{-1}(R^3_2))_L$.

8. For arbitrarily given S^0_{11H} with $S^0_{11H}[0] = S^0_{11L}[15]$, compute V_L with $V_L = (R^1_2 + S^0_{11H}) \wedge \texttt{0xffff}$.

9. Modify S^0_{13L} with $S^0_{13L} = ((V_H||V_L) \boxminus_{32} R^2_1)_H$.

In other words, for any assignment to $(S^0_{9H}, S^0_{7L}, S^0_{10H}, S^0_{8L}, S^0_{11H})$, it is always possible to find the corresponding assignment to $(S^0_{11L}, S^0_{12L}, S^0_{13L})$ with time complexity 1 such that they can lead to the given solution to (R^1_2, R^2_2, R^3_2). Note that in both the old and new loading schemes, $(S^0_{11L}, S^0_{12L}, S^0_{13L})$ are all loaded with IV bits. This is why we call it the IV-correcting technique.

Application to ZUC-256: According to the loading scheme for (S^0_5, S^0_6, S^0_7), it is necessary to fix $(IV_0, IV_1, IV_{10}, IV_{17}, IV_{18}, IV_{19})$ and $(K_5[7], K_6[7], K_7[7])$.

Then, as

$$\begin{aligned}
&S^0_{7L} = K_7||IV_2, S^0_{8L} = IV_3||IV_{11}, S^0_{9H} = K_9||1||IV_{21}||IV_{12}[7],\\
&S^0_{10H} = IV_5||1||IV_{22}||K_{10}[7], S^0_{11H} = K_{11}||1||IV_{23}||IV_6[7],\\
&S^0_{11L} = IV_6||IV_{13}, S^0_{12L} = IV_7||IV_{14}, S^0_{13L} = IV_5||IV_8,
\end{aligned}$$

we can say that for arbitrarily given $(K_7[6:0], K_9, K_{10}[7], K_{11})$, it is always possible to find the corresponding assignment to IV such that a given solution to (R^1_2, R^2_2, R^3_2) can be satisfied.

Application to ZUC-256-v2: Similarly, based on the loading scheme for (S^0_5, S^0_6, S^0_7), it is necessary to fix $(K_5, K_6, K_7, K_{21}[7], K_{22}[7])$ and $IV_0[7]$.

Then, since

$$\begin{aligned}
&S^0_{7L} = IV_0||IV_8, S^0_{8L} = IV_1||IV_9, S^0_{9H} = K_9||D_9||IV_2[7],\\
&S^0_{10H} = K_{10}||D_{10}||IV_3[7], S^0_{11H} = K_{11}||D_{11}||IV_4[7],\\
&S^0_{11L} = IV_4||IV_{12}, S^0_{12L} = IV_5||IV_{13}, S^0_{13L} = IV_6||IV_{14},
\end{aligned}$$

we can say that for arbitrarily given (K_9, K_{10}, K_{11}), it is always possible to find the corresponding assignment to IV that can lead to the given solution to (R^1_2, R^2_2, R^3_2).

Feasibility for the key recovery: If the involved key bits are wrongly guessed and we still modify IV bits as above, this assignment to IV indeed cannot lead to the given solution to (R^1_2, R^2_2, R^3_2) and hence the difference in FSM cannot be cancelled after 4 clocks. However, due to the small influence of the value of S^0_{11H} on the modification of S^0_{13L}, i.e., only V_H is constrained by R^3_2 and $V = R^1_2 \boxplus_{32} (S^0_{13L}||S^0_{11H})$, a wrong guess for the key bits loaded into S^0_{11H} may still lead to the targeted (R^1_2, R^2_2, R^3_2). However, for key bits loaded in $(S^0_{7L}, S^0_{9H}, S^0_{10H})$, due to the influence of the L_1, L_2 and S operations, it is almost impossible that they can still lead to the required (R^1_2, R^2_2, R^3_2) if they are wrongly guessed. Hence, it is very likely that we can recover at least $(K_7[6:0], K_9, K_{10}[7])$ and (K_9, K_{10}) for ZUC-256 and ZUC-256-v2, respectively.

Remarks: Let us elaborate more on the IV-correcting technique. To deterministically control the difference transitions in FSM, we need to ensure that (R_2^1, R_2^2, R_2^3) take specific values. However, what the attackers can control is (K, IV). Therefore, we need to convert the conditions on (R_2^1, R_2^2, R_2^3) into conditions on (K, IV). If directly considering all the involved bits to compute (R_2^1, R_2^2, R_2^3) and forcing them to take one specific value, too many bit conditions on (K, IV) are added, which will result in a small number of free bits in (K, IV). However, we know that there are indeed many possible assignments to all these involved bits for a specific (R_2^1, R_2^2, R_2^3). The problem then is how to efficiently compute a valid assignment. This is indeed what the IV-correcting technique achieves. Specifically, attackers can dynamically compute valid assignments to these involved bits, thus increasing the number of free bits in (K, IV).

Moreover, although it is called the IV-correcting technique, we can also interpret it as a guess-and-determine procedure. Specifically, we first guess or fix some bits in (K, IV) and then compute the remaining bits in IV that have not been determined. Our IV-correcting technique can ensure that for any guess, we can always find a valid assignment to the remaining bits in IV such that the full value of (IV, K) can lead to the specific value of (R_2^1, R_2^2, R_2^3).

Hence, in our distinguishing attacks on the initialization phase, to always find the correct input pairs for IV, we need to fix the key bits involved in the IV-correcting procedure and assume they are known to the attackers, which will increase the number of weak key bits by 24 for both ZUC-256 and ZUC-256-v2.

6 Launching the Search

Finally, we are left with the problem of how to choose a proper solution to

$$(\delta S_0^0, \delta S_4^0, \delta S_8^0, \delta S_{10}^0, \delta S_6^0, \delta S_7^0, \delta S_9^0)$$

as the input to `Procedure-DiCancel`.

6.1 Picking $(\delta S_0^0, \delta S_4^0, \delta S_8^0, \delta S_{10}^0, \delta S_6^0, \delta S_7^0, \delta S_9^0)$ for ZUC-256

In our 31-round attack, it is required that

$$\begin{aligned}
2^{21} \cdot \delta S_{10}^0 \boxplus 2^{20} \cdot \delta S_4^0 \boxplus 257 \cdot \delta S_0^0 &= 0, \\
2^{20} \cdot \delta S_8^0 \boxplus 257 \cdot \delta S_4^0 &= 0, \\
2^{20} \cdot \delta S_9^0 \boxplus 257 \cdot \delta S_5^0 &= 0, \\
2^{20} \cdot \delta S_{10}^0 \boxplus 257 \cdot \delta S_6^0 &= 0, \\
(257 \cdot \delta S_7^0)[15:0] &\in \{0, \texttt{0xffff}\}, \\
(2^{15} \cdot (257 \cdot \delta S_7^0) \boxplus 257 \cdot \delta S_8^0)[15:0] &\in \{0, \texttt{0xffff}\}.
\end{aligned}$$

We use a heuristic strategy to pick the solutions to the above system of equations. Specifically, we expect that δS_6^0 can be written as $\delta S_6^0 = 2^i + j$ where $0 < j < 2^{14}$ and $i \in [15, 30]$. This is to keep the simplicity of δS_{6H}^0. Then, for each such δS_6^0, we compute δS_{10}^0 with $\delta S_{10}^0 = (2^{20})^{-1} \cdot (p \boxminus 2^8 \cdot \delta S_6^0)$ and choose the pair $(\delta S_6^0, \delta S_{10}^0)$ satisfying $\delta S_{10H}^0 \in \{0, \texttt{0xffff}\}$ and $H(\delta S_{10}^0) \leq 2$. There are only a few solutions left and we pick the one satisfying that there exists a signed difference ∇S_{10}^0 expanded from δS_{10}^0 whose Hamming weight is 2, i.e., $\mathbb{H}(\nabla S_{10}^0) = 2$.

Then, for the chosen δS_{10}^0, we exhaust all the $2^{25} - 2$ possible values of δS_4^0 satisfying $\delta S_4^0[22:16] \in \{0, \texttt{0x7f}\}$ and compute the corresponding $(\delta S_0^0, \delta S_8^0)$ with

$$\delta S_0^0 = 257^{-1} \cdot (p \boxminus 2^{21} \cdot \delta S_{10}^0 \boxminus 2^{20} \cdot \delta S_4^0),$$

$$\delta S_8^0 \quad = \quad (2^{20})^{-1} \cdot (p \boxminus 257 \cdot S_4^0).$$

When the computed δS_0^0 satisfies $\delta S_0^0[22:16] \in \{0, \texttt{0x7f}\}$, store the corresponding δS_8^0 in a table denoted by `S8Diff`.

Finally, we constrain that $\delta S_{15}^8 = 257 \cdot \delta S_7^0$ satisfies $\delta S_{15L}^8 = 0$. Exhaust all the 2^{15} possible values of δS_{15}^8 and compute $\delta S_7^0 = 257^{-1} \cdot \delta S_{15}^8$ for each δS_{15}^8. If the computed δS_7^0 satisfies $\delta S_{7L}^0 \in \{0, \texttt{0xffff}\}$ and $H(\delta S_7^0) = 1$, exhaust all possible δS_8^0 stored in `S8Diff` and check whether $(2^{15} \cdot \delta S_{15}^8 \boxplus 257 \cdot \delta S_8^0)_L = 0$ and $H((2^{15} \cdot \delta S_{15}^8 \boxplus 257 \cdot \delta S_8^0)) \leq 2$ hold. If all the conditions are satisfied, output the corresponding $(\delta S_7^0, \delta S_8^0, \delta S_4^0, \delta S_{10}^0, \delta S_0^0, \delta S_6^0)$ as the candidate. In our configuration, we choose

$$\delta S_0^0 = \texttt{0x0d80db05}, \delta S_4^0 = \texttt{0x20ff011e}, \delta S_6^0 = \texttt{0x10001fe0},$$
$$\delta S_7^0 = \texttt{0x00020000}, \delta S_8^0 = \texttt{0x7f04fdff}, \delta S_{10}^0 = \texttt{0x7ffffefd}.$$

For such a choice,

$$\delta S_{15}^8 = 257 \cdot \delta S_7^0 = \texttt{0x02020000}, (2^{15} \cdot \delta S_{15}^8 \boxplus 257 \cdot \delta S_8^0) = \texttt{0x04030000}.$$

As already mentioned, $\delta S_{15L}^8 = 0$ does not necessarily imply $\Delta S_{15L}^8 = 0$. For our choice, to make $\Delta S_{15L}^8 \neq 0$, i.e., $((S_{15}^8 \boxplus \delta S_{15}^8) \oplus S_{15}^8)_L \neq 0$, it is required that $S_{15}^8[30:25] = \texttt{0x3f}$ or $(S_{15}^8[30:26] = \texttt{1f}, S_{15}^8[24:17] = \texttt{0xff})$, which holds with probability of about 2^{-6}. This also shows why we choose such modular differences, i.e., $\Delta S_{15L}^0 = 0$ holds with a relatively high probability of about $1 - 2^{-6}$. Similar analysis also applies to `0x04030000`.

Finally, we determine the value of δS_9^0 such that $H(G)$ is small where

$$G = 2^{15} \cdot (2^{15} \cdot \delta S_{15}^8 \boxplus 257 \cdot \delta S_8^0 \boxplus \delta S_{15}^8) \boxplus 257 \cdot \delta S_9^0.$$

In our configuration, we use $\delta S_9^0 = \texttt{0x7ffffdfb}$, which will cause $G = \texttt{0x7ffe0000}$ and $H(G) = 1$.

For the above choice of $(\delta S_0^0, \delta S_4^0, \delta S_8^0, \delta S_{10}^0, \delta S_6^0, \delta S_7^0, \delta S_9^0, \delta S_5^0)$, we first compute the set of XOR differences: $\mathrm{SET}_{\Delta S_{5H}^0}$, $\mathrm{SET}_{\Delta S_{6H}^0}$, $\mathrm{SET}_{\Delta S_{7H}^0}$, $(\mathrm{SET}_{\Delta S_{8H}^0}, \mathrm{SET}_{\Delta S_{8L}^0})$, $\mathrm{SET}_{\Delta S_{9L}^0}$ and $\mathrm{SET}_{\Delta S_{10L}^0}$. Then, `Procedure-DiCancel` is used to determine the remaining unknown variables. It is found that the program outputs many solutions in seconds. One solution is shown in Table 2.

In the search, we made an implicit assumption that

$$((\beta' \boxplus_{32} \gamma) \gg 1) \boxminus ((\beta \boxplus_{32} \gamma) \gg 1) \quad = \quad (\beta' \gg 1) \boxminus (\beta \gg 1), \tag{10}$$

where $\beta', \gamma, \beta \in \mathbb{F}_2^{32}$. For the input difference displayed in Table 2, there are three possible pairs for (β', β), as shown below:

$$(\texttt{0xc99de9d6} \oplus \texttt{0x1e000604} = \texttt{0xd79defd2}, \texttt{0xc99de9d6}),$$
$$(\texttt{0xb7b8cf96} \oplus \texttt{0x03fc0870} = \texttt{0xb444c7e6}, \texttt{0xb7b8cf96}),$$
$$(\texttt{0xfaf5498c} \oplus \texttt{0x017e1e0a} = \texttt{0xfb8b5786}, \texttt{0xfaf5498c}).$$

For each pair (β', β), we then exhaust all the 2^{32} possible values for γ and count the number of γ which can make Equation 10 hold. It is found that for the three possible pairs (β', β), Equation 10 holds with probability of $2^{-0.08}$, $2^{-0.02}$ and $2^{-0.01}$, respectively. Hence, this assumption is reasonable.

Remark. Note that we can choose different signed differences $(\nabla S_0^0, \nabla S_1^1, \nabla S_2^1, \nabla S_3^1, \nabla S_4^1)$ based on $(\delta S_0^0, \delta S_1^0, \delta S_2^0, \delta S_3^0, \delta S_4^1)$.

The reason is that we only need to ensure $\Delta S_i^0[22:16] = 0$ for $i \in [0,4]$. For ∇S_5^0 and ∇S_6^0, as there are strong conditions on $(\Delta S_{5H}^0, \Delta S_{6H}^0)$, $(\nabla S_{5H}^0, \nabla S_{6H}^0)$ have

Table 2: The input difference for the attack on 31-round ZUC-256, where the positions to set constants in the loading scheme are marked in red and the positions to set key bits are marked in blue (69 weak key bits).

i	δS_i^0	∇S_i^0
0	`0x0d80db05`	`=== nn=n n=== ==== nn=n n=nn ==== =n=n`
1	`0x7c00fb01`	`=== =u== ==== ==== nnnn n=nn ==== ==n=`
2	`0x047f38cb`	`=== =n== n=== ==== uu== u=== nn== n=nn`
3	`0x7f8034c3`	`=== ==== u=== ==== ==nn =n== nn== =n==`
4	`0x20ff011e`	`=n= ===n ==== ==== uuuu uuuu ==n= ==u=`
5	`0x20003fc0`	`nu0 0001 111n uuuu uu== ==== =u== ====`
6	`0x10001fe0`	`00n 1010 0101 1101 nuu= ==== ==u= ====`
7	`0x00020000`	`110 1101 0110 1nu0 1=== ==== ==== ====`
8	`0x7f04fdff`	`=== unnn ==== =n=n ===u nnn= ==== ====`
9	`0x7ffffdfb`	`=== ==== ==== ==== ==== ==uu nnnn nn==`
10	`0x7ffffefd`	`=== ==== ==== ==== ==== ===u =unn nnn=`
11	`0x00000000`	`=== ==== ==== ==== ==== ==== ==== ====`
12	`0x00000000`	`=== ==== ==== ==== ==== ==== ==== ====`
13	`0x00000000`	`=== ==== ==== ==== ==== ==== ==== ====`
14	`0x00000000`	`=== ==== ==== ==== ==== ==== ==== ====`
15	`0x00000000`	`=== ==== ==== ==== ==== ==== ==== ====`
$R_2^1 =$ `0xc99de9d6`, $R_2^2 =$ `0xb7b8cf96`, $R_2^3 =$ `0xfaf5498c`		
$\Delta R_2^1 =$ `0x1e000604`, $\Delta R_2^2 =$ `0x03fc0870`, $\Delta R_2^3 =$ `0x017e1e0a`		

to be fixed. Then, only $(\nabla S_5^0[14:0], \nabla S_6^0[14:0])$ can take some other forms. As for $(\nabla S_7^0, \nabla S_8^0, \nabla S_9^0, \nabla S_{10}^0)$, they have to be fixed due to the strong conditions on the XOR differences. For example, the following signed differences are also valid, where the changed parts are marked in green.

$$\begin{aligned}\nabla S_0^0 &= \texttt{=== nn=n n=== ==== nn=n n=nn ==== =nnu},\\ \nabla S_5^0 &= \texttt{nu0 0001 111n uuuu uu== ===u nn== ====}.\end{aligned}$$

6.2 Picking $(\delta S_0^0, \delta S_4^0, \delta S_8^0, \delta S_{10}^0, \delta S_6^0, \delta S_7^0, \delta S_9^0)$ for ZUC-256-v2

Note that some constraints in the 30-round attack are

$$\begin{aligned} 2^{21}\cdot\delta S_{10}^0 \boxplus 2^{20}\cdot\delta S_4^0 \boxplus 257\cdot\delta S_0^0 &= 0,\\ 2^{20}\cdot\delta S_8^0 \boxplus 257\cdot\delta S_4^0 &= 0,\\ 2^{20}\cdot\delta S_9^0 \boxplus 257\cdot\delta S_5^0 &= 0,\\ (2^{20}\cdot\delta S_{10}^0 \boxplus 257\cdot\delta S_6^0)[15:0] &\in \{0, \texttt{0xffff}\},\\ (2^{15}\cdot(2^{20}\cdot\delta S_{10}^0 \boxplus 257\cdot\delta S_6^0) \boxplus 257\cdot\delta S_7^0)[15:0] &\in \{0, \texttt{0xffff}\}.\end{aligned}$$

Since $S_{8H}^0[7:1]$ is constant, for a given δS_8^0, we know that δS_{8H}^0 cannot take too many values. Thus, to increase the possible values of $\Delta S_{10L}^0 || \Delta S_{8H}^0$, we choose a δS_{10}^0 satisfying $\delta S_{10}^0 = \pm 2^i \pm 2^j$ for $i, j \in [0, 14]$ and $i \neq j$, where $\pm$ is addition or subtraction modulo p. In this way, we can expect that the number of all possible ΔS_{10L}^0 is large.

For each such δS_{10}^0, we make a loop for δS_{15}^7 satisfying $\delta S_{15}^7 = \pm 2^i$ for $i \in [16, 29]$ and compute $\delta S_6^0 = 257^{-1}\cdot(\delta S_{15}^7 \boxminus 2^{20}\cdot\delta S_{10}^0)$. We then add a strong condition on δS_6^0, i.e., $\delta S_6^0[30:16] \in \{0, \texttt{0x7fff}\}$. If this condition is satisfied, we next make a loop for the $2^9 - 2$ possible values of δS_7^0 and compute $g = 2^{15}\cdot\delta S_{15}^7 \boxplus 257\cdot\delta S_7^0$. If $g_L \in \{0, \texttt{0xffff}\}$ and $H(g \boxplus \delta S_{15}^7) < 3$, output the candidate $(\delta S_6^0, \delta S_7^0, \delta S_{10}^0)$.

For each candidate found with the above method, we compute the possible values of δS_8^0. Specifically, exhaust all the $2^{25} - 2$ possible values of δS_4^0 and compute

$$\begin{array}{rcl} \delta S_0^0 & = & 257^{-1} \cdot (p \boxminus 2^{21} \cdot \delta S_{10}^0 \boxminus 2^{20} \cdot \delta S_4^0), \\ \delta S_8^0 & = & (2^{20})^{-1} \cdot (p \boxminus 257 \cdot S_4^0) \end{array}$$

for each δS_4^0. If $\delta S_0^0[22:16] \in \{0, \texttt{0x7f}\}$ and $\delta S_8^0[22:16] \in \{0, \texttt{0x7f}\}$, we further compute $f_0 = 2^{20} \cdot \delta S_{10}^0 \boxplus 257 \cdot \delta S_6^0$, $f_1 = 2^{15} \cdot f_0 \boxplus 257 \cdot \delta S_7^0 \boxplus f_0$ and $f_2 = 2^{15} \cdot f_1 \boxplus 257 \cdot \delta S_8^0 \boxplus f_1$. If $H(f_2) < 4$, store the current δS_8^0 in a table denoted by `S8Table`.

Based on the above heuristic strategy, in our configuration, we choose

$$\delta S_0^0 = \texttt{0x017f82fd}, \delta S_4^0 = \texttt{0x6c00200f}, \delta S_6^0 = \texttt{0x0000fe02},$$
$$\delta S_7^0 = \texttt{0x00800000}, \delta S_8^0 = \texttt{0x7e80c13d}, \delta S_{10}^0 = \texttt{0x7fffefef}.$$

In this way,

$$\delta S_{15}^7 = \texttt{0x7ffeffff}, 2^{15} \cdot \delta S_{15}^7 \boxplus 257 \cdot \delta S_7^0 \quad = \quad \texttt{0x800000}.$$

Based on the above choice, we then make a loop for δS_9^0 satisfying $\delta S_9^0 = \pm 2^i$ for $i \in [0, 13]$. For each δS_9^0, compute $\delta S_5^0 = 257^{-1} \cdot (p \boxminus 2^{20} \cdot \delta S_9^0)$ and check whether $\delta S_5^0[22:16] \in \{0, \texttt{0x7f}\}$ holds. It is found that there exist such pairs for $(\delta S_5^0, \delta S_9^0)$. Then, for each valid pair $(\delta S_5^0, \delta S_9^0)$, $(\delta S_0^0, \delta S_4^0, \delta S_8^0, \delta S_{10}^0, \delta S_6^0, \delta S_7^0, \delta S_9^0, \delta S_5^0)$ are fully determined. Similarly, we can use `Procedure-DiCancel` to determine the remaining unknown variables. It is found that solutions are generated in seconds and one solution is shown in Table 3.

Table 3: The input difference for the attack on 30-round ZUC-256-v2, where the positions to set constants in the loading scheme are marked in red and the positions to set key bits are marked in blue (87 weak key bits).

i	δS_i^0	∇S_i^0
0	0x017f82fd	`=== ===n n=== ==== u=== ==nn ==== =u=n`
1	0x037f2f49	`=== =n== u=== ==== uu=u ===u =n== n==n`
2	0x1e00f305	`=n= ==u= ==== ==== nnnn ==nn ==== =n=n`
3	0x12fff85a	`==n ==nn ==== ==== ==== u=== =n=n n=n=`
4	0x6c00200f	`=u= nn== ==== ==== ==n= ==== ===n ====`
5	0x007f00ff	`001 110n u000 0101 uuuu uuuu ==== ===u`
6	0x0000fe02	`001 1101 1101 0001 nnnn nnn= ==== ==n=`
7	0x00800000	`111 0000 n100 0010 1=== ==== ==== ====`
8	0x7e80c13d	`nnn nnn= n=== ==== nn=n uuu= uu== ==uu`
9	0x00000008	`=== ==== ==== ==== ===n uuuu uuuu u===`
10	0x7fffefef	`=== ==== ==== ==== ==un unnn nnnn ====`
11	0x00000000	`=== ==== ==== ==== ==== ==== ==== ====`
12	0x00000000	`=== ==== ==== ==== ==== ==== ==== ====`
13	0x00000000	`=== ==== ==== ==== ==== ==== ==== ====`
14	0x00000000	`=== ==== ==== ==== ==== ==== ==== ====`
15	0x00000000	`=== ==== ==== ==== ==== ==== ==== ====`
$R_2^1 = \texttt{0xa21c991b}$, $R_2^2 = \texttt{0xcf1106f0}$, $R_2^3 = \texttt{0x32f0e1e3}$		
$\Delta R_2^1 = \texttt{0xdec311a0}$, $\Delta R_2^2 = \texttt{0x1ff810de}$, $\Delta R_2^3 = \texttt{0x3ff0fd01}$		

Similarly, it is necessary to take Equation 10 into account. The three pairs for (β', β) are

$$(\texttt{0xa21c991b} \oplus \texttt{0xdec311a0} = \texttt{0x7cdf88bb}, \texttt{0xa21c991b}),$$
$$(\texttt{0xcf1106f0} \oplus \texttt{0x1ff810de} = \texttt{0xd0e9162e}, \texttt{0xcf1106f0}),$$

$$(\texttt{0x32f0e1e3} \oplus \texttt{0x3ff0fd01} = \texttt{0x0d001ce2}, \texttt{0x32f0e1e3}).$$

For these three pairs, Equation 10 holds with probability of $2^{-0.23}$, $2^{-0.01}$ and 2^{-1}, respectively.

7 Searching for Biased Linear Relations

With the discovered input differences, the next step is to search for the best biased linear relation via simulations as in [BM20]. Suppose we aim at an attack on $t+15$ initialization rounds.

The simulations are simple. First, construct four tables $\texttt{TAB}_0$, $\texttt{TAB}_1$, $\texttt{TAB}_2$ and $\texttt{TAB}_3$, which are of size 2^{15}, 2^{16}, 2^{16} and 2^{15}, respectively. The four tables are all initialized by zero. Then, uniformly at random choose N pairs of (K, IV) and (K', IV') satisfying the signed differences ∇S_i^0 for $i \in [0, 15]$. For each pair, use the IV-correcting technique to correct IV such that the fixed (R_2^1, R_2^2, R_2^3) can be satisfied and modify IV' accordingly based on the signed differences, i.e., (IV, IV') has to satisfy certain signed differences. Next, compute $\delta S_{15}^t[14:0]$, $\delta S_{15}^t[22:7]$, $\delta S_{15}^t[30:15]$ and $\delta S_{15}^t[6:0]||\delta S_{15}^t[30:23]$ for this pair and increase $\texttt{TAB}_0[\delta S_{15}^t[14:0]]$, $\texttt{TAB}_1[\delta S_{15}^t[22:7]]$, $\texttt{TAB}_2[\delta S_{15}^t[30:15]]$ and $\texttt{TAB}_3[\delta S_{15}^t[6:0]||\delta S_{15}^t[30:23]]$ by 1, respectively.

After N samples are all used, for the distribution table $\texttt{TAB}_i$, we apply Walsh-Hadamard-Transform (WHT) to it and obtain the corresponding spectrum. Then, loop through the spectrum and find the nonzero index where the absolute value is the largest. Denote the spectrum at index j by $\mathcal{W}_j$. Then, the absolute value of the bias for the linear mask j can be computed as $|\mathcal{W}_j|/2\mathcal{W}_0$. After applying WHT to the four tables, we pick the linear mask whose bias is the largest and denote it by ϵ. To avoid the false-positive results, similar to [BM20], we require that

$$N \geq 2^4 \times \frac{1}{\epsilon^2}. \tag{11}$$

In other words, if Equation 11 cannot hold, we need to increase N and repeat the same procedure until we find a reliable biased linear relation, i.e., Equation 11 holds.

The biased linear relation for 31-round ZUC-256: Based on the input difference in Table 2, we found the following best biased linear relation with about $2^{36.7}$ samples[2].

$$Pr[\delta S_0^{31}[6] = 0] \approx 0.5 + 2^{-13.5}.$$

Hence, the time and data complexity[3] of the attack on 31-round ZUC-256 are both estimated as $2^{1+27+1} = 2^{29}$ as each pair corresponds to 2 inputs.

The attack procedure: The attack procedure is essentially the same as in the simulation phase. Specifically, at the first step, we randomly fix a pair of weak keys satisfying the conditions imposed by the signed input differences. To make the IV-correcting technique work, apart from these conditions on key bits, we also assume the attackers know additional 24 key bits. In this way, the number of weak keys is reduced by a factor of 2^{24}, though we have 2^{24} different sets of such weak keys.

[2]In our simulations, we use mt19937_64 in C++ to generate a 64-bit random value and then assign this 64-bit value to the key bits and IV bits. Alexander Maximov has verified our results by using ZUC-256 as the random source. Our implementation can be found at https://github.com/LFKOKAMI/zuc-256.git.

[3]Notice that for a uniformly at random chosen element x in $GF(2^{31}-1)$, $Pr[x[i] = 0] \approx 0.5 + \frac{1}{2^{31}-1}$. As $\frac{1}{2^{31}-1}$ is much smaller than $2^{-13.5}$, the found biased linear relation can be used to construct a distinguisher. A more accurate estimation of the complexity to distinguish the two distributions will be almost the same with our way.

After fixing a pair of weak keys, we randomly generate about 2^{28} IV pairs also satisfying the signed difference. For each IV pair, we correct one with the IV-correcting technique such that the conditions on (R_2^1, R_2^2, R_2^3) can hold and modify the other according to the condition on their XOR difference because the pair needs to satisfy the input difference. Note that the IV-correcting technique is just a guess-and-determine procedure and we can interpret it in another way. Specifically, for each generated IV pair, instead of specifying all the IV bits, we can specify a small number of IV bits and determine the remaining unspecified IV bits with the IV-correcting technique.

With the IV-correcting technique, we obtain a pair of (K, IV) which satisfies all the conditions specified in Table 2 and hence the difference transitions in FSM are under control. For all the input pairs, we then compute $\delta S_0^{31}[6]$ and count the number of times when it takes 0. After processing all the 2^{28} IV pairs, supposing there are N' IV pairs such that $\delta S_0^{31}[6] = 0$, we expect that $N'/2^{28} \approx 0.5 + 2^{-13.5}$. The whole procedure is indeed how the common linear attack is performed. The only difference is that the biased linear relation holds only for a set of constrained input pairs.

The biased linear relation for 30-round ZUC-256-v2: Based on the input difference in Table 3, the following biased linear relation is found with about 2^{47} samples:

$$Pr[\delta S_0^{30}[29] = 0] \approx 0.5 + 2^{-18.9}.$$

Similarly, the time and data complexity of the attack on 30-round ZUC-256-v2 are both estimated as $2^{37.8+2} = 2^{39.8}$.

7.1 Key-recovery Attacks Using the First Keystream Word

As already mentioned in the IV-correcting technique, it is possible to recover at least 16 key bits for ZUC-256 and ZUC-256-v2, respectively, if a proper distinguisher can be constructed. Hence, we use the biased linear relation in the XOR difference ΔZ of the first 32-bit keystream word to construct such a distinguisher. The way to detect biased linear relations follows a similar idea used in the distinguishing attack.

Recovering 16 key bits for 15-round ZUC-256 in the related-key setting: With the input difference displayed in Table 2 and about 2^{32} samples, we found the following biased linear relation in ΔZ when the number of initialization rounds is reduced to 15:

$$Pr[\Delta Z[7] = 0] \approx 0.5 + 2^{-9.5}.$$

Our key-recovery attack naturally works in the weak-key setting due to the constraints of the signed differences. First, we generate many IV pairs (IV, IV') satisfying the signed differences. Then, guess $(K_7[6:0], K_9, K_{10}[7], K_{11})$ and correct the IV pair using the IV-correcting technique. If the key is correctly guessed, the above biased linear relation will hold. However, if the key is wrongly guessed, the above linear relation will behave randomly. As explained before, we can at least expect to recover $(K_7[6:0], K_9, K_{10}[7])$. According to the experiments discussed below, to increase the success rate, the time and data complexity will be estimated as around $2^{3+19+24+1} = 2^{47}$.

Remark: Note that the 30- and 31-round distinguishing attacks can be converted into partial key-recovery attacks if the attacker has access to S_0 after these many rounds as well, whose time and data complexity will be increased by a factor[4] of about 2^{24}, respectively. Specifically, the attackers guess the involved 24 key bits and perform the IV-correcting technique for each IV pair under each guess. For the correct guess of these 24 key bits, the

[4]We may say the factor is 2^{26} to increase the success rate.

used IV pairs will lead to $Pr[\delta S_0^{31}[6] = 0] \approx 0.5 + 2^{-13.5}$ and $Pr[\delta S_0^{30}[29] = 0] \approx 0.5 + 2^{-18.9}$ for 31-round ZUC-256 and 30-round ZUC-256-v2, respectively. For the wrong guess, these biased linear relations will behave randomly. As explained before, based on this procedure, we can expect to recover at least 16 key bits for both ZUC-256 and ZUC-256-v2.

Recovering 16 key bits for 14-round ZUC-256-v2 in the related-key setting: With the input difference displayed in Table 3 and about 2^{36} samples, we found the following biased linear relation in ΔZ when the number of initialization rounds is reduced to 14.

$$Pr[\Delta Z[30] = 0] \approx 0.5 - 2^{-14.5}.$$

Based on a similar procedure, we can recover at least 16 key bits (K_9, K_{10}). To increase the success rate, both the time and data complexity are estimated as around $2^{3+29+24+1} = 2^{58}$.

Experiments: To support our claim that at least 16 key bits can be recovered for ZUC-256 and ZUC-256-v2, respectively, we performed experiments for the key-recovery attacks on 14-round ZUC-256 and 13-round ZUC-256-v2. In such attacks, with the input differences in Table 2 and Table 3, respectively, the best biased linear relations have much larger biases as we even do not need to approximate the update in FSM. Specifically, in the key-recovery attack on 14-round ZUC-256, there exists a linear relation with a bias of $2^{-3.2}$, i.e., $Pr[\Delta Z[7] = 0] \approx 0.5 + 2^{-3.2}$. In the key-recovery attack on 13-round attack, there exists a linear relation with a bias of $2^{-3.5}$, i.e., $Pr[\Delta Z[14] = 0] \approx 0.5 + 2^{-3.5}$. Hence, we can repeat the experiments several times to verify our claims due to the low time complexity of the attacks.

In the experiment for the attack on 14-round ZUC-256, for each guess of the key bits, we use 2^{10} random samples of IV pairs and check whether $Pr[\Delta Z[7] = 0] \approx 0.5 + 2^{-3.2}$ holds. It is found that the 16 key bits $(K_7[6:0], K_9, K_{10}[7])$ can always be correctly recovered for ZUC-256, while there are still many possible candidates for K_{11}.

In the experiment for the attack on 13-round ZUC-256-v2, for each guess of the key bits, we again use 2^{10} random samples of IV pairs and check whether $Pr[\Delta Z[14] = 0] \approx 0.5 + 2^{-3.5}$ holds. It is found that the 16 key bits (K_9, K_{10}) are always correctly recovered, while there are still many possible values for K_{11}.

Therefore, our claim to recover at least 16 key bits for both, 15-round ZUC-256 and 14-round ZUC-256-v2 is correct.

8 Conclusion

While the round function of ZUC-256 is well designed to resist differential attacks with simple input differences, by carefully controlling the interactions between all the operations in the round function, we report for the first time that complex input differences can be found and utilized to mount practical attacks on 31 and 30 initialization rounds of ZUC-256 and ZUC-256-v2, which reduce their security margins against this kind of distinguishing attacks to only 2 and 3 rounds, respectively. Finding such complex input differences is challenging as it is essential to solve a system of complex equations. By using the signed difference to build the bridge between the modular difference and the XOR difference and developing advanced guess-and-determine techniques, we finally overcome this obstacle and succeed in finding solutions to such equations. A notable feature of our attacks is to control one memory register in FSM for 4 clocks. It is unclear whether better ways can be found to further control the difference transitions in FSM.

Although our distinguishing attacks work in a very strong attack scenario, the scenario has been taken into account by the ZUC team and SAGE and therefore we believe this work is still meaningful. We view our main contribution as the introduction of modular difference and signed difference in the context of ZUC-256 to significantly strengthen the analysis of the interactions between LFSR, BR and FSM of the round function.

Acknowledgments

We thank Alexander Maximov for reviewing the preliminary version of this paper and providing many important comments to improve its quality. Also, we thank him for performing intensive simulations to verify the 30-round attack on ZUC-256-v2 with his efficient implementation of ZUC-256-v2. We also thank the reviewers of ToSC 2022 Issue 2 and Issue 3 for putting lots of efforts to further improve this paper. Takanori Isobe is supported by JST, PRESTO Grant Number JPMJPR2031 and Grant-in-Aid for Scientific Research (B)(KAKENHI 19H02141). This research was in part conducted under a contract of "Research and development on new generation cryptography for secure wireless communication service" among "Research and Development for Expansion of Radio Wave Resources (JPJ000254)", which was supported by the Ministry of Internal Affairs and Communications, Japan. Gaoli Wang is supported by National Natural Science Foundation of China (No. 62072181), NSFC-ISF Joint Scientific Research Program (No. 61961146004), Shanghai Trusted Industry Internet Software Collaborative Innovation Center.

References

[AFK$^+$08] Jean-Philippe Aumasson, Simon Fischer, Shahram Khazaei, Willi Meier, and Christian Rechberger. New Features of Latin Dances: Analysis of Salsa, ChaCha, and Rumba. In Kaisa Nyberg, editor, *Fast Software Encryption, 15th International Workshop, FSE 2008, Lausanne, Switzerland, February 10-13, 2008, Revised Selected Papers*, volume 5086 of *Lecture Notes in Computer Science*, pages 470–488. Springer, 2008.

[BM20] Steve Babbage and Alexander Maximov. Differential Analysis of the ZUC-256 Initialisation. Cryptology ePrint Archive, Report 2020/1215, 2020. https://eprint.iacr.org/2020/1215.

[BS90] Eli Biham and Adi Shamir. Differential Cryptanalysis of DES-like Cryptosystems. In Alfred Menezes and Scott A. Vanstone, editors, *Advances in Cryptology - CRYPTO '90, 10th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1990, Proceedings*, volume 537 of *Lecture Notes in Computer Science*, pages 2–21. Springer, 1990.

[BVC16] Alex Biryukov, Vesselin Velichkov, and Yann Le Corre. Automatic Search for the Best Trails in ARX: Application to Block Cipher Speck. In Thomas Peyrin, editor, *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, volume 9783 of *Lecture Notes in Computer Science*, pages 289–310. Springer, 2016.

[CR06] Christophe De Cannière and Christian Rechberger. Finding SHA-1 Characteristics: General Results and Applications. In Xuejia Lai and Kefei Chen, editors, *Advances in Cryptology - ASIACRYPT 2006, 12th International Conference on the Theory and Application of Cryptology and Information Security, Shanghai, China, December 3-7, 2006, Proceedings*, volume 4284 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2006.

[ETS11] ETSI SAGE. *Specification of the 3GPP Confidentiality and Integrity Algorithms 128-EEA3 & 128-EIA3. Document 4: Design and Evaluation Report*, 2011. https://www.gsma.com/aboutus/wp-content/uploads/2014/12/EEA3_EIA3_Design_Evaluation_v2_0.pdf.

[ETS21] ETSI SAGE. LS (21) 04: Further discussion on ZUC-256, January 2021. https://www.3gpp.org/ftp/tsg_sa/WG3_Security/TSGS3_104e/Inbox/Drafts/S3-212676%20with%20reference2%20and%204.zip *(accessed on 2021-08-20)*.

[Leu13] Gaëtan Leurent. Construction of Differential Characteristics in ARX Designs Application to Skein. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 241–258. Springer, 2013.

[LP19] Gaëtan Leurent and Thomas Peyrin. From collisions to chosen-prefix collisions application to full SHA-1. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part III*, volume 11478 of *Lecture Notes in Computer Science*, pages 527–555. Springer, 2019.

[LP20] Gaëtan Leurent and Thomas Peyrin. SHA-1 is a shambles: First chosen-prefix collision on SHA-1 and application to the PGP web of trust. In Srdjan Capkun and Franziska Roesner, editors, *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*, pages 1839–1856. USENIX Association, 2020.

[MNS11] Florian Mendel, Tomislav Nad, and Martin Schläffer. Finding SHA-2 Characteristics: Searching through a Minefield of Contradictions. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings*, volume 7073 of *Lecture Notes in Computer Science*, pages 288–307. Springer, 2011.

[SBK+17] Marc Stevens, Elie Bursztein, Pierre Karpman, Ange Albertini, and Yarik Markov. The first collision for full SHA-1. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I*, volume 10401 of *Lecture Notes in Computer Science*, pages 570–596. Springer, 2017.

[SSA+09] Marc Stevens, Alexander Sotirov, Jacob Appelbaum, Arjen K. Lenstra, David Molnar, Dag Arne Osvik, and Benne de Weger. Short chosen-prefix collisions for MD5 and the creation of a rogue CA certificate. In Shai Halevi, editor, *Advances in Cryptology - CRYPTO 2009, 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings*, volume 5677 of *Lecture Notes in Computer Science*, pages 55–69. Springer, 2009.

[Tea21] ZUC Design Team. An addendum to the zuc-256 stream cipher. Cryptology ePrint Archive, Report 2021/1439, 2021. https://ia.cr/2021/1439.

[The18] The ZUC Team. *The ZUC-256 Stream Cipher*, 2018. http://www.is.cas.cn/ztzl2016/zouchongzhi/201801/W020180126529970733243.pdf.

[WLF+05] Xiaoyun Wang, Xuejia Lai, Dengguo Feng, Hui Chen, and Xiuyuan Yu. Cryptanalysis of the Hash Functions MD4 and RIPEMD. In Ronald Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, volume 3494 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2005.

[WY05] Xiaoyun Wang and Hongbo Yu. How to Break MD5 and Other Hash Functions. In Ronald Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, volume 3494 of *Lecture Notes in Computer Science*, pages 19–35. Springer, 2005.

[WYY05] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding Collisions in the Full SHA-1. In Victor Shoup, editor, *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, volume 3621 of *Lecture Notes in Computer Science*, pages 17–36. Springer, 2005.

[YJM20] Jing Yang, Thomas Johansson, and Alexander Maximov. Spectral analysis of ZUC-256. *IACR Trans. Symmetric Cryptol.*, 2020(1):266–288, 2020.

A Some Proofs

A.1 Proving Fact 2

If we restrict that $\nabla a[i] \in \{\mathtt{n}, \mathtt{=}\}$ $(0 \le i \le 30)$, the signed difference is uniquely determined for a given modular difference δa, as specified below:

$$\nabla a[i] = \begin{cases} \mathtt{n} & (\delta a[i] = 1) \\ \mathtt{=} & (\delta a[i] = 0) \end{cases}$$

Proof. Suppose there are two different signed differences ∇a_0 and ∇a_1 satisfying the restrictions $\nabla a_0[i] \in \{\mathtt{n}, \mathtt{=}\}$ and $\nabla a_1[i] \in \{\mathtt{n}, \mathtt{=}\}$ for $(0 \le i \le 30)$, while they both correspond to the same modular difference δa. Denote the modular difference of ∇a_0 and ∇a_1 by δa_0 and δa_1, respectively. Note that

$$\delta a_0 = \sum_{i=0}^{30} \mu_i^0 \cdot 2^i, \quad \delta a_1 = \sum_{i=0}^{30} \mu_i^1 \cdot 2^i,$$

where

$$u_i^j = \begin{cases} 1 & (\nabla a_j[i] = \mathtt{n}) \\ 0 & (\nabla a_j[i] = \mathtt{=}) \end{cases}$$

As $\mu_i^0, \mu_i^1 \in \mathbb{F}_2$ in this case, $\delta a_0 = \delta a_1$ is equivalent to $\mu_i^0 = \mu_i^1$ for $0 \le i \le 30$.

When ∇a_0 and ∇a_1 are different, there must exist an index x such that $(\nabla a_0[x] = \mathtt{=}, \nabla a_1[x] = \mathtt{n})$ or $(\nabla a_0[x] = \mathtt{n}, \nabla a_1[x] = \mathtt{=})$. For both cases, there must be $\mu_x^0 \ne \mu_x^1$, thus contradicting with the assumption that $\delta a_0 = \delta a_1$.

Moreover, if ∇a satisfies

$$\nabla a[i] = \begin{cases} \mathtt{n} & (\delta a[i] = 1) \\ \mathtt{=} & (\delta a[i] = 0) \end{cases}$$

for $0 \le i \le 30$, it must correspond to δa according to Fact 1. Hence, Fact 2 is proved. □

A.2 Proving Proposition 1

Proof. `Necessity:`

When $\Delta a[j:i]=0$, there must be $\nabla a[x] = \texttt{=}$ for $x \in [i,j]$. Notice that

$$\delta a = \sum_{g=0}^{30} \mu_g \cdot 2^g,$$

where $\mu_g = 0$ for $\nabla a[g] = \texttt{=}$, $\mu_g = 1$ for $\nabla a[g] = \texttt{n}$ and $\mu_g = -1$ for $\nabla a[g] = \texttt{u}$. Therefore, when $\nabla a[x] = \texttt{=}$ for $(i \leq x \leq j)$, we have

$$\delta a = \sum_{g=0}^{i-1} \mu_g \cdot 2^g \boxplus \sum_{g=j+1}^{30} \mu_g \cdot 2^g.$$

Let

$$\nu_0 = \sum_{g=0}^{i-1} \mu_g \cdot 2^g, \quad \nu_1 = \sum_{g=j+1}^{30} \mu_g \cdot 2^g.$$

As the addition is defined over $GF(p)$, we have that

$$\nu_0 \in \{s|0 \leq s < 2^i\} \cup \{s|p \boxminus 2^i < s \leq p \boxminus 1, s[t]=1, i \leq t \leq 30\}.$$

Similarly, we have

$$\nu_1 \in \{0\} \cup \{s|2^{j+1} \leq s \leq 2^{31} \boxminus 2^{j+1}, s[t]=0, 0 \leq t \leq j\}$$

or

$$\nu_1 \in \{s|2^{j+1} \boxminus 1 \leq s \leq p \boxminus 2^{j+1}, s[t]=1, 0 \leq t \leq j\}$$

Therefore, there are 6 possible combinations of (ν_0, ν_1).

When $\nu_1 = 0$, it is trivial to prove that $(v_0 \boxplus v_1)[j:i] = 0$ or $(v_0 \boxplus v_1)[j:i] = 2^{j-i+1}-1$. Then, we are only left with 4 combinations.

When $\nu_1 \in \{s|2^{j+1} \leq s \leq 2^{31} \boxminus 2^{j+1}, s[t]=0, 0 \leq t \leq j\}$ and $\nu_0 \in \{s|0 \leq s < 2^i\}$, we have $(v_0 \boxplus v_1)[j:i] = 0$.

When $\nu_1 \in \{s|2^{j+1} \leq s \leq 2^{31} \boxminus 2^{j+1}, s[t]=0, 0 \leq t \leq j\}$ and $\nu_0 \in \{s|p \boxminus 2^i < s \leq p \boxminus 1, s[t]=1, i \leq t \leq 30\}$, we have $\nu_1[j:0]=0$ and $\nu_0[j:i] = 2^{j-i+1}-1$. As $2^{31} = 1 \bmod (p)$, when $\nu_1 + \nu_0 > 2^{31}$, it can be derived that $(v_0 \boxplus v_1)[j:i] \in \{0, 2^{j-i+1}-1\}$. When $\nu_1 + \nu_0 < 2^{31}$, we have $(v_0 \boxplus v_1)[j:i] = 2^{j-i+1}-1$.

When $\nu_1 \in \{s|2^{j+1} \boxminus 1 \leq s \leq p \boxminus 2^{j+1}, s[t]=1, 0 \leq t \leq j\}$ and $\nu_0 \in \{s|0 \leq s < 2^i\}$, we have $(v_0 \boxplus v_1)[j:i] = 2^{j-i+1}-1$.

When $\nu_1 \in \{s|2^{j+1} \boxminus 1 \leq s \leq p \boxminus 2^{j+1}, s[t]=1, 0 \leq t \leq j\}$ and $\nu_0 \in \{s|p \boxminus 2^i < s \leq p \boxminus 1, s[t]=1, i \leq t \leq 30\}$, we have $\nu_1[j:0] = 2^{j+1}-1$ and $\nu_0[j:i] = 2^{j-i+1}-1$. Similarly, it can be derived that $(v_0 \boxplus v_1)[j:i] = 0$. This completes the proof for necessity.

`Sufficiency:`

According to Fact 2, given an arbitrary modular difference δa, there always exists a corresponding signed difference ∇a such that

$$\nabla a[i] = \begin{cases} \texttt{n} & (\delta a[i] = 1) \\ \texttt{=} & (\delta a[i] = 0) \end{cases}$$

When $\delta a[j:i] = 0$, there always exists such a ∇a that $\nabla a[t] = \texttt{=}$ $(i \leq t \leq j)$, which is equivalent to that there exists a pair (a, a') satisfying $\Delta a[j:i] = 0$.

When $\delta a[j:i] = 2^{j-i+1}-1$, there must be $(p \boxminus \delta a)[j:i] = 0$. Based on the above proof, we can always find a pair (b, b') satisfying $\Delta b[j:i] = 0$ and $b' \boxminus b = p \boxminus \delta a \Leftrightarrow b' \boxminus p = b \boxminus \delta a \Leftrightarrow b' = b \boxminus \delta a$. In other words, we can always find a pair $(a, a') = (b', b)$ such that $a' \boxminus a = \delta a$ and $\Delta a[j:i] = \Delta b[j:i] = 0$, which completes the proof. □

A.3 Proving the Correctness of `ENU-H`

Proof. Let $x = a+\delta a$ where $a, \delta a \in [0, p)$ and $x \in [0, 2^{32}-1)$. We discuss three possible cases for $\delta a[14:0]$ since the addition is modulo p, which are $\delta a[14:0] = 0$, $\delta a[14:0] = $ `0x7fff` and $\delta a[14:0] \notin \{0, \texttt{0x7fff}\}$. It should be emphasized that $a \boxplus \delta a = x$ when $x < p$ and $a \boxplus \delta a = x - 2^{31} + 1$ when $x \geq p$ since $2^{31} - 1 \leq x < 2^{32} - 2 \Rightarrow 0 \leq x - 2^{31} + 1 < 2^{31} - 1$.

`Case-1`: When $\delta a[14:0] = 0$, there will always be $x[31:15] = a_H + \delta a_H$. When $a[14:0] \neq$ `0x7fff`, there is always $a'_H = x[30:15]$. In other words, if $a[14:0] \neq$ `0x7fff` holds, whatever $a[14:0]$ is, it will correspond to the same set of possible pairs (a'_H, a_H) satisfying $a' = a \boxplus \delta a$. Therefore, by fixing $a[14:0] = 0$ and traversing a_H, we can obtain all the possible pairs (a'_H, a_H) for the case $a[14:0] \neq$ `0x7fff`. After fixing $a[14:0] =$ `0x7fff` and traversing a_H, all possible values of $a[14:0]$ are taken into account and the generated pairs (a'_H, a_H) are all the possible pairs satisfying $a' = a \boxplus \delta a$ and we do not miss any of them.

`Case-2`: For $\delta a[14:0] =$ `0x7fff`, when $a[14:0] \neq 0$, there will be $\delta a[14:0] + a[14:0] \geq 2^{15}$. Hence, there will always be $x[31:15] = a_H + \delta a_H + 1$ and $x[14:0] \neq$ `0x7fff`. Therefore, there must be $a'_H = x[30:15]$. In other words, if $a[14:0] \neq 0$ holds, whatever $a[14:0]$ is, it will correspond to the same set of possible pairs (a'_H, a_H) satisfying $a' = a \boxplus \delta a$. Therefore, by fixing $a[14:0] =$ `0x7fff` and traversing a_H, we can obtain all the possible pairs (a'_H, a_H) for the case $a[14:0] \neq 0$. After a_H is also traversed for $a[14:0] = 0$, all possible values of $a[14:0]$ are considered and the generated pairs (a'_H, a_H) are all the possible pairs.

`Case-3`: For $\delta a[14:0] \notin \{0, \texttt{0x7fff}\}$, we classify $a[14:0]$ into three categories, which are $a[14:0] + \delta a[14:0] \geq 2^{15}$, $a[14:0] + \delta a[14:0] <$ `0x7fff` and $a[14:0] + \delta a[14:0] =$ `0x7fff`.

`Case-3-1`: When $a[14:0] + \delta a[14:0] \geq 2^{15}$, there is always $x[31:15] = a_H + \delta a_H + 1$ and $x[14:0] \neq$ `0x7fff`. Due to $x[14:0] \neq$ `0x7fff`, whatever $x[31]$ takes, there is always $a'_H = x[30:15]$. By fixing $a[14:0] =$ `0x7fff`, there must be $a[14:0] + \delta a[14:0] \geq 2^{15}$. Hence, for all $a[14:0]$ satisfying $a[14:0] + \delta a[14:0] \geq 2^{15}$, we obtain all possible pairs (a'_H, a_H) by fixing $a[14:0] =$ `0x7fff` and traversing a_H. Denote this set of all possible (a'_H, a_H) by $\mathrm{SET}_{\texttt{3-1}}$.

`Case-3-2`: When $a[14:0] + \delta a[14:0] <$ `0x7fff`, there is always $x[31:15] = a_H + \delta a_H$. Whatever $x[31]$ is, there is always $a'_H = x[30:15]$ due to $x[14:0] \neq$ `0x7fff`. By fixing $a[14:0] = 0$, there must be $a[14:0] + \delta a[14:0] <$ `0x7fff`. In other words, for all $a[14:0]$ satisfying $a[14:0] + \delta a[14:0] <$ `0x7fff`, we obtain all possible pairs (a'_H, a_H) by fixing $a[14:0] = 0$ and traversing a_H. Denote this set of all possible (a'_H, a_H) by $\mathrm{SET}_{\texttt{3-2}}$.

`Case-3-3`: When $a[14:0] + \delta a[14:0] =$ `0x7fff`, $x[31:15] = a_H + \delta a_H$ still always holds. If $x[31] = 0$, we will have $a'_H = x[30:15]$. For this case, when traversing a_H, the generated pairs (a'_H, a_H) satisfying $x[31] = 0$ is a subset of $\mathrm{SET}_{\texttt{3-2}}$. If $x[31] = 1$, we will have $a'_H = x[30:15] + 1$. For this case, when traversing a_H, the generated pairs (a'_H, a_H) satisfying $x[31] = 1$ is a subset of $\mathrm{SET}_{\texttt{3-1}}$. Until now, all possible values of $a[14:0]$ have been taken into account. As the generated set of possible pairs (a'_H, a_H) in `Case-3-3` must be a subset of $\mathrm{SET}_{\texttt{3-1}} \cup \mathrm{SET}_{\texttt{3-2}}$, it implies that traversing a_H for $a[14:0] \in \{\texttt{0,0x7fff}\}$ is sufficient to generate all possible pairs (a'_H, a_H) satisfying $a' = a \boxplus \delta a$, which completes the proof.

□

B Revisiting Babbage-Maximov's Attacks [BM20]

A major difference between ZUC-256 and ZUC-128 is that there are more state bits loaded by key bits. This naturally provides more degrees of freedom to choose the injected differences for an attacker, which is indeed exploited in [BM20].

There are two kinds of attacks described in [BM20]. The first one is to inject differences in up to 5 key bits, while the second one is to inject differences in IV bits in an advanced way.

B.1 Injecting Differences in Key Bits

To find the optimal key differences, Babbage and Maximov treated ZUC-256 as a blackbox. Specifically, they first randomly choose up to 5 key bits to inject differences. Then, for a fixed key difference, randomly generate sufficiently many (K, IV) pairs satisfying the fixed key difference and collect the corresponding XOR difference ΔS_{15}^{t} if the target is $t+15$ initialization rounds as $\Delta S_0^{t+15} = \Delta S_{15}^{t}$. Supposing there are N samples, i.e., N random pairs of (K, IV), they can collect a distribution table of ΔS_{15}^{t} from these N samples. Specifically, in this distribution table, the number of times that ΔS_{15}^{t} takes the value i for each $i \in \mathbb{F}_2^{31}$ will be recorded. After collecting the distribution table, they will apply the Walsh-Hadamard Transform (WHT) to it in order to search for the boolean linear relation in terms of the 31 bits of ΔS_{15}^{t} with the highest bias. As the table is of size 2^{31}, i.e., ΔS_{15}^{t} is a 31-bit value, finding the best linear relation (linear mask) for ΔS_{15}^{t} will take time $2^{31} \times 31$ by applying WHT to the distribution table. After obtaining the highest bias denoted by ϵ by applying WHT, i.e., the best linear relation holds with probability $0.5 + \epsilon$ in these N samples, it is further required to check whether $N \geq \frac{2^4}{\epsilon^2}$ holds to rule out the false-positive results. Finally, they will select the injected difference leading to the highest bias as the final key difference.

The input difference used in [BM20] is as follows:

$$\Delta S_2^0 = \texttt{0x01000000}, \ \Delta S_6^0 = \texttt{0x00001010}.$$

For such an input difference, the best biased linear relation in terms of ΔS_0^{28} is

$$Pr[\Delta S_0^{28}[9] \oplus \Delta S_0^{28}[10] = 1] \approx 0.5 - 2^{-10.46},$$

which indicates that using about 2^{25} samples, it is possible to construct a distinguisher for 28 (out of 33) rounds ZUC-256. Extending this method to more rounds becomes infeasible because it requires an impractical number of samples. Notice that the biased linear relation is fully derived via experiments.

B.2 Injecting Differences in IV Bits

In addition to the above attack strategy, the authors also explored how many rounds such a distinguisher could reach by injecting differences in IV bits. To achieve this, they observed that it was possible to control the difference transitions in FSM for the first 3 clocks. Specifically, they will inject differences at $(S_{5H}^0, S_{6H}^0, S_{7H}^0, S_{8L}^0, S_{9L}^0)$ due to the restriction that the differences can only be injected in IV bits. To find a solution to the input difference, they constructed the following equations:

$$\begin{aligned}
&\Delta R_2^1 = S \circ L_2(S_{5H}^0 || S_{11L}^0) \oplus S \circ L_2((S_{5H}^0 \oplus \Delta S_{5H}^0) || S_{11L}^0),\\
&(R_2^1 \gg 1) \boxplus (S_{5H}^0 \ll 4) = ((R_2^1 \oplus \Delta R_2^1) \gg 1) \boxplus ((S_{5H}^0 \oplus \Delta S_{5H}^0) \ll 4),\\
&\Delta S_{8L}^0 = \Delta R_{2H}^1,\\
&R_1^1 = S \circ L_1(S_{9H}^0 || S_{7L}^0),
\end{aligned}$$

$$\begin{aligned}
&y = (R_1^1 \boxplus_{32} (S_{12L}^0 || S_{10H}^0)) \gg 16, \\
&\Delta R_2^2 = S \circ L_2((R_{2L}^1 \oplus S_{6H}^0)||y) \oplus S \circ L_2((R_{2L}^1 \oplus \Delta R_{2L}^1 \oplus S_{6H}^0 \oplus \Delta S_{6H}^0)||y), \\
&(R_2^2 \gg 1) \boxplus (S_{6H}^0 \ll 4) = ((R_2^2 \oplus \Delta R_2^2) \gg 1) \boxplus ((S_{6H}^0 \oplus \Delta S_{6H}^0) \ll 4), \\
&\Delta S_{9L}^0 = \Delta R_{2H}^2, \\
&\Delta S_{7H}^0 = \Delta R_{2L}^2.
\end{aligned}$$

Based on the round update function, it is not difficult to observe that the above equations are used to ensure that $\Delta S_{15}^t = 0$ for $t \in [1, 3]$ and that the difference in FSM will be cancelled after three clocks.

To solve the above equations, the authors used an optimized exhaustive search. In short, they first loop for $(S_{5H}^0, S_{11L}^0, \Delta S_{5H}^0)$ and derive ΔS_{8L}^0 . Then, they loop for $(y, S_{6H}^0, \Delta S_{6H}^0)$ to derive $(\Delta S_{7H}^0, \Delta S_{9L}^0)$. Finally, they loop for (S_{9H}^0, S_{7L}^0) to derive S_{12L}^0 to satisfy y. More details can be referred to [BM20]. **It is now obvious that they did not exploit the relations between the XOR difference and modular difference to solve the above equations.**

Based on the above strategy, they succeeded in finding several solutions to the input difference. Then, based on similar sampling techniques discussed above, they finally identified an input difference which can lead to a distinguisher for 26 rounds of ZUC-256.